

May 2019

Telenursing RoboPuppet

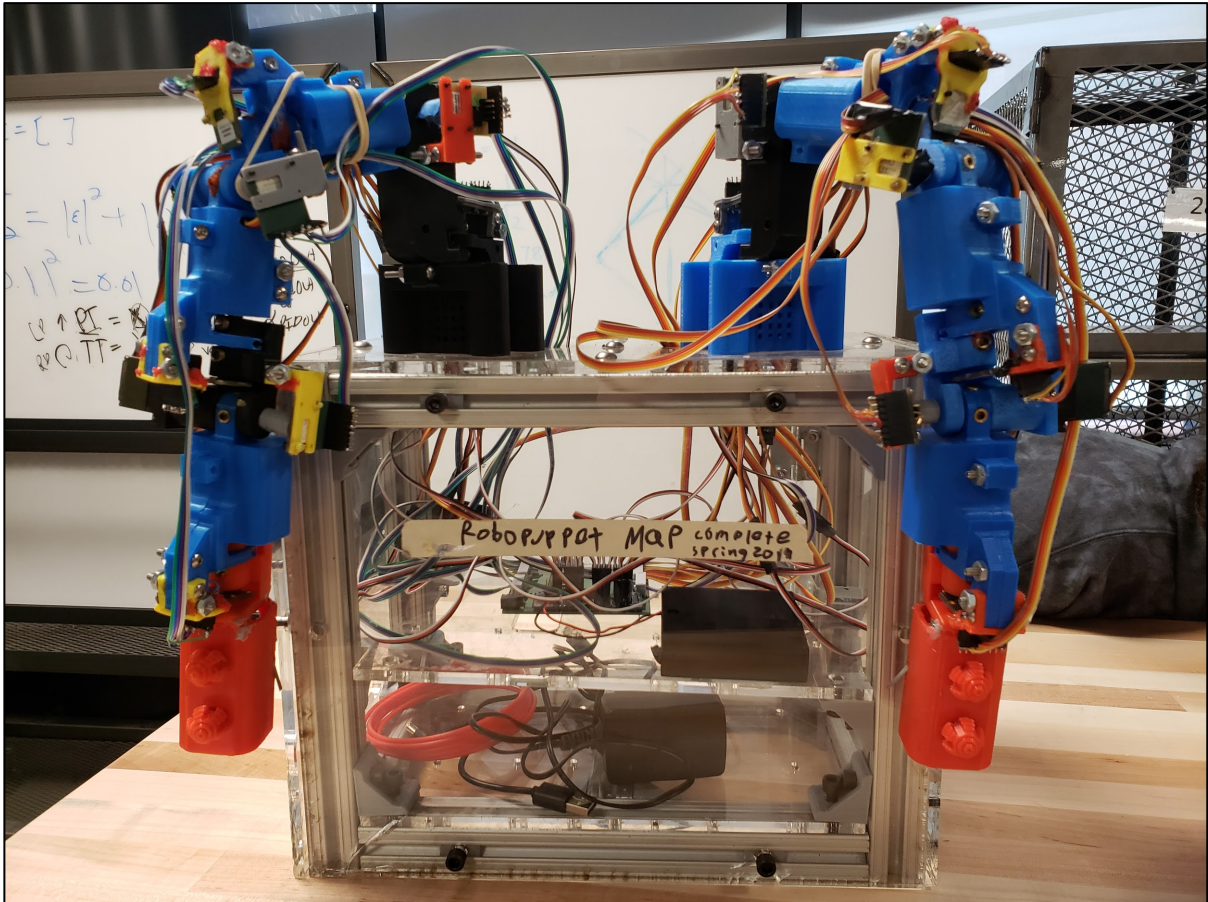
Grant Clark Zahorsky
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

Repository Citation

Zahorsky, G. C. (2019). *Telenursing RoboPuppet*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/7068>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.



Telenursing RoboPuppet

A Major Qualifying Project Report in partial fulfillment of the requirements for the Degree of Bachelor of Science in Electrical and Computer Engineering, Mechanical Engineering, and Robotics Engineering

Submitted by:

Zachary Caplin (ME)

Ryan Kennedy (ECE)

Grant Zahorsky (RBE)

Advisor:

Professor Zhi (Jane) Li

Professor Maqsood Mughal



WPI



Acknowledgement

Our team would like to express our most sincere appreciation to the following individuals. Without their knowledge, enthusiasm, and dedication, the success of our project would never have been possible.

Christian Elzey, WPI: For helping with troubleshooting the circuitry for the RoboPuppet and design consultation

Nicholas Hollander, WPI: For help with navigating the KiCad PCB development software and making sure the PCB was as robust as possible.

Prof. Maqsood Mughal, WPI: For being our Electrical and Computer Engineering Advisor

Professor Jane Li, WPI: For being our primary advisor and all of her help

Vignesh Manoj, WPI: For his help and involvement in every state of this project and the major help in the communication with the project and Baxter

Ozan Akyildiz, WPI: For helping with the general design and interaction of our system

Katherine Crighton, WPI: For help with the ordering of all of our needed parts

Evan Stelly, WPI: For his input with the mechanical design of the robot in its early development

Manon Miller, WPI: For helping with the entire assembly process of the RoboPuppets

The RBE Department, WPI: For their general help and involvement in our project

The ME Department, WPI: For their general help and involvement in our project

The Staff of Washburn Shops, WPI: For their help and involvement with laser cutting and manufacturing of the metal components

Abstract

The Tele-Robotic Intelligence Nursing Assistant (TRINA) assists nurses working with infectious patients by performing highly repetitive tasks. The project focused on fabricating a new version of the 2017-2018 Telenursing RoboPuppet through the addition of motors and sensors that provide haptic feedback to the user while simultaneously resisting gravity when the user releases the device. The addition of a simulation environment to visualize the forward/inverse kinematics as well as fully functioning hand controls for TRINA allows for a more user friendly working system. This article discusses the process of the fabrication and implementation of the circuitry, hardware, and programming for the first iteration of the active version of the Telenursing RoboPuppet.

Table of Contents

| | |
|---|----|
| Aknowledgemnt | 1 |
| Abstract | 2 |
| 1. Introduction | 7 |
| 2. Background | 8 |
| 3. System Overview | 9 |
| 3.1 Mechanical Overview | 9 |
| 3.2 Electrical Overview | 9 |
| 3.3 Software Overview | 10 |
| 4. Design and Methodology | 11 |
| 4.1 Mechanical Components | 11 |
| 4.1.1 Arms | 11 |
| 4.1.2 Hands | 12 |
| 4.2 Electrical System | 20 |
| 4.2.1 Sensors | 20 |
| 4.2.2 Motors | 26 |
| 4.2.3 Circuit Boards | 30 |
| 4.3 Software Analysis | 36 |
| 4.3.1 High Level Overview | 36 |
| 4.3.2 Simulation Software Architecture | 36 |
| 4.3.2.1 Kinematics | 36 |
| 4.3.2.1.1 Reference Frames | 36 |
| 4.3.2.1.2 Forward Kinematics | 39 |
| 4.3.2.1.3 Inverse Kinematics | 44 |
| 4.3.2.1.4 Gravity Compensation | 47 |
| 4.3.2.2 Visualization and Control Interface | 47 |
| 5. Testing | 51 |
| 5.1 Mechanical Testing | 51 |
| 5.2 Electrical Testing | 51 |
| 5.2.1 Encoder Testing | 52 |
| 5.2.2 Motor testing | 54 |
| 5.2.3 PCB Testing and Troubleshooting | 55 |
| 5.3 Software Testing | 55 |
| 6. Recommendations | 57 |
| 6.1 Mechanical Recommendations | 57 |
| 6.2 Electrical Recommendations | 58 |
| 6.3 Software Recommendations | 59 |
| 7. Appendices | 60 |
| 7.1 References | 62 |

List of Figures

| | |
|--|----|
| Figure 1: Part 5 before and after it was slimmed for user friendliness | 11 |
| Figure 2: Limit Switch | 13 |
| Figure 3: Hand control graph | 14 |
| Figure 4: Potentiometer | 14 |
| Figure 5: Initial Hand Design | 15 |
| Figure 6: Internal Initial Hand Design | 16 |
| Figure 7: Second Design Comparison | 17 |
| Figure 8: Version Three Hand Design | 18 |
| Figure 9: Final Hand CAD | 19 |
| Figure 10: Final Hand Design Parts | 19 |
| Figure 11: Final Hand Design and Mounting | 20 |
| Figure 12: Passive Potentiometer | 21 |
| Figure 13: Arduino Mega 2560 and PCB Shield | 22 |
| Figure 14: Passive RoboPuppet PCB layout | 22 |
| Figure 15: Hall Effect Header Pins | 24 |
| Figure 16: Address Configuration | 25 |
| Figure 17: Operation Configuration | 26 |
| Figure 18: 16 Channel I2C Driver | 27 |
| Figure 19: Driver Breakout Board | 28 |
| Figure 20: External Power Supplies | 29 |
| Figure 21: Capacitor | 30 |
| Figure 22: Hand Control Design | 31 |
| Figure 23: Active KiCad | 33 |
| Figure 24: Final PCB Layout | 34 |
| Figure 25: Bare Circuit Board | 35 |
| Figure 26: Fully Connected Active PCB | 35 |
| Figure 27: Cartesian Reference Frame | 38 |
| Figure 28: Output of Transform.m | 44 |
| Figure 29: 3D Virtualization | 48 |
| Figure 30: Forward Kinematics Control Interface | 49 |
| Figure 31: Inverse Kinematics Control Interface | 49 |
| Figure 32: 5V and 3.3V Operations | 52 |
| Figure 33: Confirmation of I2C Communications | 52 |
| Figure 34: Breadboard Layout of Hall Effect Encoder | 53 |
| Figure 35: Breadboard Testing I2C Servo Driver | 54 |
| Figure 36: Wire Connection from Sensors to PCB | 55 |
| Figure 37: I2C Multiplexer | 58 |

List of Tables

| | |
|------------------------------------|----|
| Table 1.1 Naming Conventions | 37 |
| Table 1.2 DH Parameters | 39 |
| Table 1.3 Linkage Lengths | 43 |

1. Introduction

A nurse's duty to help those in need does not stop when patients are infected with a contagious disease. Last year, an MQP tackled this issue through the use of a project developed at Duke University as the basis for their research. With a direct focus on the Tele-Robotic Intelligent Nursing Assistant (TRINA), referred to as Baxter, to safely and efficiently combat infectious diseases, the control system developed last year, affectionately referred to as the "RoboPuppet", was improved upon. The two main objectives of this year's project was to design a slimmer, more manageable RoboPuppet as well as develop a complete simulation of Baxter inside of the Matlab programming environment. By adding functioning hand controls to the RoboPuppet, as well as motors and hall effect encoders to each of the joints, the arms could be more efficiently used and controlled. The hand controls allow the user to be able to control Baxter's end effectors while the joint motors allow the RoboPuppet to make use of the forward and inverse kinematics of the arms. This addition shifts the focus of the user from operating the robot to helping the patient in more effective and efficient ways. The forward and inverse kinematics were solved not only for the RoboPuppet, but also for Baxter by creating a table of the DH parameters of each robot. The team was successful in creating a more usable RoboPuppet as well as developing a useful simulation environment in which the kinematics and gravity compensation could be accurately tested.

2. Background

Last year, the MQP team developed a new and innovative way of controlling Baxter's two arms through the use of a 3D printed puppet, affectionately referred to as the "RoboPuppet". The RoboPuppet is made to be an intuitive control system for nurses to care for patients or to perform repetitive tasks. This system was created in order to assist nurses in controlling Baxter and make their work much more effective and efficient.

This year, our team was tasked with improving on last year's system to make it even more efficient and user-friendly. This was done through the modification of last year's passive robot by adding various sensors and motors to control each joint of the arm. The motors are embedded into the parts in an interchanging order where they begin integrated into the part themselves to create axial rotation, then are outboard in order to create a translational rotation. This is all done to mirror the design of the Baxter robot.

This project also involved the development of a complete simulation environment that allowed complicated movements and mathematics to be properly tested and visualized. With these modifications and additions, the RoboPuppet is one step closer to aiding nurses in their quests to save lives.

3. System Overview

3.1 Mechanical Overview

Mechanically, this project focused on redesigning and fabricating an intuitive input device for TRINA in the form of the first completed active version of the RoboPuppet. The design of the two 7DOF arms were primarily accomplished by last year's MQP group. Upon review of their design it became apparent that the foresight of the original RobotPuppet design was made to be functional over practical for their users. The goal of the mechanical portion of this year's design was to make the arms more user friendly to people of all sizes, as well as develop an intuitive design for a hand controller that be adapted to both the new active robot and the passive robot.

The past design was difficult for people of a smaller stature to use. The design from last year was made using rapid additive manufacturing, commonly referred to as 3D printing. The parts were individually printed and assembled into two arms of a much higher tolerance than the previous group's project.

3.2 Electrical Overview

For the active RoboPuppet, several components were implemented to a printed circuit board to ensure functionality for joint angle sensing, motor control, and hand control. The Teensy 3.5 microcontroller was used to process the angle data as well as communicate with the servo driver, both via Inter-integrated circuit (I2C) protocol. AMS_AS5048B hall effect encoders were attached to each of the 7 angles on both arms (14 total) to detect the change in angle. This was done by installing a tiny neodymium magnets into the joints, where the encoder would detect the change in magnetic field, associate it with a voltage, and turn that voltage into digital logic via an analog to digital converter. Servos were also included to implement force feedback with the purpose of having the robot maintain position when let go. These 14 servos were connected to an I2C servo driver, which allowed the 14 servos to be driven at once. The Teensy's PWM pins could not have performed this task as the Teensy can only drive up to 12 PWM outputs at a time. The servos were also externally powered, which takes load off the Teensy so it can focus on powering the sensors and hand controls. The printed circuit board designed implemented the Teensy, I2C driver, wire connections for the sensors and hand

controls, a 3.3V regulator to make sure all the hall effect encoders ran at 3.3V, pull-up resistors to help with I2C communication, as well as a barrel jack for externally powering the servos.

3.3 Software Overview

In addition to the physical RoboPuppet that was designed and manufactured for this project, a complete three-dimensional visualization environment was also created. The purpose of this simulation environment was to be able to accurately visualize the movement of Baxter and the RoboPuppet, as well as accurately test the derived equations for forward and inverse kinematics and gravity compensation. MATLAB was used because it facilitates the creation of easy-to-use control interfaces and visuals while maintaining the high level of performance needed to run the simulation. A complete model of Baxter was rendered in the simulation so that the movements could be accurately expressed in real-time situations. All of the kinematics and math for the simulation was done in MATLAB since most of the kinematics revolves around matrix multiplications. This allowed the simulation to run as quickly and as efficiently as possible, while still retaining the easy-to-understand visuals presented with the simulation.

4. Design and Methodology

4.1 Mechanical Components

4.1.1 Arms

The design of the arms is based on the original Baxter robot. This was to maintain a proper ratio from each section of the arms. The bulky nature of the original design lent much to be desired for future iterations. The parts were assembled into the arm following the instructions of last year's design.

