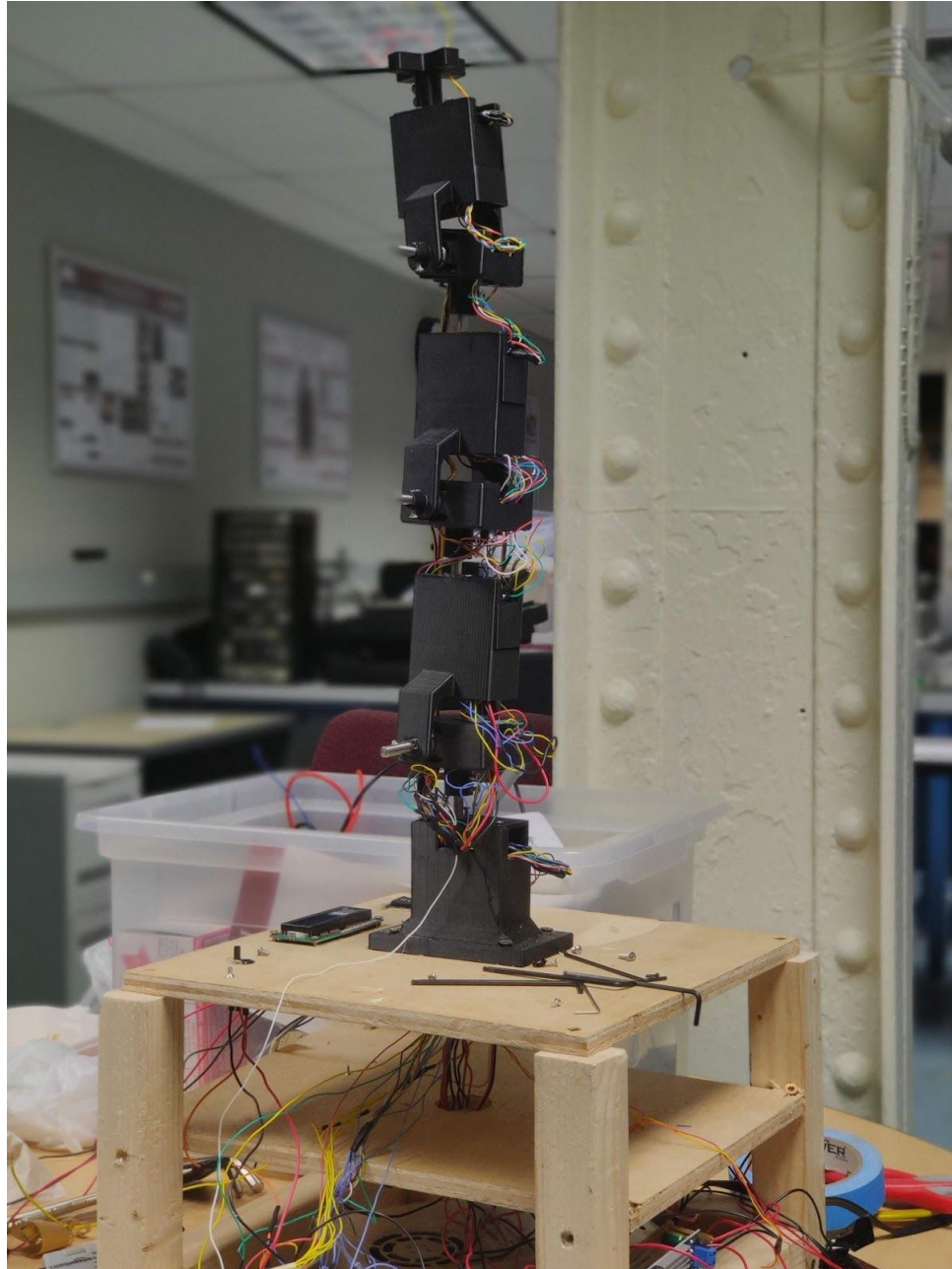


RoboPuppet for Teleoperation

Mario Castro, Francesco De Leo, Isabella Feeney, Jason Jemison, Yichen Yang



A Major Qualifying Project submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of of the requirements for the degree of Bachelor of Science

Authors:

Jason Jemison

Yicheng Yang

Date:

December 11, 2020

Report submitted to:

Professor Zhi Li

Worcester Polytechnic Institute

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see <http://www.wpi.edu/academics/ugradstudies/project-learning.html>.