The primary design was handed to us from the previous group. The parts given to us were within the ratio of Baxter's arm joints. We optimized the parts to make the sections more user friendly for the potential users of this device while leaving the lengths untouched. The parts needed to have their "waist" reduced to be as minimal as possible while maintaining their structural integrity. To accomplish this, the parts needed to be individually analyzed for their limiting factor. This is the section of the part that is the weakest and has the highest chance of becoming deformed and breaking. This is easy to find with the weakest point in the part being the thinnest section.

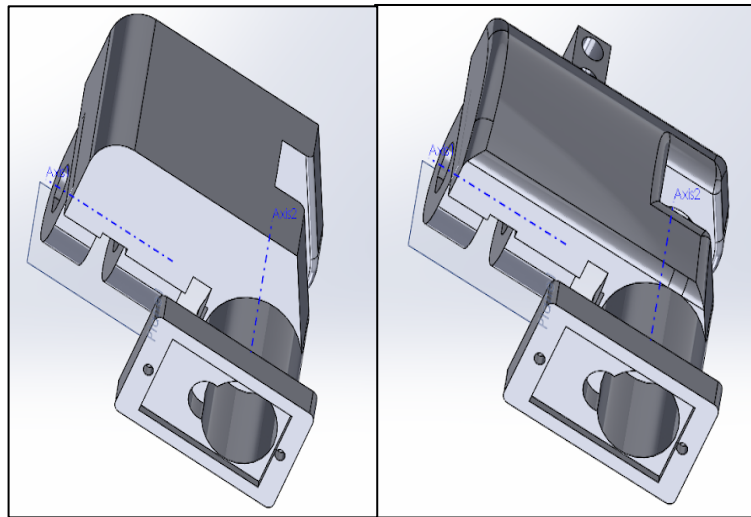


Figure 1 Part 5 before and after it was slimmed for user friendliness.

Once the part's slimming limiting factor is identified, it is a test of creativity to reduce the parts as best as possible. With some parts having a limiting factor of a clean internal sweep, it was apparent that the wall of the part needed to be concentric with the rotation of the internal sweep. These parts were the sections of the arms that carried external motors. These parts rotated

axially. The limiting factor for the other set of parts tended to be where the external structure began to form that would attach the external encoder mounts. These parts were primarily slimmed down using a flat surface with just enough room for a clean fillet feature to the external structures like the encoder mount. The edges of all of the pieces that may come into contact with the user were then touched up by the fillet tool in order to create a smoother surface.

The range of motion for the active arms was based off of the original Baxter robot. The range could not exceed what the Baxter robot could reach itself. If it were to exceed what the actual Baxter robot could do, there would be a distinct error where the feedback would not match the proper rotation of the parts of the remote control. With this in mind, the ratio of the length of each arm and the angle of effect of each joint had to be exact.

4.1.2 Hands

The hand designs for the Baxter RoboPuppet were made with several design requirements in mind. The hand had to be: easy for people of a small stature to manipulate, intuitive for someone not technically adept, comfortable to hold and use, be able to detect the user leaving the device, be able to switch Baxter's hands into different positions, and be able to open and close Baxter's fingers in unison.

The first aspects of the design that were tackled were the most basic functional requirements. The hands needed to open and close the mechanical hands from Baxter, toggle between positions, and sense the presence of the user. Grasping at low hanging fruit, detecting user presence was the initial aspect tackled. This could have been done in one of two ways. It could have been done with the addition of a heat sensing material that would have been adhered to the outside of the hand control and wired internally and would detect a person's body heat, or an internally placed button/switch that would detect a person's hand pressure when gripping the device. I chose the latter, due to its cheaper part cost, ease of coding complexity, and internal mounting process.

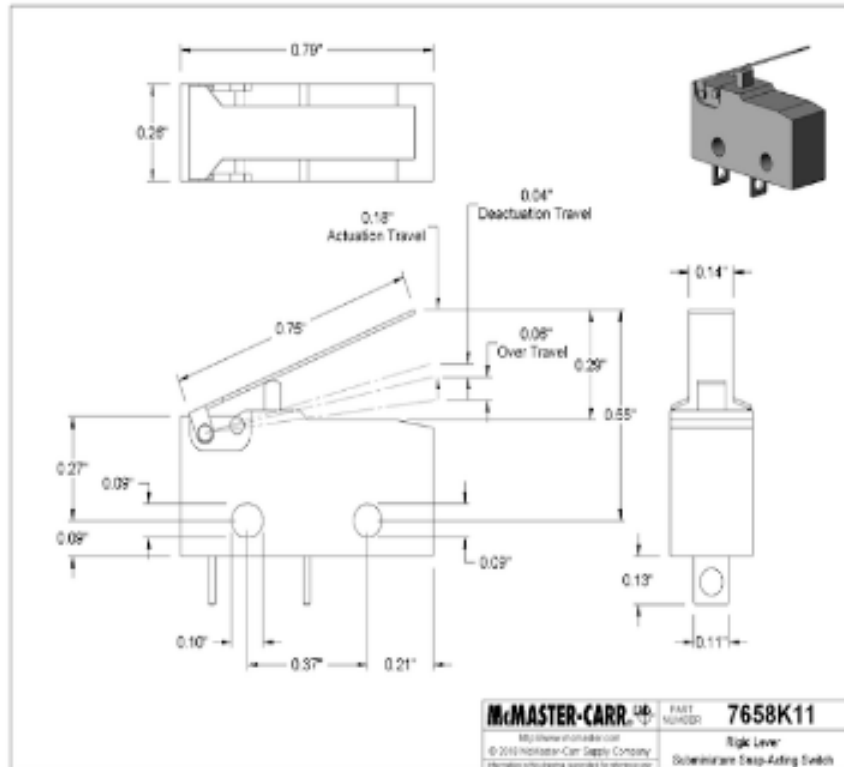


Figure 2 A screenshot of the specs of the limit switch used in the hand controls from McMaster-Carr.

The second aspect that needed to be considered was how the user would utilize the hand control to interact with Baxter’s hands. This involved the ability to open and close all of Baxter’s fingers uniformly, as well as the ability to switch between several hand positions. With Baxter currently having only two hand positions it would have been easy to implement a switch, however with the foresight of the implementation of multiple different hand positions this appeared to not be the most efficient idea. All switches available to our group were not able to be reprogrammed or restructured to switch between more than two hand positions. This meant that we needed a more flexible system. The development of a system that can be adapted for later development on Baxter led us towards using a potentiometer. This allows the different points in the rotation of the potentiometer to determine the position that the hands will be in.

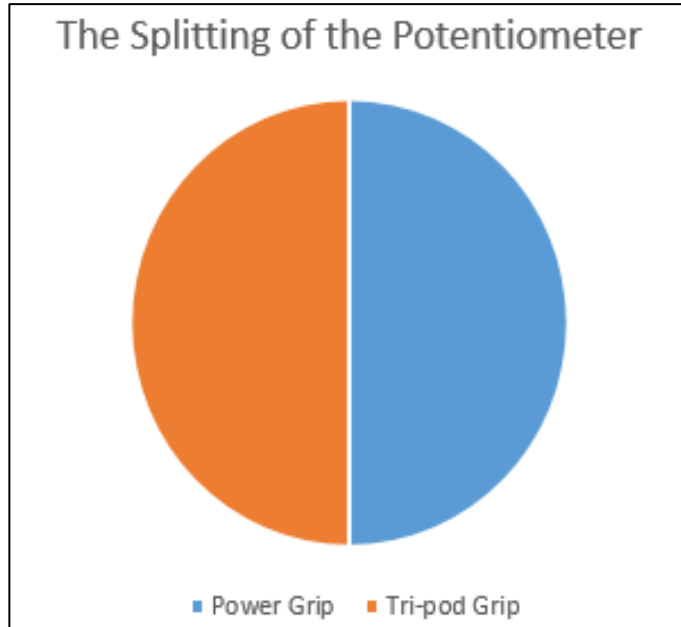


Figure 3 A graph showing how the potentiometer will read its range of motion to determine the hand position of Baxter.

The determination of the hand position was easily adapted into the potentiometer system. Future adaptations of Baxter’s hand design can use this same system for future hand iterations that may be more flexible in their positioning.

Potentiometer Switch with On/Off Control

Max. Resistance, kilohms
✓ 5

Each

ADD TO ORDER

In stock
\$10.34 Each
3980T1

1 added to your order.
Ships today

| | |
|---------------------------|-----------------------|
| Application | Power |
| Component | Switch |
| Number of Turns | 1 |
| Wattage | 0.125 W @ 350 V AC |
| Number of Circuits | 1 |
| Controlled | 1 |
| Switch Starting Position | 1 Off (Normally Open) |
| Switching | |
| Current | 1.5A |
| Voltage | 12V DC |
| For Panel Cutout Diameter | 3/16" |

Figure 4 A screenshot of the McMaster Carr webpage showing the potentiometer used in the hand design.

The final component needed was the ability to open and close the hand in unison. The initial thought was to have a single component operate as a method to open the hand and another component to shut the hand. This involved two small buttons, one that would operate the opening function of the robot and the other would operate the closing function. After creating the part in Solidworks, shown in the figure below, it proved to be an inefficient design with more parts than necessary. The fewer the parts and the less complex the design, the easier the build and maintenance will be for all groups that may be involved with the project.

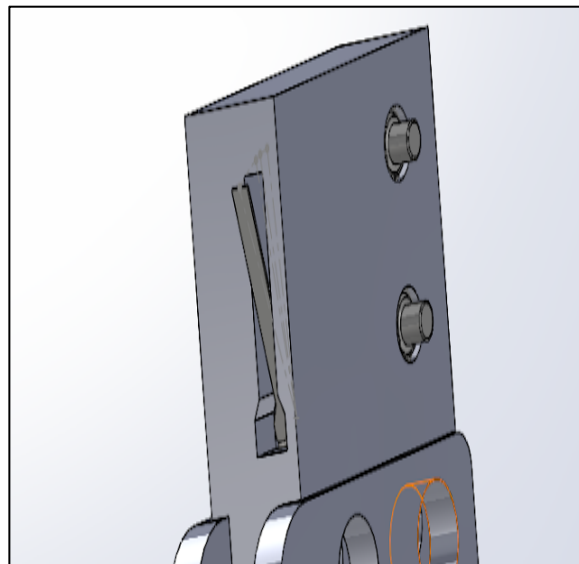


Figure 5 A photo of the initial design with two buttons.

This led to the realization that there only needed to be a single component to open and close the hand. With the component for the hand switch already decided to be a potentiometer, the obvious choice was a second potentiometer. This allows for the hand control to have symmetry that will allow for people to have an easy time operating Baxter.

The final, and most important aspect for comfort of the user was the geometry of the hand control itself. This started with an initial design that did not account for the comfort of the user. It only accounts for the space needed to integrate all of the components. The initial design, shown on the left in figure 6, utilized the left over attachment process from the last group. This involved two large holes that would be used to secure the parts together using pins.

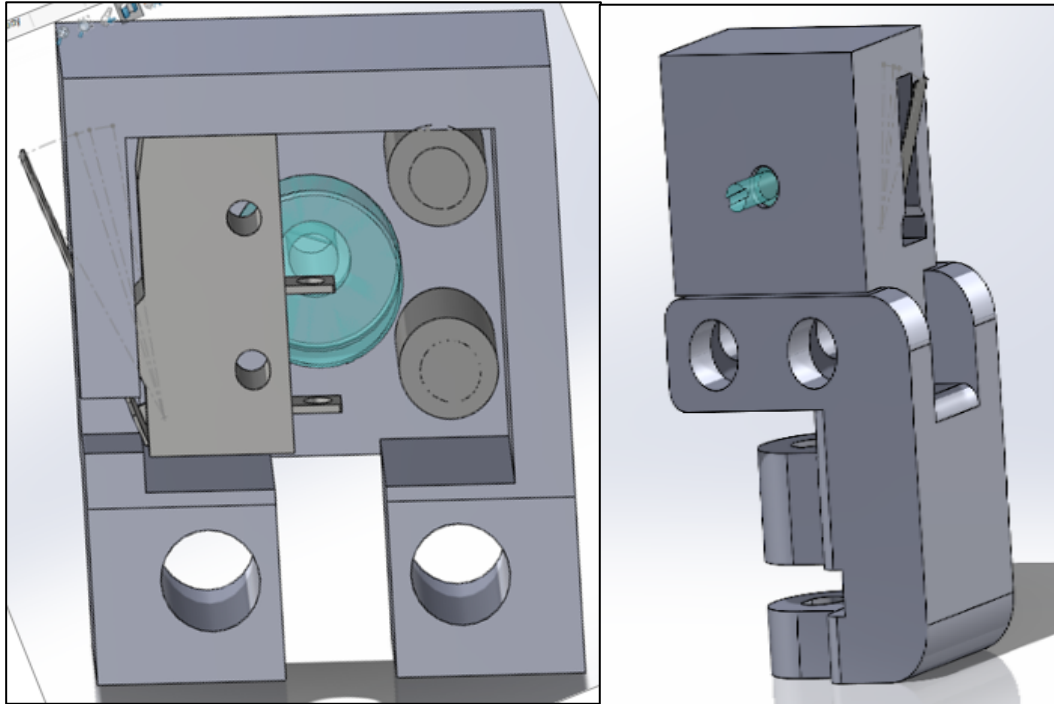


Figure 6 A cross section of the Solidworks of the initial design of the hand control with a single potentiometer, two buttons and a limit switch (Left). The initial hand control design would interact with the 7th digit part of the RoboPuppet (Right).

With dimensions of 1in x 1.5in x 2in this design was too small for even the smallest users to find useful. Once a version was realized using additive manufacturing, the first redesign was implemented. This involved making the design larger in its base footprint as well as taller to allow for a user of even large hands to hold it easily. The second iteration of this design was much larger with a base of 2in x 2in and a height of 3in. This design was also the first to incorporate rounded edges for the comfort of the user, shown below.

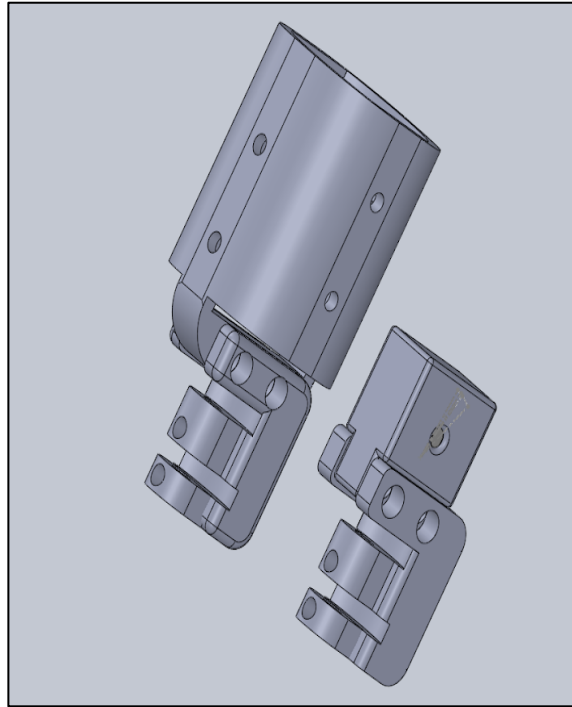


Figure 7 An assembly showing the difference in size from the initial design to the second design.

Once the second hand control was printed, it was clear that we were moving in the right direction but not totally there. The third version of this design was actually printed as different versions of the same part. This was to determine the optimal size for the hands of both larger users and smaller users. Versions were printed at both 1.75in x 1.75in base dimensions and 1.5in x 1.5in base dimensions. This was to find the “goldilocks zone” for the waist dimensions of the hand control. The 1.75in x 1.75in version was more comfortable for users over the height of 5’ 5” while the 1.5in x 1.5in was better suited for people under 5’ 5”, like our project advisor. With this in mind, a final base size was determined at 1.6in x 1.6in. This size was comfortable for both larger and smaller users.

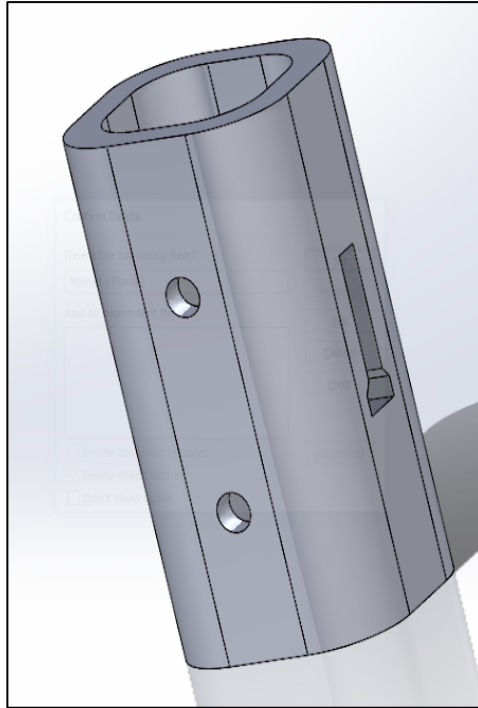


Figure 8 Version 3 of the hand control before it was split open.

With the dimensions of the design realized, and all of the components picked out, it was time to start the building process. There is a slot inserted into the hand control where the user's palm or pointer finger could be (dependent on how they hold the control) with two 4/40th holes for the limit switch to be secured. Then there were two recessed holes implemented 90 degrees from the placement of the limit switch for the potentiometers to screw into. The recess is for the potentiometer to fit snugly into the device. The last feature that was added to the hand controls was a slot for the wiring to easily leave the hand housing. This was put at the base and back of the control to be as out of the way of the user as physically possible. The building process was incredibly difficult. The hand housing did not allow for much room to install the components. This led to several failed attempts to screw in the potentiometers. This process did not need to be this difficult. So a plan was put in place to make a fourth and final iteration of the hand control that was easier for the manufacturer.

The fourth iteration of the hand control design involved splitting the hand control in half and developing a pin system with the same 4/40th screws that were used to secure the limit switch. From experience it is known that there needs to be at least 0.1in of plastic in order to create a solid design that is not prone towards self-destruction. With this in mind, and the

knowledge that these screws are made purely to secure the parts together and will not be resisting any significant force, there were three pin and hole systems put into place within each hand. This led to the 3D printing of two parts for each hand for the first time.

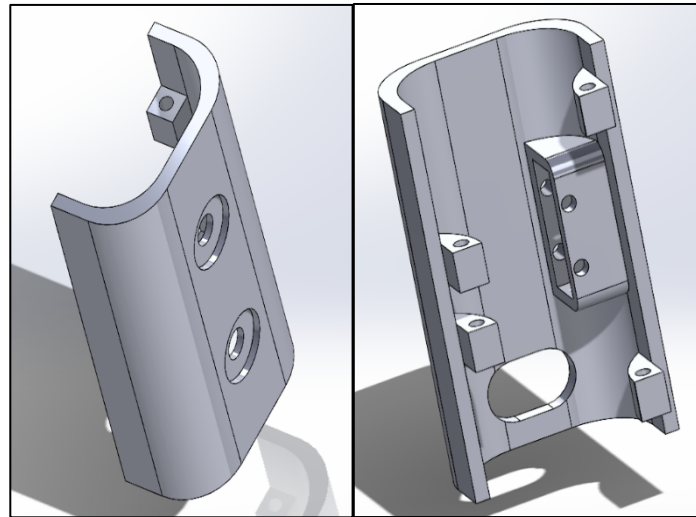


Figure 9 CAD of the final hand design showing the pin system as well as the mounting components

The holes for the potentiometer tended to be a bit small and hard to initially screw in. to fix this the holes were worked with a hot soldering iron to make them soft and malleable. Once they were softened, even a little, it was significantly easier to screw the potentiometer into the part. The potentiometer was then secured using a hex nut that it came with. The limit switch easily slid into place once all support material was removed but would have a lot of trouble if there was even the smallest amount of support material obstructing its placement.

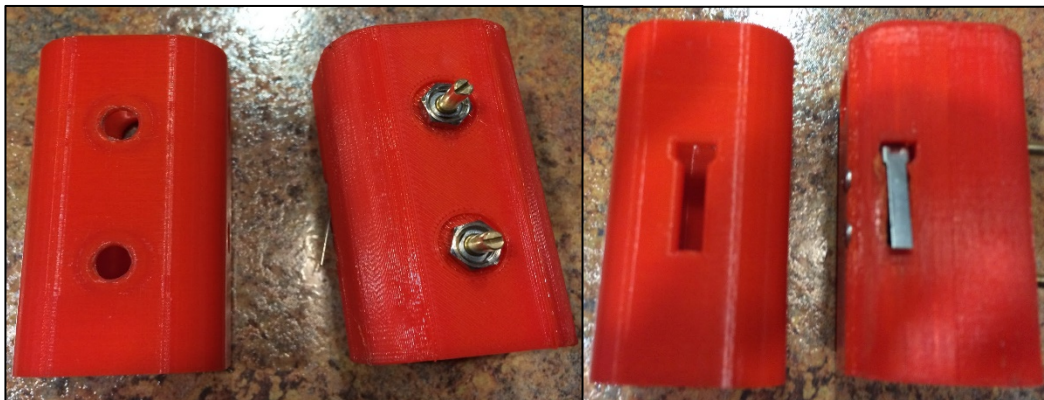


Figure 10 The completed final version of the hand control for the Baxter Robot.

The hand controls then needed to be attached to the 7th digit of the puppet's arms as a single piece. The previous design, as mentioned earlier, involved two 1/4in holes that would secure the hand control to the 7th digit of the RoboPuppet. This was not efficient in a design where we needed to be weight conscious. With this in mind the entire system was redesigned to be two flat components that would be friction welded/epoxied together. This involved redesigning the top of the 7th digit to be flat to allow another component to be secured to it.

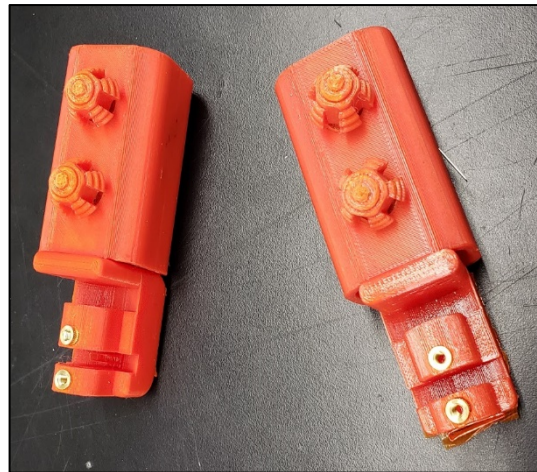


Figure 11 Completed hand controls with end parts attached to it.

4.2 Electrical System

This chapter will discuss the electrical design and implementation of electrical subsystems in the Passive and Active RoboPuppet arms. This section will involve the sensing capabilities, motor implementation for force feedback, and the designing of the printed circuit board to create a more professional looking product.

4.2.1 Sensors

In order to control the much larger Baxter robot, it was important to be able to sense the movement in the controller so Baxter could move accordingly. For the passive and active RoboPuppet versions, potentiometers and hall effect encoders were used, respectively. Each arm had 14 sensors, 7 for each arm to be able to cover the 7 degrees of freedom (DOF) arms. This section will go through the selection of these sensors as well as extra steps required in their implementation.

For the passive RoboPuppet, simple potentiometers were used due to the fact they were small, compact, and easy to use for the primitive design. They were selected for the previous year's passive arm development and were kept for this year's design as well. The potentiometers used were 270 degree analog potentiometers. This means they don't have a full 360 degree range of motion, a small disadvantage but not a catastrophic one. A picture of the potentiometer used can be shown in Figure 12.



Figure 12 Potentiometer used for Passive RoboPuppet angle sensing.

Calibration for these devices were performed last year to make sure turning the potentiometer to a certain input angle matched the output angle, thereby making the measurement reliable. More in-depth information on this calibration process can be found in last year's paper. Assuming everything is calibrated properly, the next step was to recreate the passive arms to be slimmer and a higher quality material, as discussed in the mechanical engineering part of this paper. To connect the potentiometers, a printed circuit board was made as an Arduino Mega shield. This means the board was modelled in the shape of the Mega and connected directly to the Arduino. A picture of the circuit board and Arduino Mega as well as the board diagram in the PCB development software KiCad is shown in Figure 13 and 14.

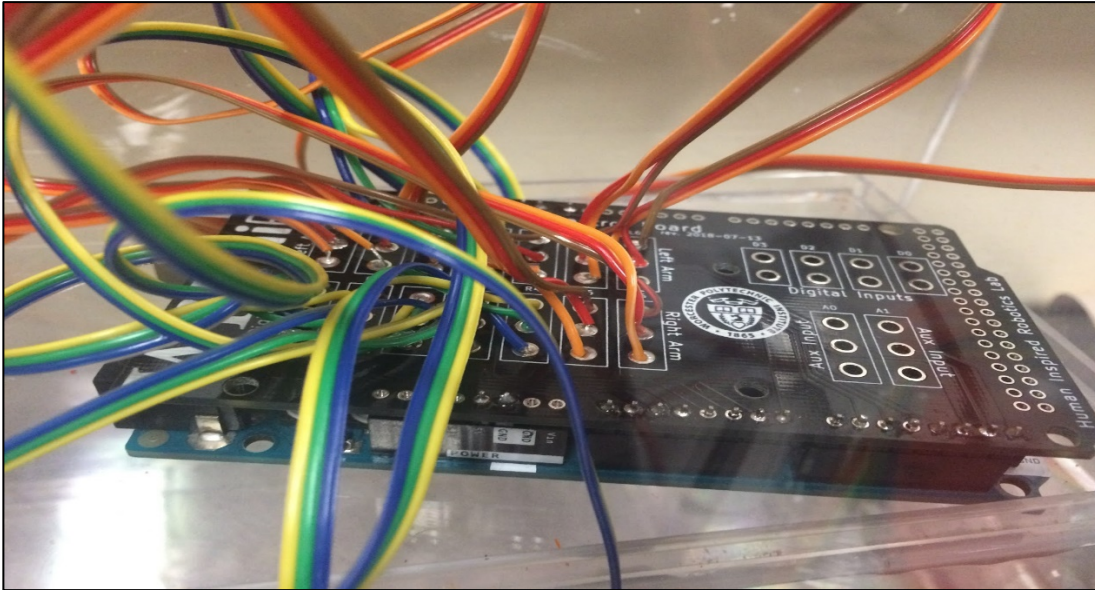


Figure 13 Arduino Mega 2560 and PCB Shield.

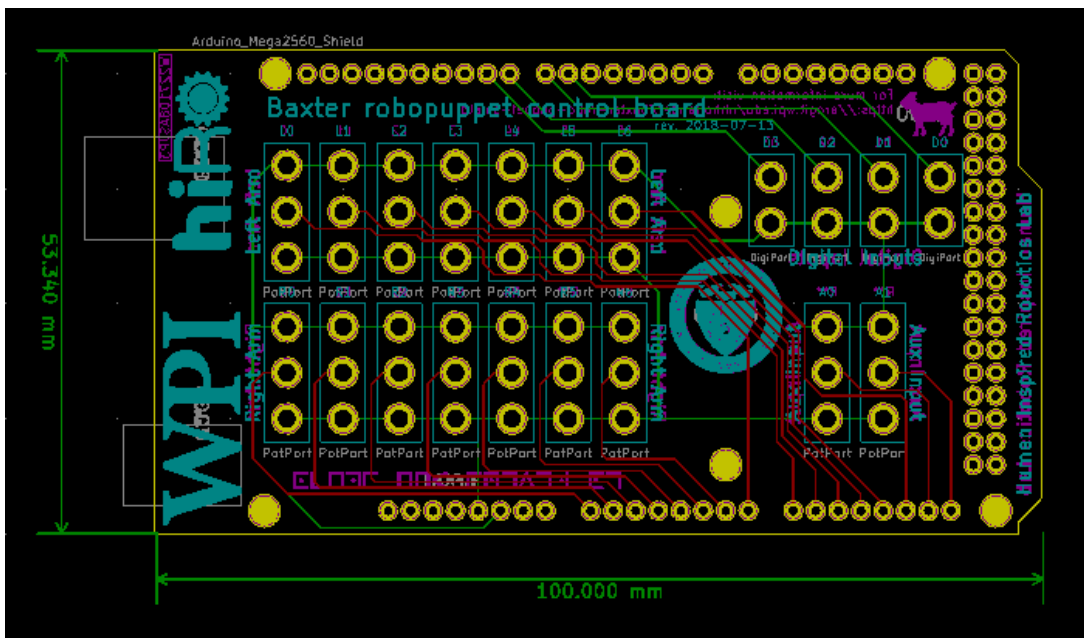


Figure 14 Passive RoboPuppet PCB layout.

This PCB was printed by robotics student, Nicholas Hollander in the summer of 2018. One copy was used to connect an older iteration of the passive arm. Another copy connected the new passive arm wires to the Arduino Mega. The design was simple as the only connections were a 5V voltage line, a ground line, and 14 analog inputs. The design for the active arm

sensors and connections were much more complex, however, involving more than just a voltage, ground and data line.

The active version of the Telenursing RoboPuppet was designed to be a more robust version of the passive arms. This was in regards to the sensing capabilities as well as implementing feedback to the controller. The haptic feedback from the motors will be discussed in a later section. This section will discuss the sensors used to detect joint angle changes so that Baxter can move accordingly. Hall effect encoders are a special type of sensor that measures the magnitude of a magnetic field and associates the strength of a field with a voltage value, leading to a wide range of applications. These applications include proximity sensing, speed detection, and positioning. For this particular project, these types of sensors were used to measure the angle of a joint of a robot arm. To make this possible small, disk shaped neodymium magnets were fit into sockets of the active arms to line up with the sensors. Using this method, turning any joint would turn the magnet, and thereby triggering the sensors.