Abstract

The RoboPuppet is a model version of the Kinova Robotics' Kinova Gen 3 arm used for intuitive remote control. The RoboPuppet Arm contains joint angle sensors and motors which allow for gravity compensation. The project has a ROS package for controlling the Kinova Gen 3 arm in the Kinova Simulation or Trina2 Simulation. It also includes a basic GUI for controlling the simulation. This platform is ideal for helping nurses work remotely with patients in high-risk environments.

Table of Contents

Abstract	2
Table of Figures	5
Authorship	7
Mechanical Design	8
Design Goals	8
Design	8
Overview	8
Rotary Link	8
Twist Link	9
Base	10
End Piece	11
Connections	12
Motor Selection	12
Design Iterations	15
Iteration 1: “Scaled” Arm	15
Iteration 2: “Un-Scaled” Arm 1.0	16
Iteration 3: “Un-scaled arm” 2.0	17
Design Issues	18
Mechanical Analysis	19
Torque Calculation Script (torques.py)	19
Minor Design Changes	20
Electrical Design	21
Microcontroller Subsystem	22
Sensors	22
Microcontroller	23
Power Distribution Subsystem	24
PCB Design	26
Firmware (Firmware/)	28
Main Control Program (src/Main.cpp)	28
Firmware Subsystems (sub/)	28
Absolute Encoders Subsystem (absEncoders/)	30
Buttons Subsystem (Buttons/)	30
Hall Encoders Subsystem (hallEncoders/)	30
Servos Subsystem (Servos/)	30
LCD Info Subsystem (SerialInterface/)	31
Serial Interface Subsystem (SerialInterface/)	31
Config Subsystem (Config/)	31
Serial Message Protocol	31
ROS Software (Catkin/src/RoboPuppetMQP/)	32
Launch File Structure (launch/)	33

ROS Source Code (src/)	34
Constants File (constants.py)	34
Login Interface (gui_login.py)	34
Sign Up Interface (gui_signup.py)	34
GUI Lite (gui_lite.py)	35
GUI Pro(gui_pro.py)	36
Kinematics Class (kinematics.py)	37
Feedback Survey(gui_feedback.py)	37
GUI Design Folder(ui/)	38
Trina2 Environment	38
ROS Topics and Services	39
Conclusion	40
Appendices	41
Acknowledgements	41
Kinova Launch Instructions for Ubuntu 18.04	42
Important Links	43
D) Parts List	44

Table of Figures

<i>Figure 1: Rotary Link</i>	9
<i>Figure 2: Twist Link</i>	10
<i>Figure 3: Base Link</i>	11
<i>Figure 4: End Piece</i>	12
<i>Figure 5: Stepper vs Servo Comparison</i>	13
<i>Figure 6: Table of Torques and Motor Selections</i>	14
<i>Figure 7: Servo Model Choice</i>	15
<i>Figure 8: Iteration 1 Design</i>	16
<i>Figure 9: Iteration 2 Design</i>	17
<i>Figure 10: Iteration 3 Design</i>	18
<i>Figure 11: Hardware Schematic</i>	21
<i>Figure 12: AMT222A Absolute Encoders</i>	22
<i>Figure 13: AS5048A Hall Encoders</i>	23
<i>Figure 14: Arduino Mega 2560 Pinout Assignments</i>	24
<i>Figure 15: Table of Module Power Requirements</i>	24
<i>Figure 16: 12V DC Power Supply</i>	25
<i>Figure 17: TOBSUN 5V DC Power Supply</i>	25
<i>Figure 18: WHTDS DC Buck Boost Power Supply Module</i>	26
<i>Figure 19: PCB Trace Layout</i>	27
<i>Figure 20: Table of Firmware Subsystem Descriptions</i>	29
<i>Figure 21: Block Diagram for Firmware Subsystems</i>	29
<i>Figure 22: Table of Servo PWM Ranges</i>	31
<i>Figure 23: Table of Arm 1 Serial Communication Variable Types</i>	32
<i>Figure 24: Table of Arm 2 Serial Communication Variable Types</i>	32
<i>Figure 25: Block Diagram for ROS Information Nodes</i>	33
<i>Figure 26: User Login Interface</i>	34
<i>Figure 27: User Sign Up Interface</i>	35

<i>Figure 28: GUI Lite Interface</i>	35
<i>Figure 29: GUI Pro Interface</i>	36
<i>Figure 30: NASA-TLX Feedback Survey</i>	37
<i>Figure 31: Basic Trina2 Environment</i>	38
<i>Figure 32: Table of Ros Services and Topics</i>	39