These sensors were chosen for a multitude of reasons. These include full 360-degree rotation, meaning they lack a dead zone unlike the potentiometers. The sensors also have a 14-bit digital output allowing for a more accurate and robust angle reading. This digital output comes from a built in analog to digital converter on the breakout board for the sensor. Voltage values associated with the detected magnetic field strength was converted to a digital value, that being a 14-bit angle reading. Digital communication is another advantage for board as the signals are less vulnerable to noise that may interfere with communication. Using Inter-integrated Circuit (I2C) communication protocol, data is more reliably transferred from the sensors to the processing unit, the Teensy 3.5 microcontroller. Another strength the hall effect encoders have is they are easier to manipulate mechanically. This comes from the fact the potentiometers are implanted in the passive bot, making it hard to troubleshoot once installed. The hall effect encoders, however, are external and not only can be removed to replace if necessary, but have wires that can be easily removed and replaced. Ribbon cables soldered to male header pins connected to female headers soldered to the encoder breakout boards make this easy implementation possible. Pictures of this connection scheme is shown in Figure 15.

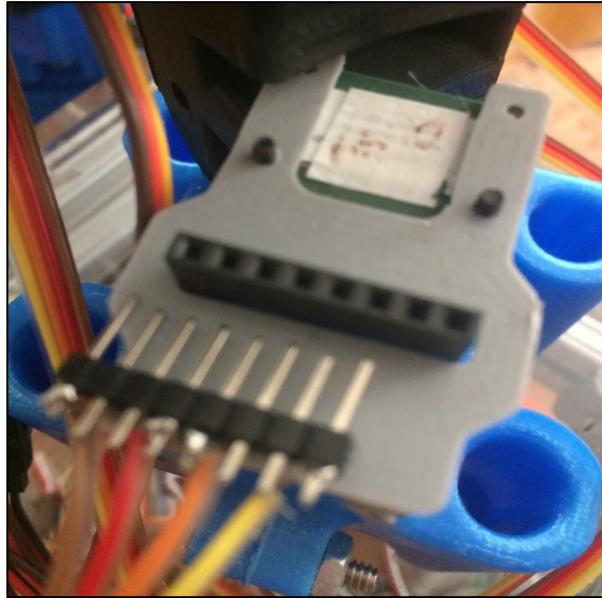


Figure 15 Header pins and connection to hall effect encoder.

There were also some challenges that came with using the hall effect encoders. One important challenge was the cost. Each encoder was approximately 15 -18 dollars apiece, being almost twice the price of the potentiometers. With a substantial budget in possible future iterations of this project, this challenge can be overcome, but the price of these sensors is definitely something to take into account. Another roadblock with these sensors, more specifically the AMS_AS5048B sensors bought last year for the project, was that they are hard to find according to many electrical component vendors like DigiKey, Mouser, and Arrow. Therefore, replacing them is almost impossible. To counter this, similar looking AS5061 hall effect encoders can be used to replace the 5048Bs enough to do the job. More information on the differences and potential challenges with these replacements will be discussed later in the section. Another caveat of the sensors is the size of the breakout boards. On some of the joints, particularly the side joints, the rectangular boards containing the sensors jut out and make use of the puppet a little more difficult and can also interfere with wires draped down the arm. Also the large sized boards can interfere with the mechanical movement of the arms if not fit perfectly. This was mitigated by shaving off some of the corners in order for everything to fit better.

To digitally communicate with the Teensy 3.5 microcontroller, I2C protocol was used. This was an advantageous system compared to other communication systems due to the reduced number of wires necessary. In a typical I2C device, connections are at voltage, ground, data

(SDA), and clock (SCL). These four wires are all that's necessary for a device to be hooked up in I2C. Compare this to Serial Peripheral Interface (SPI) where a Master In Slave Out (MISO) and Master Out Slave In (MOSI) may be necessary in addition to a slave select (SS), clock (SCK), voltage and ground. For this particular application, only MISO would have been necessary as data is only going way. It was still important to reduce wires on the arm wherever possible.

Unlike SPI interface, I2C does not have a slave select function, so it needs another way to determine what device it is "talking" to. This is done by using addressing. An address frame is put at the beginning of a data sequence transmitted by the master device to see if any of its slave devices have a matching address. If there is a match, a low voltage acknowledgement (ACK) is sent and data can be transmitted. For this particular system, the default address of the AS5048B is 0x40. Since unique addresses are needed to communicate with multiple devices at once, there needed to be a way to change addresses to distinguish one sensor from another. Luckily, the AS5048B has pins called A1 and A2, which can be either pulled high or low (to voltage or ground) to change the devices address by 0x01 or 0x02, respectively. This was able to make it so 4 unique sensors could be read at once, with addresses 0x40, 0x41, 0x42, and 0x43. Instead of connecting these pins high on the circuit board, it was easier to connect them on the sensor breakout board. This is shown below in Figure 16.

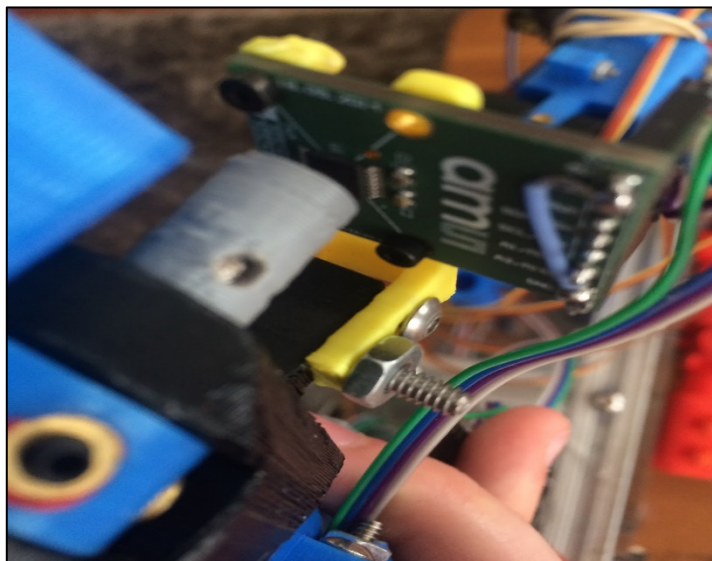


Figure 16 Address Configuration on AS5048B breakout board.

Since there were 14 sensors, there needed to be something else that accounted for more sensors. The solution was another feature of the AS5048B, that being one-time programming

(OTP). A built in function in a user created Arduino library specifically for these sensors allows a user to manually change the address of a AS5048B. This is a permanent function as the new address is burned into the chip and cannot be changed after the one time. After this process was done to 10 of the 14 chips, 14 unique addresses were available, allowing the Teensy 3.5 to decipher between each slave device

An important factor to consider was the voltage drawn by the sensors. The AMS_AS5048B sensors have a capability to be configured in a 3.3V setting or a 5V setting. Below is a diagram from the datasheet showing the difference of each configuration.

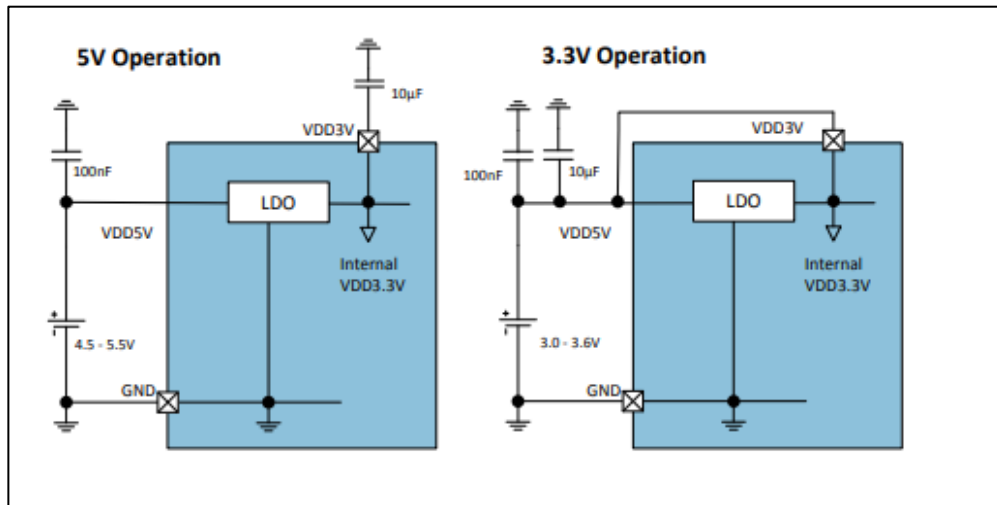


Figure 17 5V and 3.3V operation configurations for AS5048B.

On the breakout board, capacitors C1 and C2 and installed, so the board is in 3.3V operation automatically. This works out because a lower voltage level is safer for the system in general as the Teensy 3.5 pins are 5V tolerant, but have a rating barely above 5V, so having 3.3V on each I2C pin is the safer option. On each hall effect encoder, the 5V pin and 3.3V pin were shorted with solder, being right next to each other. This ensured the system being in 3.3V operation. On the circuit board connecting the Teensy to the sensor wires, a 3.3V regulator was implemented to make sure the Teensy was only sending 3.3V to each sensor, making sure the encoders weren't getting fried themselves.

4.2.2 Motors

The main difference between the active and passive arms is the use of haptic feedback. More specific information on the calculations involved with gravity compensation can be found

in the software chapter of this report. Electrically, the feedback was implemented with the use of servos. The previous year's project selected the Towerpro MG90D servos specifically as a result of their ability to rotate continuously, relatively compact size, and ability to produce large amounts of torque for their size.

To drive all 14 motors on the active RoboPuppet, a PCA9685 16 channel servo driver was used. The Teensy 3.5 has digital pulse width modulation (PWM) ports able to drive servos. However, the Teensy 3.5 and 3.6 can only control 12 PWM inputs at a time. Therefore, using these pins would not be applicable for the RoboPuppet application. To solve this problem, I2C is used once again, this time with the slave device being the servo driver. This device is shown in Figure 18.

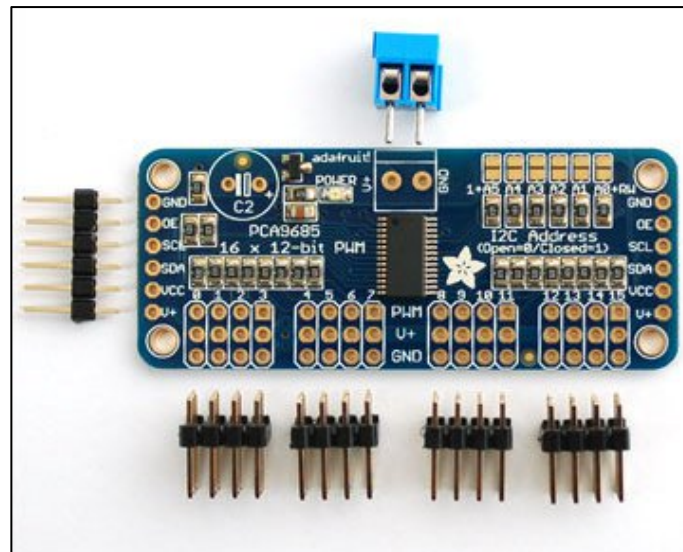


Figure 18 PCA9685 16 Channel I2C Driver.

Oddly enough, the default address for the servo driver was 0x40, but changing the address was quite simple. On the top right of the chip, there are address jumpers that can be soldered to configure a new address. This is shown in Figure 19.

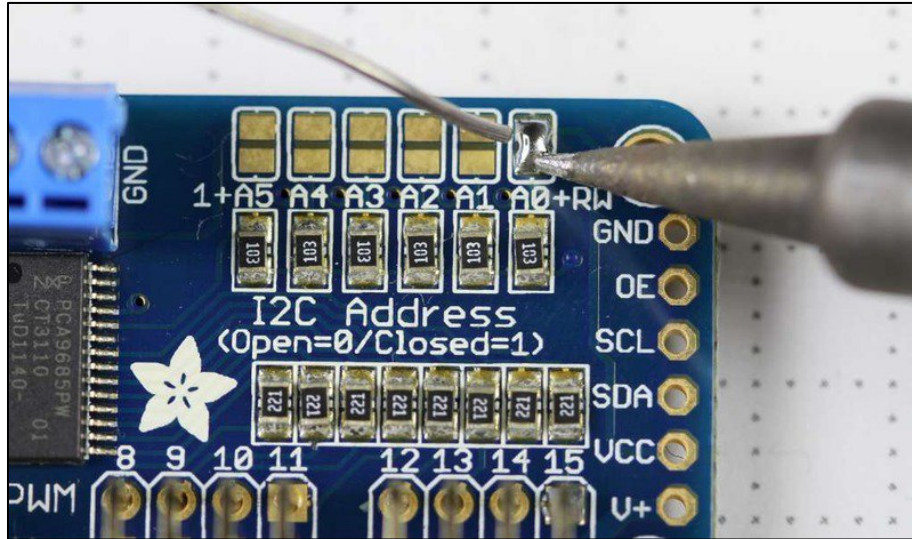


Figure 19 Address Configuration on I2C Driver Breakout Board.

For this project A4 was soldered to create a new address of 0x50 (0x16 higher than 0x40). This was chosen to make sure there was no chance of interference with any of the addresses of the hall effect encoders. The LED also has an address of 0x70 which indicates if the chip is plugged in correctly. All of the I2C devices are connected to the same I2C line in the final version of the circuit board.

To safely implement all 14 servos to the design of the active RoboPuppet, it was a good idea to externally power all of the motors. This is because the Teensy 3.5 is already providing power to 14 hall effect encoders, so to give it the burden of doing that on top of controlling 14 servos would be unwise. Therefore, different methods of externally powering the servos were examined. One method was a wall plug in AC adapter, one that would be plugged into the board via a barrel jack. The other method was a battery pack that had 4 AA batteries. Both ways had strengths and weaknesses. The power jack can be easily plugged into a wall, but reduces mobility of the RoboPuppet. The battery pack does not require a restricting cable to the wall, but the batteries eventually run out of power. In the design, both methods were implemented as options. Both components supply approximately 5V 2A and can be shown in Figure 20 below.



Figure 20 External Power Supplies for Servo Driving.

It is important to know for whoever operates the RoboPuppet that both power supplies cannot be on at the same time, for there is a risk to damage the system. The I2C servo driver makes it so Vcc which takes power from the Teensy and V+ which is the supply power to the servos, do not intersect, so the power supplies of the microcontroller and external supply are completely independent. Also installed on the servo driver is a space for a through hole capacitor. According to Adafruit, the creators of the board, it is helpful to have a capacitor with a value of $n * 100\mu\text{F}$ with n being the number of servos in the system. Since 14 servos were used, a 1500 μF electrolytic capacitor like the one in Figure 21 was chosen.

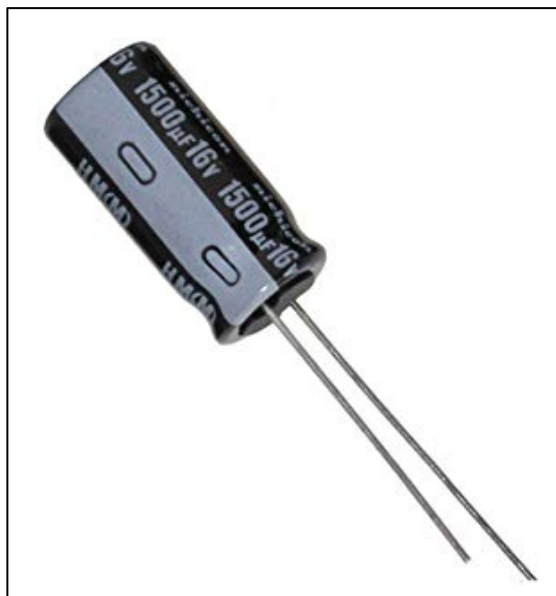


Figure 21 1500uF electrolytic capacitor.

The purpose of this is to compensate for the many servos relying on one power supply. When the servos move, the power supply may dip and fluctuate and behave erratically. A capacitor that can account for every motor should solve this potential issue.

4.2.3 Circuit Boards

To combine all of the electrical components of the active RoboPuppet, a printed circuit board was designed and manufactured. This board contained and connected the components so that the whole RoboPuppet system could operate properly. This section discusses important components of the board not discussed earlier as well as the steps and process of creating this system, from breadboarding to a more professional looking printed circuit board.

A microcontroller is necessary for any version of the RoboPuppet to process data and perform other tasks. Important factors when choosing a microcontroller include the processing power, memory capabilities, and number of specific ports, among other things. The passive bot used an Arduino Mega, not powerful or compact chip, but one that has enough analog ports to perform the low level processes. Arduinos in general are also well supported by not only manufacturers but the community as well. For the active bot, the Teensy 3.5 was chosen for many reasons. One being the 256KB of RAM allowing for fast processing power. The ability to connect to a breadboard or PCB was also favorable to make it easy to test and install. The team from last year chose a Teensy 3.6, which has small differences from the 3.5. The 3.5 was chosen

due to it being five dollars cheaper, as well as all the pins being 5V tolerant rather than the 3.6 having only 3.3V tolerant pins. For any Teensy microcontroller, software called Teensyduino was available. This allows a Teensy board to be programmed with Arduino code, combining the robustness of the Teensy as a unit with the open source documentation of Arduino. Overall the Teensy 3.5 was an excellent choice to be the microcontroller tasked with conducting processes for the active RoboPuppet.

In order to provide hand controls for Baxter, hand units were printed and installed onto the ends of the passive and active arms. On these hand control units, two potentiometers were installed in addition to a limit switch. A picture of this layout can be seen in Figure 22.

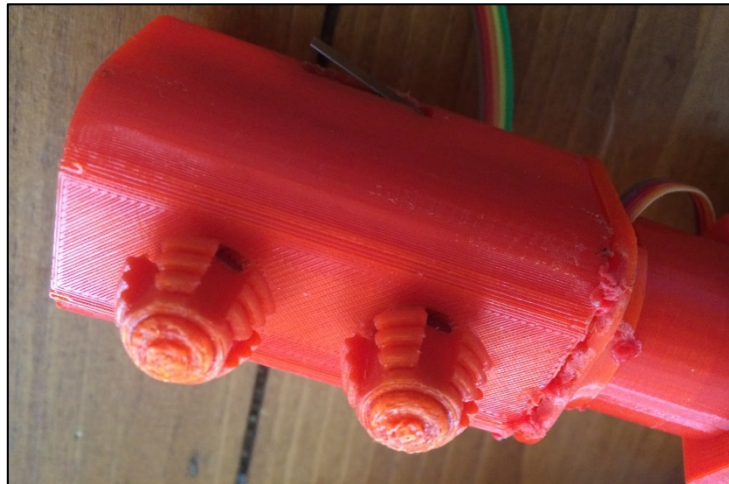


Figure 22 Hand control design.

The limit switch had one side connected to analog data, the other connected to a voltage line. Upon triggering the switch, the two sides are connected electrically, creating the analog input to pull high. In the case of the RoboPuppet, this can be used to indicate that the device is being operated, temporarily disabling the force feedback, allowing the arms to be manipulated. Each hand had three analog inputs, creating six total pins on the Teensy 3.5 needed to be set aside for analog input. A voltage and ground line also ran through each hand, resulting in 5 total wires per hand. A rectangular connector connected to the PCB with 10 slots was enough to accommodate all of these hand control connections.

Before a printed circuit board could be made, a breadboard design was necessary to make sure the connections between the hall effect encoders and the Teensy were reliable. The Teensy

had to detect the presence of an I2C device using an example Scanner Arduino file. If the Teensy could detect all of the sensor addresses at once, then the breadboard design was successful. Before a final, optimized connection scheme was determined a couple components were considered to mitigate any potential problems regarding I2C connection and wire connection in general. I2C buffers were originally implemented to boost a signal over distance. I2C signal strength has been known to deteriorate over long distances and this challenge was taken into consideration. However, after testing it was determined that these amplifiers were not necessary to the design and were therefore removed to maintain simplicity. I2C communication was still reliable even with the longest wires on the active RoboPuppet. If future designers want to incorporate these buffers to improve robustness of the communication, they are welcome to do so, however these are not vital. Another conclusion from breadboard testing was that all 14 sensors could be tied to one I2C line. Having only one SDA and one SCL line for all the sensors allowed for the design to be much simpler and easier to troubleshoot. It was possible to split the I2C ports as the Teensy 3.5 had more than one set, but the design decision to put every I2C device on the same line proved to be effective.

To produce a professional looking circuit board, as recommended by last year's RoboPuppet team, a printed circuit board was produced. There are many options when it comes to software available to create PCB schematics and boards. Among those are Autodesk's Eagle, Altium, and KiCad. Any of these softwares would have been good choices for software, however this project went with KiCad for a few reasons. One being the software being completely free and open sourced, unlike Eagle which is considerably expensive. There was also more comfort with using KiCad. The issue came from the fact that KiCad is not necessarily user friendly. That being said, there is no known PCB software that can be described as "user friendly" and what is considered the best choice depends on the user and their experience.

The first step in PCB creation is to create a schematic. This is simply a basic drawing of how all of your components will connect. In this stage, it doesn't matter where your components are in the workspace, just as long as everything is connected the way you want. For this project, many different versions of schematics were drawn up. The final schematic can be shown below in Figure 23.

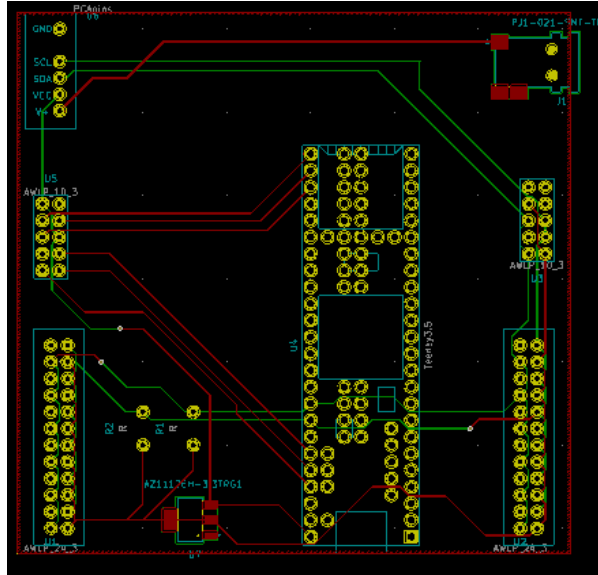


Figure 24 Final PCB Layout.

On this final board layout are red and green lines called traces. These are the wires that connect everything on the PCB. Red traces go on the front of the board, while green traces go on the back of the board. Having two paths of wires allows for more compact spacing and makes board design a lot easier. Traces with the same color cannot intersect, so drawing the wires on the board becomes a little bit of a puzzle. In some cases, vias are necessary to connect a top trace to a bottom trace if the board calls for it. This particular design had 3 vias in total. To ensure all the ground pins for every component was connected a ground plane was used. This plane at the center of the board grounded everything to each other. Another important step was to edge cut the board, finalizing the size. It was ideal to minimize the size of the board in order to minimize the manufacturing cost overseas. As long as it is below 100mm x 100mm in area, the board would stay at a fixed price of 2 dollars. After the PCB was finalized, it was ordered off the internet and delivered a few business days later. A copy of a bare PCB can be seen in Figure 25.

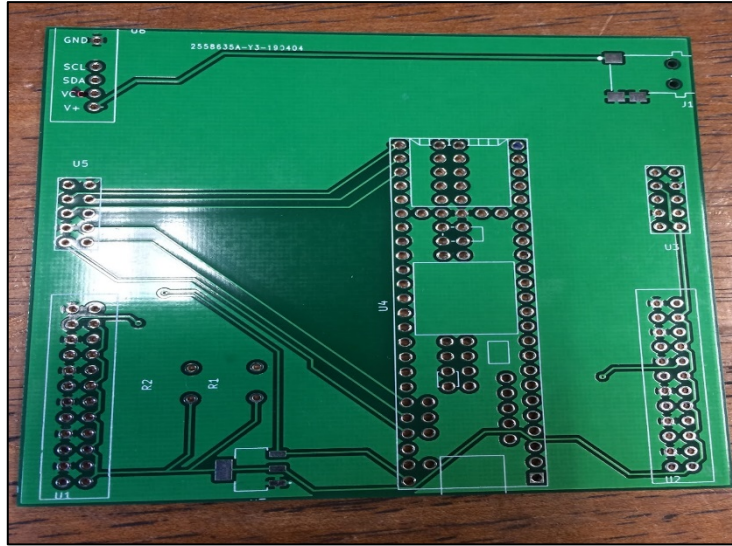


Figure 25 Bare Printed Circuit Board.

After these boards were delivered, the components including the Teensy, servo driver, and more, were then soldered onto the board. The circuit board was then tested after everything was connected. A picture of everything on the circuit board level can be seen in Figure 26.

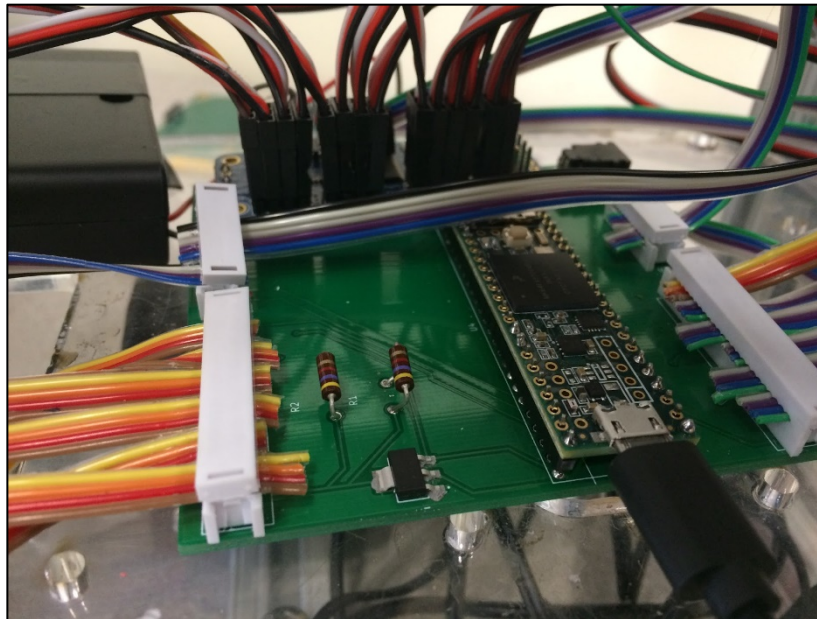


Figure 26 Fully Connected PCB for Active RoboPuppet.

4.3 Software Analysis

4.3.1 High Level Overview

In addition to the physical RoboPuppet that was designed and manufactured for this project, a complete three-dimensional visualization environment was also created. This simulation environment was developed so that a user could properly visualize the movement of Baxter and the RoboPuppet as well as accurately test the derived equations for forward and inverse kinematics and gravity compensation. Due to the usability and functionality of the MATLAB programming software and language, the simulation environment for this project was written completely in MATLAB. This programming language facilitates the creation of easy-to-use control interfaces and visuals while maintaining the high level of functionality and efficiency needed to run the simulation.

4.3.2 Simulation Software Architecture

The architecture for the three-dimensional simulation environment can be divided into three different parts: the kinematics of the robot, visualization of Baxter, and control interface. In doing so, a closer and more in-depth look can be taken into exactly how the software functions.

4.3.2.1 Kinematics

This chapter discusses the forward and inverse kinematics of the Rethink Baxter robot, as well as the implemented gravity compensation policy used in the three-dimensional visualization software.

4.3.2.1.1 Reference Frames

Since the basis for a successful simulation environment is reliant upon the correct derivation of the kinematics of the robot, reference frames must be set for each link of each arm of the robot. In order to do so, the two arms must be analyzed so that the most useful reference frames for each arm can be set. Fortunately, the two arms are exact mirrors of each other. This allows the following derivations to be much simpler as only one derivation must be done, rather than two.

The two arms both have seven degrees of freedom and seven revolute joints. Each arm has an offset shoulder and elbow joint (two degrees of freedom) as well as an offset wrist joint

(three degrees of freedom). A set of naming conventions was also created in order to correctly label and thereafter refer to each joint. These naming conventions are found in Table 1.1:

| <i>Joint Name</i> | <i>Motion</i> |
|-------------------|----------------|
| S ₀ | Shoulder Roll |
| S ₁ | Shoulder Pitch |
| E ₀ | Elbow Roll |
| E ₁ | Elbow Pitch |
| W ₀ | Wrist Roll |
| W ₁ | Wrist Pitch |
| W ₂ | Wrist Yaw |

Table 1.1 Naming Conventions for Each Joint.

Now that each joint can be properly referred to, the reference frames for each arm can be made. As stated before, each arm is an exact mirror of each other. Baxter is designed with this in mind so that the Denavit-Hartenberg (DH) Parameters are identical for each arm. As such, only the derivations for Baxter’s right arm will be shown. In doing so, redundancy is lessened, and the actual derivation of the kinematics is emphasized. Figure 1 portrays the reference frame of Baxter’s left arm using standard Cartesian coordinates.

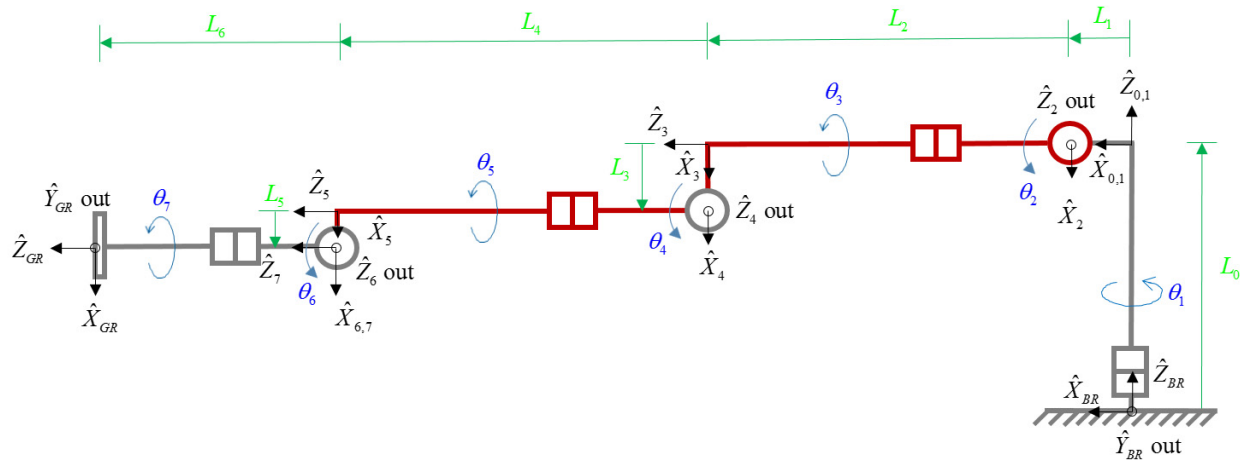


Figure 27 Baxter's Right Arm's Cartesian Reference Frame. ^[1]

Once a proper reference frame has been created for the arm, a table of DH parameters can be solved for by looking at the position and orientation of each joint (see Table 1.2). This table of parameters is necessary not only to accurately and completely solve for and derive the kinematics equations for the arm but also to find the relation between the end-effector and the base.

In order to do so, four variables are assigned to each joint on the arm. Firstly, the variable a (sometimes referred to as r) is assigned to describe the length of the common normal. For revolute joints, such as the ones on Baxter, this is the radius about the previous z -axis. Next, α is assigned to describe the angle about the common normal. This is calculated by the finding the angle from the old z -axis to the new z -axis. These two variables are used to relate the size and shape of the link being analyzed to the arm itself. The other two variables, d and θ , are used to relate the relative position of the links to the arm itself. The variable d describes the offset along the previous z -axis to the common normal. In the case of Baxter's arms, joints three and five are variable since they are offset from the previous joint. Finally, the variable θ describes the angle about the previous z -axis, from the old x -axis to the new x -axis.

| Joint (<i>i</i>) | a_{i-1} | α_{i-1} | d_i | θ_i |
|--------------------|-----------|----------------|-------|---------------------|
| 1 | 0 | 0 | 0 | θ_1 |
| 2 | L_1 | -90° | 0 | θ_2+90° |
| 3 | 0 | 90° | L_2 | θ_3 |
| 4 | L_3 | -90° | 0 | θ_4 |
| 5 | 0 | 90° | L_4 | θ_5 |
| 6 | L_5 | -90° | 0 | θ_6 |
| 7 | 0 | 90° | 0 | θ_7 |

Table 1.2 DH Parameters for Baxter's Seven Degree of Freedom Right Arm.

Now that the DH parameters have been found according to the conventions above, the forward kinematics of the arms can be derived.

4.3.2.1.2 Forward Kinematics

In general, the process for solving the forward kinematics of a serial-chain robot is straight-forward and as follows: given the proper joint angles of the robot, calculate the position and orientation (pose) of the end-effector. The solution consists of a homogeneous matrix T :

$${}^{i-1}_iT = \begin{bmatrix} {}^{i-1}_iR & \{{}^{i-1}_iP\} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where the upper left 3x3 matrix, expressed as R , is the orientation (or rotation) of the end-effector and the upper right 3x1 matrix, expressed as P , is the position of the end-effector. This matrix is found by inputting the DH parameters found in Table 1.2 into the following equation:

$$[{}^{i-1}T_i] = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -d_i s\alpha_{i-1} \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & d_i c\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $c\theta_i = \cos\theta_i$, $s\theta_i = \sin\theta_i$, $c\alpha_i = \cos\alpha_i$, and $s\alpha_i = \sin\alpha_i$ were used as abbreviations.

The solution is found by examining the arm on a link-by-link basis and then finally by multiplying each homogeneous transformation matrix together to find the final position and orientation of the end-effector. To do this, each row of the table of DH parameters is substituted into the above equation. This will produce seven neighboring homogeneous transformation matrices, one for each link: ⁽¹⁾

$$[{}^0T_1] = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [{}^1T_2] = \begin{bmatrix} -s_2 & -c_2 & 0 & L_2 \\ 0 & 0 & 0 & 0 \\ -c_2 & s_2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [{}^2T_3] = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ 0 & 0 & -1 & -L_2 \\ s_3 & c_3 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[{}^3T_4] = \begin{bmatrix} c_4 & -s_4 & 0 & L_4 \\ 0 & 0 & 1 & 0 \\ -s_4 & -c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [{}^4T_5] = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & -1 & -L_4 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [{}^5T_6] = \begin{bmatrix} c_6 & -s_6 & 0 & -L_5 \\ 0 & 0 & 0 & 0 \\ -s_6 & -c_6 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[{}^6T_7] = \begin{bmatrix} c_7 & -s_7 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_7 & c_7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $c_i = \cos\theta_i$, $s_i = \sin\theta_i$, for $i = 1, 2, \dots, 7$ were used as substitutions.

Finally, each homogeneous transformation matrix was multiplied together in order to solve for the final transformation matrix $[{}^0T_7]$.

$$[{}^0T_7] = [{}^0T_1(\theta_1)][{}^1T_2(\theta_2)][{}^2T_3(\theta_3)][{}^3T_4(\theta_4)][{}^4T_5(\theta_5)][{}^5T_6(\theta_6)][{}^6T_7(\theta_7)]$$

To simplify this process, the matrices were separated to denote which matrix pertained to the shoulder of the arm, elbow of the arm, and the wrist of the arm. Not only does this help visualize how the kinematics were being solved, but it also allows each two degree of freedom (2-dof) system and the three degree of freedom (3-dof) system to be separated from each other. To accomplish this, the following calculations were done:

$$[{}^0_7T] = [{}^0_2T(\theta_1, \theta_2)][{}^2_4T(\theta_3, \theta_4)][{}^4_7T(\theta_7, \theta_6, \theta_7)]$$

where:

$$[{}^0_2T(\theta_1, \theta_2)] = \begin{bmatrix} -c_1s_2 & -c_1c_2 & -s_1 & -L_1c_1 \\ -s_1s_2 & -s_1c_2 & c_1 & L_1s_1 \\ -c_2 & s_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[{}^2_4T(\theta_3, \theta_4)] = \begin{bmatrix} c_3c_4 & -c_3s_4 & -s_3 & -L_3c_3 \\ s_4 & c_4 & 0 & -L_2 \\ s_3c_4 & -s_3s_4 & c_3 & L_3s_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[{}^4_7T(\theta_7, \theta_6, \theta_7)] = \begin{bmatrix} -s_5s_7 + c_5c_6c_7 & -s_5c_7 - c_5c_6s_7 & c_5s_6 & L_5c_5 \\ s_6c_7 & -s_6s_7 & -c_6 & -L_4 \\ c_5s_7 + s_5c_6c_7 & c_5c_7 - s_5c_6s_7 & s_5s_6 & L_5s_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The final homogeneous transformation matrix is then calculated and the result is:

$$[{}^0_7T] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x_7^0 \\ r_{21} & r_{22} & r_{23} & y_7^0 \\ r_{31} & r_{32} & r_{33} & z_7^0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

$$r_{11} = ((as_4 - c_1c_2c_4)s_6 - (bs_5 + (ac_1c_2s_4)c_5)c_6)c_7 + ((ac_4 + c_1c_2s_4)s_5 - bc_5)s_7$$

$$r_{12} = ((ac_4 - c_1c_2s_4)s_5 - bc_5)c_7 + ((as_4 - c_1c_2c_4)s_6 - (bs_5 + (ac_4 + c_1c_2s_4)c_5)c_6)s_7$$

$$r_{13} = -(as_4 - c_1c_2c_4)c_6 - (bs_5 + (ac_4 + c_1c_2c_4)c_5)s_6$$

$$r_{21} = -((ds_4 + s_1c_2c_4)s_6 - (fs_5 + (dc_4 - s_1c_2s_4)c_5)c_6)c_7 - ((dc_4 - s_1c_2s_4)s_5 - fc_5)s_7$$

$$r_{22} = -((dc_4 - s_1c_2s_4)s_5 - fc_5)c_7 + ((ds_4 - s_1c_2c_4)s_6 - (fs_5 + (dc_4 + s_1c_2s_4)c_5)c_6)s_7$$

$$r_{23} = (ds_4 - s_1c_2c_4)c_6 - (fs_5 + (dc_4 + s_1c_2s_4)c_5)s_6$$

$$r_{31} = (hs_6 + (gc_5 + c_2s_3s_5)c_6)c_7 - (gs_5 - c_2s_3c_5)s_7$$

$$r_{32} = -(gs_5 - c_2s_3c_5)c_7 - (hs_6 + (gc_5 + c_2s_3s_5)c_6)s_7$$

$$r_{33} = -hc_6 + (gc_5 + c_2s_3s_5)s_6$$

and:

$$a = s_1s_3 + c_1s_2c_3$$

$$b = s_1c_3 - c_1s_2s_3$$

$$d = c_1s_3 - s_1s_2c_3$$

$$f = c_1c_3 + s_1s_2s_3$$

$$g = s_2s_4 - c_2c_3c_4$$

$$h = s_2c_4 + c_2c_3s_4$$

Using the same a - h terms as defined above, the translational terms are:

$$x_7^0 = L_1c_1 + L_2c_1c_2 - L_3a - L_4(as_4 - c_1c_2c_4) - L_5(bs_5 + (ac_4 + c_1c_2s_4)c_5)$$

$$y_7^0 = L_1s_1 + L_2s_1c_2 + L_3d + L_4(ds_4 + s_1c_2c_4) + L_5(fs_5 + (dc_4 + s_1c_2s_4)c_5)$$

$$z_7^0 = -L_2s_2 + L_3c_2c_3 - L_4h + L_5(gc_5 + c_2s_3s_5)$$

It is important to note that the origins of frames six and seven both exist at the joint of the wrist, therefore the translational terms as defined above only use the first five angle joints in their calculations.

To put these calculations to use in the simulation, a similar approach was taken in MATLAB to solve for the kinematics. Fortunately, MATLAB can handle complex calculations without the need of completely and numerically solving for each variable of the homogeneous transformation matrix as was done above. This was done by creation a script in MATLAB that

contained two functions: `transform` and `DHMatrix`. The first function, `transform`, takes each of the seven joint angles, as well as the desired joint number as inputs, and solves for the homogenous transformation up to that joint. Its output contains the x,y,z coordinates obtained from the upper right 3x1 position matrix of the transformation matrix. The function `DHMatrix` takes the DH parameters of the current joint as an input and outputs the associated homogeneous transformation matrix.

To work as a complete unit, the function `transform` is first called. The lengths used in this function will be of Baxter's arm and are defined as follows:

| Length | Value (mm) |
|----------------|------------|
| L ₀ | 270.35 |
| L ₁ | 69.00 |
| L ₂ | 364.35 |
| L ₃ | 69.00 |
| L ₄ | 374.29 |
| L ₅ | 10.00 |
| L ₆ | 368.30 |

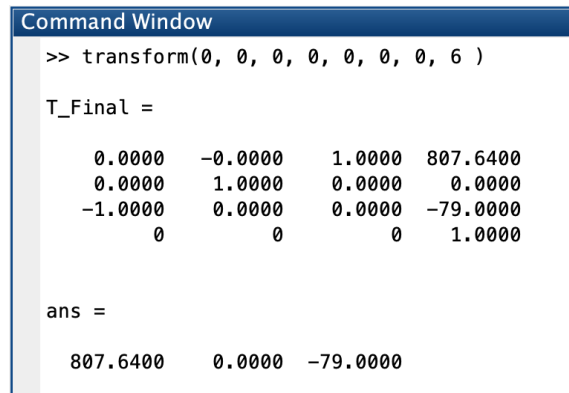
Table 1.3 Lengths of Baxter's Linkages

Then, each row of the table of DH parameters is put into the proper homogenous transformation matrix variable through the use of `DHMatrix`. This function uses the general solution of forward kinematics, ${}^{i-1}_iT$, as expressed above to calculate each matrix accurately and efficiently. Now that each of the transformation matrices for each link are separately defined, a `switch` case is used to determine the extent of the calculations done. For example, if the function call is `transform(0, 0, 0, 0, 0, 0, 0, 6)`, the function will calculate the forward kinematics up until the final joint (given by the input 6), with each joint angle being zero. If the last input in the function call was a different number, for example a three, the calculations would

be done up until the third joint. After the homogeneous matrices are multiplied together, the program extracts the data for the 3x1 position matrix from the final homogeneous matrix and outputs those coordinates.

In order to test the functions and verify their results, the output of the MATLAB script is compared with the actual kinematics of Baxter.^[1] Both calculations are computed assuming that each joint on Baxter’s arm is at zero:

$${}^0_7T = \begin{bmatrix} 0 & 0 & 1 & 0.808 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & -0.079 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



```

Command Window
>> transform(0, 0, 0, 0, 0, 0, 0, 6)

T_Final =

    0.0000    -0.0000    1.0000   807.6400
    0.0000    1.0000    0.0000    0.0000
   -1.0000    0.0000    0.0000   -79.0000
         0         0         0         1.0000

ans =

   807.6400    0.0000   -79.0000
  
```

Figure 28 Output of transform.m.

Note that, while the given solution of Baxter’s kinematics are expressed in meters, the output of transform.m is given in millimeters.

4.3.2.1.3 Inverse Kinematics

In general, the process for solving the inverse kinematics of a serial-chain robot is as follows: given the position and orientation (or pose) of the end-effector of the robot, calculate the joint angles needed to achieve that pose. For robots such as Baxter, this solution begins with the forward kinematics equations solved for above and will be computed using the Newton-Raphson numerical approach. This approach requires sufficient knowledge of Baxter’s movements as it requires a good initial guess and will only result in one solution. This method requires six scalar

functions to be defined, three of which are the three translational functions defined above, and the other three are three of the nine rotational matrix terms defined above.

To begin, the first three functions are defined as:

$$f_1(\Theta) = L_1c_1 + L_2c_1c_2 - L_3a - L_4(as_4 - c_1c_2c_4) - L_5(bs_5 + (ac_4 + c_1c_2s_4)c_5) - x_7^0$$

$$f_2(\Theta) = L_1s_1 + L_2s_1c_2 + L_3d + L_4(ds_4 + s_1c_2c_4) + L_5(fs_5 + (dc_4 + s_1c_2s_4)c_5) - y_7^0$$

$$f_3(\Theta) = -L_2s_2 + L_3c_2c_3 - L_4h + L_5(gc_5 + c_2s_3s_5) - z_7^0$$

The second three functions will be chosen from the rotation matrix terms. In order to correctly solve for the inverse kinematics of Baxter using the Newton-Raphson solution method, these three terms must all be independent from each other.

$$f_4(\Theta) = -(as_4 - c_1c_2c_4)c_6 - (bs_5 + (ac_4 + c_1c_2c_4)c_5)s_6 - r_{13}$$

$$f_5(\Theta) = (ds_4 - s_1c_2c_4)c_6 - (fs_5 + (dc_4 + s_1c_2s_4)c_5)s_6 - r_{23}$$

$$f_6(\Theta) = -(gs_5 - c_2s_3c_5)c_7 - (hs_6 + (gc_5 + c_2s_3s_6)c_6)s_7 - r_{32}$$

Now that the functions are all defined, a set of vectors can also be defined in accordance with the Newton-Raphson method:

$$\{F(\Theta)\} = \begin{Bmatrix} f_1(\Theta) \\ f_2(\Theta) \\ f_3(\Theta) \\ f_4(\Theta) \\ f_5(\Theta) \\ f_6(\Theta) \end{Bmatrix}$$

$$\{F\} = \begin{Bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \end{Bmatrix}$$

With the vectors defined, a Taylor Series Expansion of $\{F\}$ can be performed about $\{\Theta\}$:

$$f_i(\{\Theta\} + \{\delta\Theta\}) = \{F(\Theta)\} + \sum_{j=1}^n \frac{\partial f_i}{\partial \theta_j} \delta\theta_j + O(\{\delta\Theta^2\}), \quad i = 1, 2, \dots, m$$

In accordance with the Newton-Raphson method, $[J_{NR}] = [J_{NR}(\Theta)] = \left[\frac{\delta f_i}{\delta \theta_j} \right]$ is introduced as the Newton-Raphson Jacobian Matrix. It is important to note that this matrix is not square, but 6×7 . This greatly reduces the amount of computation required no matter how many joints a kinematically-redundant robot has beyond seven because only a 6×6 inversion is ever required in order to form a solution. Since many of the higher-order terms $O(\{\delta\Theta^2\})$ from the above Taylor Series Expansion are negligible when $\{\delta\Theta\}$ is small, we can assume that they go to zero. The resulting equation is:

$$\begin{aligned} f_i(\{\Theta\} + \{\delta\Theta\}) &= f_i\{\Theta\} + \sum_{j=1}^n \frac{\partial f_i}{\partial \theta_j} \delta\theta_j, \quad i = 1, 2, \dots, m \\ &= f_i\{\Theta\} + [J_{NR}]\{\delta\Theta\} = \{0\} \end{aligned}$$

This method calculates the solution in iterative steps. This is done because the basis of this method is that an good initial guess is given and one possible solution is found out of multiple. Because of this, a correction factor $\delta\Theta$ is calculated at each iterative step and $\{F(\{\Theta\})\} + [J_{NR}]\{\delta\Theta\} = \{0\}$ must be solved. One such possible solution is:

$$\{\delta\Theta\} = -[J_{NR}]^* \{F(\{\Theta\})\}$$

where:

$$[J_{NR}]^* = [J_{NR}]^T ([J_{NR}][J_{NR}]^T)^{-1}$$

In this solution, the iterations are not made to optimize the path in which the arms travel while attempting to find the desired end-effector location. Instead, a Moore-Penrose Pseudoinverse finds so that it is the minimum possible at each step. This is not the fastest way of solving this problem and does not account for the Jacobian velocity values of the robot, but since the simulation is developed purely to visualize the movement of Baxter as well as test the derived math, this solution is acceptable and provides sufficient results.

4.3.2.1.4 Gravity Compensation

Since both the joints and links have weight, a simple position control loop was implemented to compensate for gravity. At a high level, the angles of each joint are monitored so that they do not overshoot or undershoot the desired angle values. This allows for the monitoring of the angles during movement, as well as when the arms have reached their final positions and orientations and must be held in place. Although a more complicated gravity compensation policy can be implemented, the scope of this project did not necessarily require one. This is because the main focus of this simulation was to analyze the forward and inverse kinematics of both Baxter and the RoboPuppet while simultaneously developing a system to visualize their movements. The simple position control loop that is implemented in the software allows the robot to be accurate with its movement (less than half a millimeter) as well as hold its place, while maintaining the primary focus of the project. Since the Jacobian velocity matrix was not needed in the derivation of the inverse kinematics of the robot, the velocity is set at a constant so that its effects are negligible, specifically with regards to the current gravity compensation policy.

4.3.2.2 Visualization and Control Interface

With the main focus of this project being a complete three-dimensional simulation, the actual visualization of Baxter must be an accurate representation of what Baxter actually looks like and how Baxter actually moves. If the visualization is inaccurate, it is impossible to correctly understand how Baxter is moving through time and space.

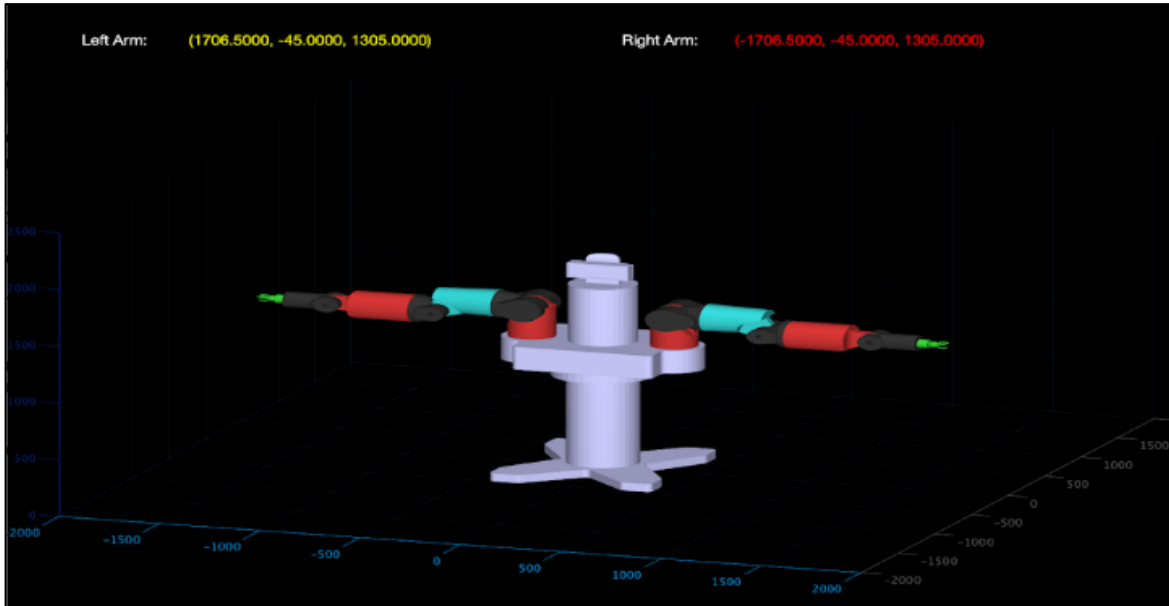


Figure 29 Three-Dimensional Visualization of Baxter.

To accomplish this, a full scale model of Baxter was created and loaded into MATLAB. This was done by using MATLAB's various built-in functions to properly read a .stl file that was given to the main function call in the software. These files are CAD renderings of each link of Baxter's arms, as well as a rough model of Baxter's body.

In order to properly control the simulation, a graphics user interface must be created for easy access to all of the necessary variables. Upon running the simulation, the user is greeted with the following screen, prompting them to input their desired joint angles for the calculation and subsequent animation of the forward kinematics of the robot:

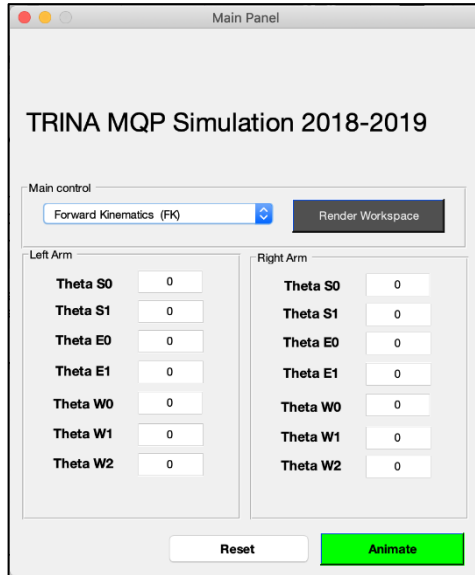


Figure 30 Forward Kinematics Control Interface.

By selecting “Inverse Kinematics (IK)” from the drop-down menu, the control interface for the inverse kinematics of the robot appear and the user is prompted to input the desired end-effector position for the calculation and subsequent animation of the robot (see Figure 31).

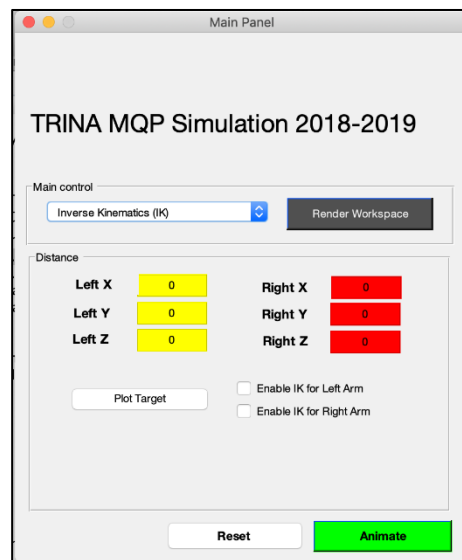


Figure 31 Inverse Kinematics Control Interface.

In order for the user to actually animate the simulated robot using inverse kinematics, the user must enable the inverse kinematics for either the right or left arm, or enable both. This is

done to increase the speed and efficiency of the simulation as there are many iterative steps that the calculations must go through in order to find the correct solution for the inverse kinematics of the robot using the Newton-Raphson method.

5. Testing

5.1 Mechanical Testing

The testing for this project was done by the creation of an assembly in Solidworks to see if parts crashed into one another, and by the realized fabrication of the parts. When an issue was not caught in the CAD design process it was solved by altering the plastic components of the arm itself.

There were components where the hall effect encoders did not fit with the given mounts and were easily fixed by redesigning either the parts or the mounts to not be in the way of the part. The parts were physically altered by applying heat to a desired area, making the plastic easy to mold, warp or cut. When pieces needed to be slightly angled due to the parts shifting from the mass of the arm itself, it was easy to address this issue by heading the mount and bending it to the angle desired.

The assembly process is not one to be taken lightly. It takes hours of sanding, altering, and stripping the pieces of any unwanted support material in order to get a clean final design. Once all of the parts were printed and shown to fit well, many of the mounts would print with small deformities that would not allow the encoders to be properly attached. This was rectified by heating the plastic until it was soft and pinching the part to make the mount holes the proper desired tightness.

Testing the parts was very trial and error based. This meant that even though the design worked in a Solidworks assembly, it did not mean that it would necessarily work in a real work situation. When parts would have issues or not like up it was important to methodically find the issue, starting with the possibility that the error was done by the manufacturer. These issues often are due to missed details in the stripping of the part of supports. Following manufacturer error that net most likely issue is in the shifting of parts due to gravity. It is common for things to start to move, which is easily fixed by reducing the tolerance of the part or physically adjusting part of the RoboPuppet to the issue at hand.

5.2 Electrical Testing

This section concerns the electrical testing portion of the project. It involves testing of the components to evaluate their functionality as well as determining optimal methods to assemble the system that will control the active RoboPuppet.

5.2.1 Hall Effect Encoder Testing

To test the functionality of the hall effect encoders, they had to be hooked up individually to see if their I2C address was read by the microcontroller. At first, it was difficult to figure out how to exactly hook the sensors up so that they would read properly. According to the datasheet, there were two options to hook up the encoders, either in 3.3V operation or 5V operation. This diagram also shown earlier in this paper is shown below.

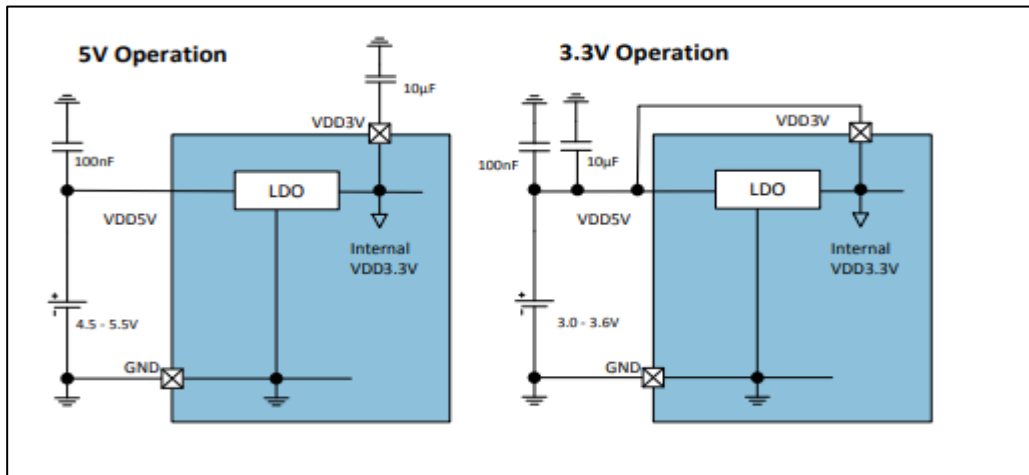


Figure 32 5V and 3.3V operations for AS5048B.

This would require a 100nF and a 10uF capacitor to counteract any inductance from the wire transmitting the I2C signal. A breadboard test was conducted, connecting a single hall effect encoder to a Teensy. The goal is to be able to run a built in Scanner Arduino function and read the address of the chip. A successful reading of the Arduino serial monitor can be shown in Figure 33.

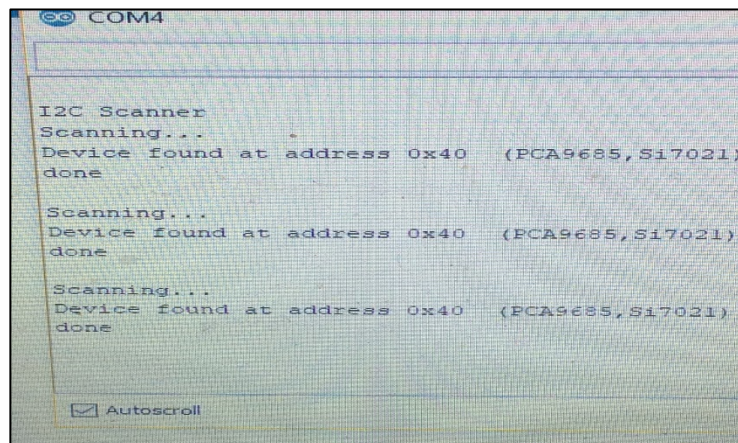


Figure 33 Confirmation of I2C communication in Arduino Serial Monitor.

Once this configuration was proven to work, another encoder set-up can be put on the breadboard. Figure 34 shows two sets of encoder circuits connected to a single Teensy I2C line. In this particular picture, encoders are not connected yet.

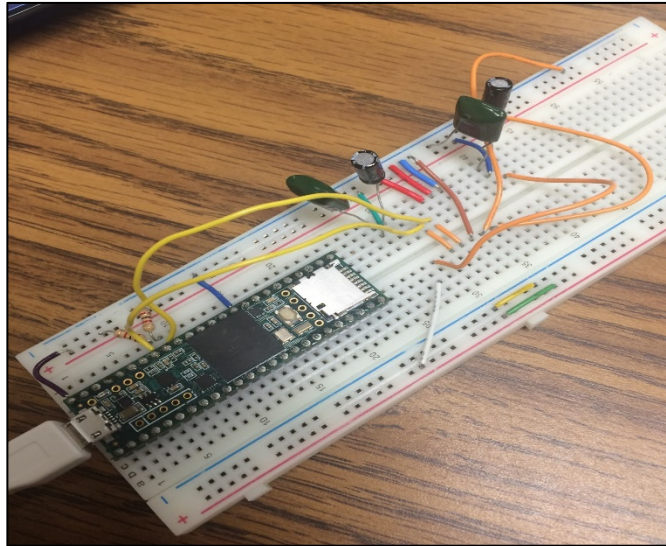


Figure 34 Breadboard layout of hall effect encoder driving circuit.

Another important aspect to consider is the use of pullup resistors on the SDA and SCL pins. The purpose of a pullup resistor is to “pull-up” a floating voltage to truly distinguish if a device is triggering the pin. It increases the robustness of the I2C communication and is found in almost every I2C application. For the Teensy 3.5 it is recommended that 4.7k ohm resistors are used for I2C. (PJRC) Eventually a series of breadboards were assembled to be able to chain 14 encoder circuits on one SDA and SCL line. It was a messy wire configuration, but it was okay for a prototype knowing the PCB would be a lot neater and more professional looking.

In this particular project design, it was found that one of the AS5048B hall effect encoders seemed to be defective and ordering a replacement proved to be difficult. To temporarily solve this, a similar sensor was purchased, the AS5601. The overall breadboard and eventual PCB design did not have to be altered drastically for one sensor. The only main difference was the ordering of the four connection pins, Vcc, GND, SDA, and SCL. After all 14 sensors could be detected at once, the next step was to develop Arduino code that could iterate through the addresses and produce the angle reading based on the magnet in the arm’s position. The full code developed for this step can be found in the Appendix. User created Arduino

libraries for particular sensors made this step particularly easier than developing code from scratch. The code creates two arrays of sensors that has associated addresses. What the code does is first initialize the encoders, calibrating them to zero. After that it reads the joint angle value and prints the value to the serial monitor.

5.2.2 Motor Testing

To test the servos, a couple methods were used, one being more effective than the other. The initial idea was to simply use PWM pins on the Teensy to drive the servos and use external power knowing that having the Teensy drive motors and read encoder values at the same time would be a bad idea. Using PWM digital output to control motors was a good idea in theory, however there were problems that came with this method. According to the Teensyduino website, the Teensy can only output up to 12 PWM signals at once, making an application like this difficult with 14 servos needed to be used. The solution was to implement a servo driver that has 14 or more ports. As discussed earlier, the PCA9685 I2C PWM servo driver solved this issue. 14 of the 16 ports were used for the active RoboPuppet. Testing of this breakout board proved to be similar to testing the hall effect encoders due to the I2C communication. A breadboard layout can be found in Figure 35.

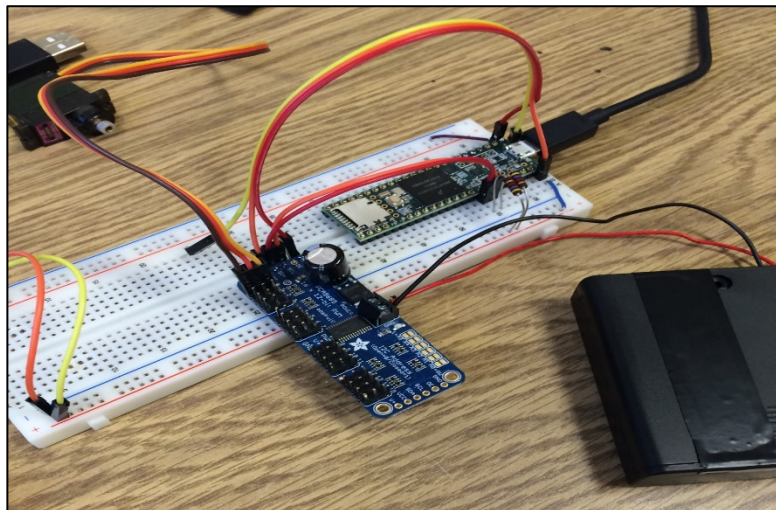


Figure 35 Breadboard testing for I2C Servo Driver.

The only connection to the Teensy was SDA and SCL, Voltage and Ground. A single motor was tested to make sure the chip worked. A built in example function from the Adafruit website tested the servo's sweeping abilities. The board was proven successful when the motors

activated and rotated when the program ran. After the I2C board was proven functional, it could be implemented into the PCB.

5.2.3 PCB Testing/Troubleshooting

To hook up the PCB, most of the components had to be through hole soldered. Components that weren't through hole soldered were surface mounted like the 3.3V regulator and the barrel jack. The encoder wires had to be pressed down with crimp connectors to pierce through the metal spikes, connecting the wires to the board via the rectangular connectors. The wires had to be connected one at a time because the I2C interface seemed to be very picky with connections. The wire configuration had to be very precise because any short would kill the I2C signal, resulting in the serial monitor of the scanning function to read "No I2C devices found." Making sure every set of hall effect encoder wires connected to the board took some time, but it was eventually achieved. The crimp connector connections can be seen in Figure 36.

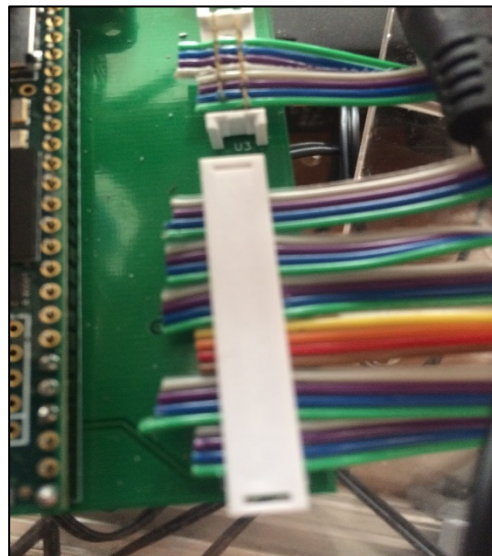


Figure 36 Wire Connection from Sensors to PCB.

5.3 Software Testing

Thankfully, testing the simulation was relatively straight forward and consisted of standard debugging practices. Firstly, to test the derived equations for miscalculations, separate MATLAB scripts were created that solved for the forward or inverse kinematics separately. Later on, these scripts were implemented into the final program that ran the simulation. This

proved to be an effective way of solving for and testing the equations that were originally derived by hand because MATLAB handled most of the calculations. This allowed the equations derived by hand to be directly compared to the solutions solved for in MATLAB.

From here, standard debugging practices were used so that the program could be first checked for syntax errors within the code. Once the syntax errors had been fixed, the logic of the code could then be checked and tested. “Milestones” were placed at every major aspect of the code to ensure that it was working all of the way throughout the program. These “milestones” came in the form of basic print statements, as well as outputs of more intense calculations. Once all of the calculations and logic of the code was reviewed, the movement of the simulated robot Baxter could finally be checked. With all of these different parts working in conjunction with each other, the simulation ran flawlessly and efficiently.

6. Recommendations

6.1 Mechanical Recommendations

For next year's team, it would be well advised to have one or multiple people that are very well adept at using Solidworks and know how to expertly manipulate PLA plastic. The mechanical design focus for next year should be on the integration of the wiring to a smooth internal channel within the RoboPuppet. This will make the entire design easier for the user and less likely to be pulled apart by an awkward angle. This can be accomplished by either a total redesign of the internal components of the puppet, or a redesign of the outside of the components to account for a sleeve that can be placed over the parts.

The other major aspect that can be improved mechanically is the shape of the hand control to be more intuitive and comfortable for users. The control is currently in the shape of an extruded rounded square, or a "squircle". This design is not uncomfortable, however can be vastly improved upon by making it more ergonomic towards an actual human hand.

Another aspect that should be looked into is making the hand control more slip resistant. This can be done using a rubberized dip or spray such as, "plasti-dip". However, if you are to do this you will need to take special care to not have any of the components, or component housings damaged by this process. Look into the way that sculptors and ceramic artists ensure that their pieces have even coatings of glaze while also being selective on where the glazes touch.

The arms are held together using a D-axle, which has a single flat surface as its key point. This means that the part holding it together will be strong, however does not always have a good grip. With this in mind the RoboPuppet is in need of constant tightening of the collars that hold the axle to the motors. This can be possibly solved by using a material like locktite on the collar, or by replacing the axles with one that is less likely to slip or come loose from the collar's key.

In conclusion there are several different ways that this design can be improved upon that can easily take an entire MQP group. When fabricating the plastic components do not use a low quality printer. Having a high tolerance in your parts is what will ultimately be the deciding factor to your design working or not. Parts like the hand housing utilized an exceptionally tight margin for error and the Prusa 3D printers in the Higgins MQP lab were more than sufficient at accomplishing everything required of it. It is easier to purchase your own filament and print using the Higgins Prusa printers than to have parts printed by Foisie.

6.2 Electrical Recommendations

This year, major strides were made to optimize the electrical workings of the active RoboPuppet. A functional PCB was developed in addition to being able to connect everything to one system. As well as this version of the RoboPuppet was, there is room for improvement for the next group of students who may work on this project. The most impactful recommendation for the future would probably be replacing sensors so that they are all the same model. The only reason there is a mix of old AS5048B sensors with a new AS5061 sensor is that during the development process of this project, AS5048B sensors were not available anywhere on the internet to buy, therefore leading to believe they were not being sold anymore. At the time of the writing of this paper, however, they are, in fact, in stock. In conclusion, a recommendation would be to keep the sensors consistent throughout the arms. It would be preferable to stay with the old AS5048B sensors for a few reasons, one being that it would be cheaper to fully replace (only one sensor rather than many), the male header pin installation would be intuitive and easy to use, and AS5061 sensors have a fixed address of 0x36, so an I2C multiplexer, or mux would have to be implemented in the future iterations of the PCB. An example would be the TCA9548 1-8 I2C Multiplexer breakout board from Adafruit, as shown in Figure 37.

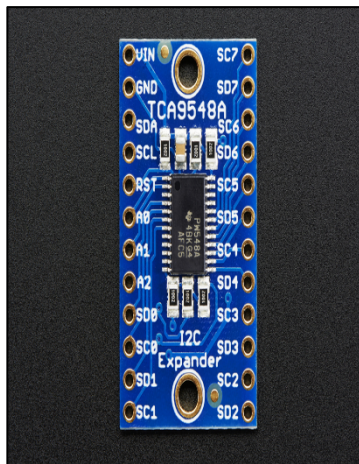


Figure 37 TCA9548A I2C Multiplexer.

This way a microcontroller like the Teensy can sort through the ports with the same address. This board would have to be included in any future circuit design regarding fixed address sensors like the AS5061. This is the more expensive route to take, but it is an option. If 14 sensors are used, then 2 muxes are necessary to account for the 16 I2C lines.

6.3 Software Recommendations

The simulation that was developed for the 2018-2019 Telenursing RoboPuppet MQP focused mainly on the derived equations and calculations needed for a working model of Baxter in MATLAB. This simulation did not focus on the interactivity between the RoboPuppet and the simulation and Baxter. This was done to ensure that the focus of the project was on the kinematics of the robot and the compensation of gravity in Baxter's arms. This was an important step in the process because the current active RoboPuppet would be rendered useless without them in terms of integrating it with Baxter. This means that without the correct calculations being done and tested for through the use of a simulation, next year's team would have to focus more on the calculations, rather than the integration between the RoboPuppet and Baxter. With this component being complete, more work can be done on the communication between RoboPuppet and Baxter through the use of programming languages and environments such as Python and ROS. In conclusion, the work done this year is only a step in the process of the bigger picture. Next year's team will be able to more easily and more readily expand upon the current active RoboPuppet because of the derivation of the kinematics equations and the focus on creating a working gravity compensation policy that was done this year.

7. Appendices

Arduino code for testing hall effect encoders

```
#include <AS5601.h>

#include <ams_as5048b.h>
#define LEFT_ARM 7
#define RIGHT_ARM 6
#define U_DEG 3

AMS_AS5048B left_sensors[LEFT_ARM] = {AMS_AS5048B(0x47), AMS_AS5048B(0x46),
AMS_AS5048B(0x40), AMS_AS5048B(0x49), AMS_AS5048B(0x48), AMS_AS5048B(0x4C),
AMS_AS5048B(0x44)};
AMS_AS5048B right_sensors[RIGHT_ARM] = {AMS_AS5048B(0x42), AMS_AS5048B(0x45),
AMS_AS5048B(0x4B), AMS_AS5048B(0x44), AMS_AS5048B(0x4A), AMS_AS5048B(0x43)};

AS5601 new_sensor1;

void setup() {
  Serial.begin(4800);
  Wire.begin();
  Wire.setSCL(19);
  Wire.setSDA(18);
  while (!Serial);
  for (int i = 0; i < LEFT_ARM; i++) {
    left_sensors[i].begin();
    left_sensors[i].setZeroReg();
  }

  for (int j = 0; j < RIGHT_ARM; j++) {
    right_sensors[j].begin();
    right_sensors[j].setZeroReg();
  }
}

void loop() {
  for (int i = 0; i < LEFT_ARM; i++)
  {
    Serial.print("Sensor ");
    Serial.print(i);
    Serial.print(" | angle: ");
    Serial.println(left_sensors[i].angleR(U_DEG, true));
    delay(500);
  }

  for (int j = 0; j < RIGHT_ARM; j++)
  {
```

```
Serial.print("Sensor ");
Serial.print(j +7 );
Serial.print(" | angle: ");
Serial.println(right_sensors[j].angleR(U_DEG, true));
delay(500);
}
Serial.print("Sensor 13");
Serial.print(" | angle: ");
Serial.println(new_sensor1.getAngle());
delay(500);
}
```

7.1 References

1. [1] Williams II, R. (2019). *Baxter Humanoid Robot Kinematics*. [online] Ohio.edu. Available at: <https://www.ohio.edu/mechanical-faculty/williams/html/pdf/BaxterKinematics.pdf> [Accessed 22 Apr. 2019].
2. “Adafruit PCA9685 16-Channel Servo Driver,” *Overview | Adafruit PCA9685 16-Channel Servo Driver | Adafruit Learning System*. [Online]. Available: <https://learn.adafruit.com/16-channel-pwm-servo-driver>. [Accessed: 24-Apr-2019].
3. “Adafruit TCA9548A 1-to-8 I2C Multiplexer Breakout,” *Overview | Adafruit TCA9548A 1-to-8 I2C Multiplexer Breakout | Adafruit Learning System*. [Online]. Available: <https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/overview>. [Accessed: 24-Apr-2019].
4. “AS5048 - 14-bit rotary position sensor with digital angle (interface) and PWM output,” *ams*. [Online]. Available: <https://ams.com/as0548b>. [Accessed: 24-Apr-2019].
5. Bitfasching, “bitfasching/AS5601,” *GitHub*, 30-Sep-2017. [Online]. Available: <https://github.com/bitfasching/AS5601>. [Accessed: 24-Apr-2019].
6. M. M. Lamare, B. M. Heavey, and E. M. Stelly, “Telenursing RoboPuppet,” rep.
7. Last year's MQP Paper. This project built off of the project from last year.
8. Sosandroid, “sosandroid/AMS_AS5048B,” *GitHub*, 26-Feb-2018. [Online]. Available: https://github.com/sosandroid/AMS_AS5048B. [Accessed: 24-Apr-2019].
9. P. J. Stoffregen and R. Coon, “Teensyduino,” *PJRC*. [Online]. Available: <https://www.pjrc.com/teensy/teensyduino.html>. [Accessed: 24-Apr-2019].
10. “Tutorials,” *KiCad EDA*. [Online]. Available: <http://kicad-pcb.org/help/tutorials/>. [Accessed: 24-Apr-2019].