Authorship

Section	Author
Abstract	Isabella Feeney
Mechanical Design - Design Goals	Francesco De Leo
Mechanical Design - Design	Francesco De Leo, Jason Jemison
Mechanical Design - Motor Selection	Jason Jemison
Mechanical Design - Design Iterations	Francesco De Leo
Mechanical Design - Design Issues	Francesco De Leo, Jason Jemison
Mechanical Design - Mechanical Analysis	Jason Jemison
Electrical Design	Isabella Feeney, Jason Jemison
Firmware - Main Control and Subsystems	Isabella Feeney, Mario Castro, Jason Jemison
Firmware - Serial Message Protocol	Mario Castro
ROS Software - Launch File Structure	Yicheng Yang
ROS Software - ROS Source Code	Yicheng Yang
ROS Software - Trina2 Environment	Isabella Feeney
ROS Software - ROS Topics and Services	Yicheng Yang
Conclusion	Mario Castro, Jason Jemison

Mechanical Design

Design Goals

This project is the fourth iteration of the RoboPuppet MQP project. The project has been updated to work with different robotic nursing arms each year. The major goals of the mechanical design portion of this year's project were focused on improving the maneuverability and ergonomics of the arm relative to previous designs. The most prominent issue of the previous year's design was the high amount of friction present in the arm during motion. In order to improve this, motors with more appropriate torques were selected for each joint in the RoboPuppet arm. In addition to this, parts were designed to hold all components inside them, as well as appear smoother for increased ergonomics and for a more aesthetically pleasing design.

Design

Overview

The arm has seven different linkages and seven points of motion to completely replicate the motion of the Kinova Gen3 arm. The arm is made up of three distinct types of linkages: the "rotary" link, the "twist" link, and a base piece. While each rotary and twist link look similar to one another, they are all unique in size to be specifically fitted to their specific motor. This allowed for the arm to not only be as small and light as possible, but to also have the lowest amount of torque between each link and therefore move as smoothly as possible.

Rotary Link

The rotary links were made up of two 3D-printed PLA pieces. The larger piece being the "casing" piece holds all the components including the motor and bearings. This piece includes several holes and cutouts to make space for tools during assembly as well as spaces for wiring to run through. The smaller piece, the "cap", encloses the parts into the casing and also allows for connection to the previous twist link.

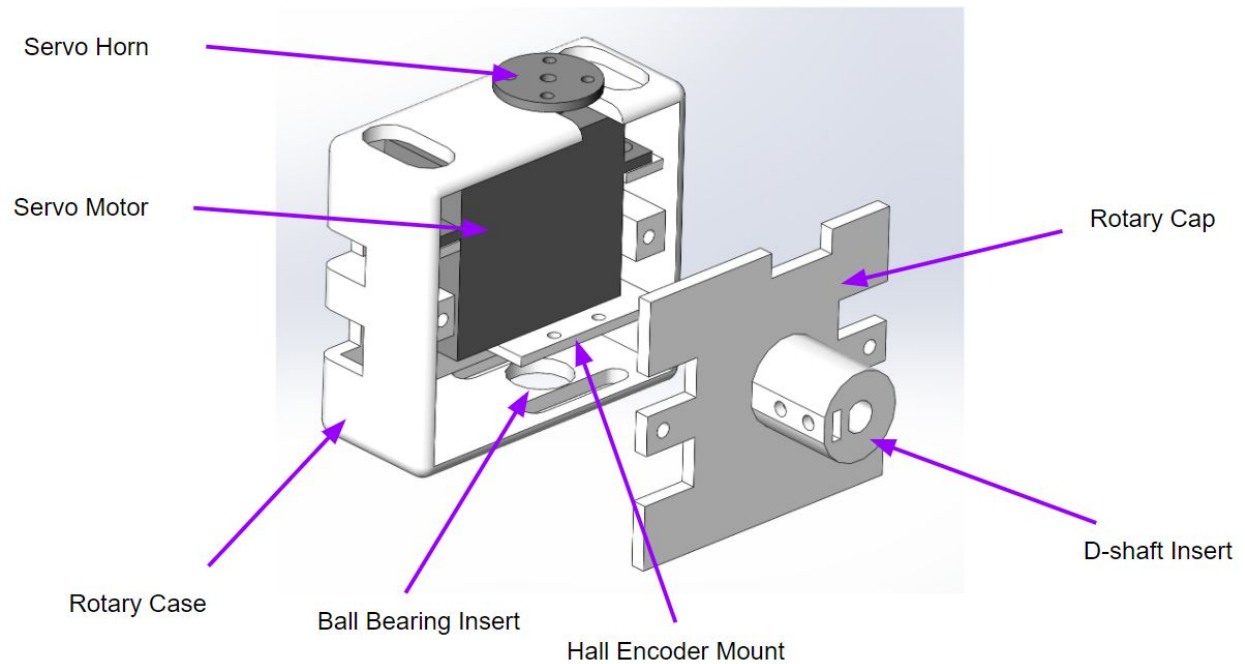


Figure 1: Rotary Link

Each rotary link includes:

- Six 3M bolts
- One Servo Motor
- One Servo Horn
- One Hall Encoder
- One Magnet
- One Magnet Mount
- One Ball Bearing
- One 0.25" Diameter aluminum D-shaft
- One Shaft Collar
- One rotary case piece
- One rotary cap piece

Twist Link

The twist links were also made of two 3D-printed PLA pieces. The larger piece being the “casing” piece holds all the components including the motor, shaft coupler, aluminum D-shaft, encoder, and bearing. This casing piece includes a U-shaped extension extruding from the bottom which fits around the previous rotary link and connects the two links. This U-shaped piece contains one 7 mm D hole for the D-Shaft to be fit into and on one side and four smaller holes for mounting the link onto the servo horn on the other side. The smaller piece being the

“cap” piece encloses everything into the casing piece and has holes for fastening the encoder into place.

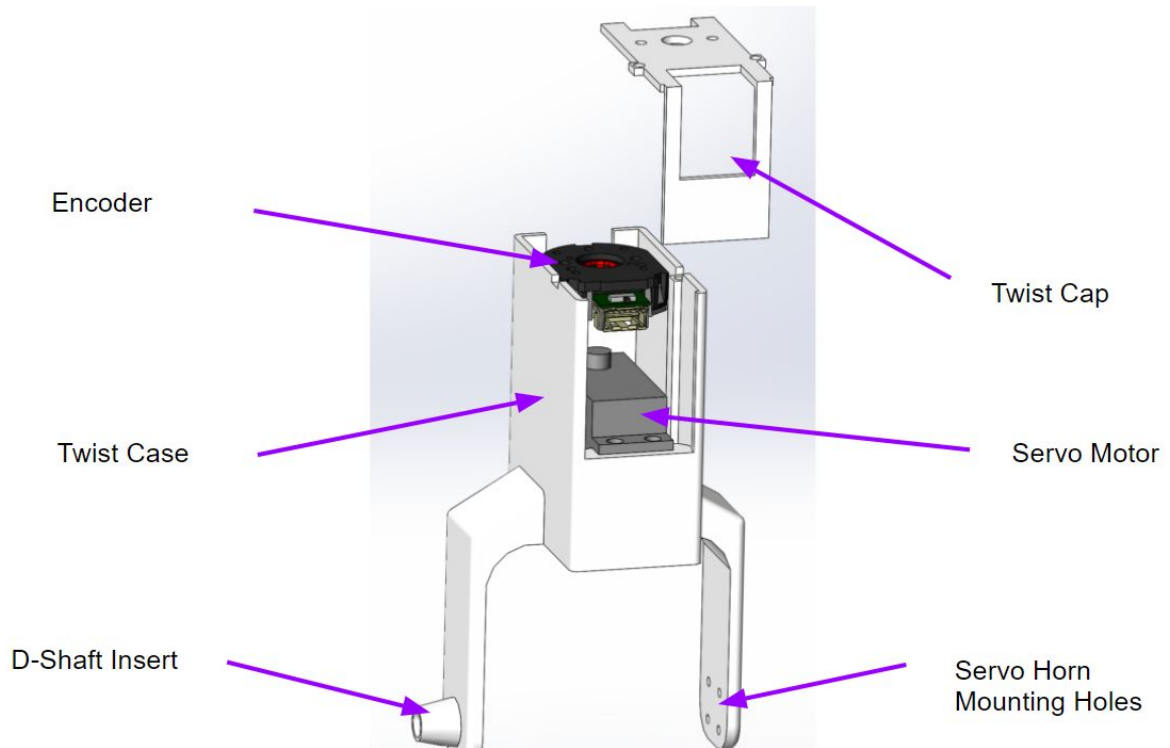


Figure 2: Twist Link

Each twist link includes:

- Eight 3M Bolts
- Four 2M Bolts
- One Servo Motor
- One AMT222A Encoder
- One 24T Servo Shaft Coupler
- One 0.25” Diameter aluminum D-shaft
- One twist case piece
- One twist cap piece

Base

The base piece, like the rotary and twist pieces is also made of two 3D-Printed PLA pieces. The larger piece is the housing which holds all components including a motor, encoder, shaft coupler, and aluminum D-shaft. This piece also has a platform at the bottom which allows for mounting the entire arm. The second and smaller piece is the “cap” piece which encloses everything into the case piece and has holes for fitting and mounting the encoder.

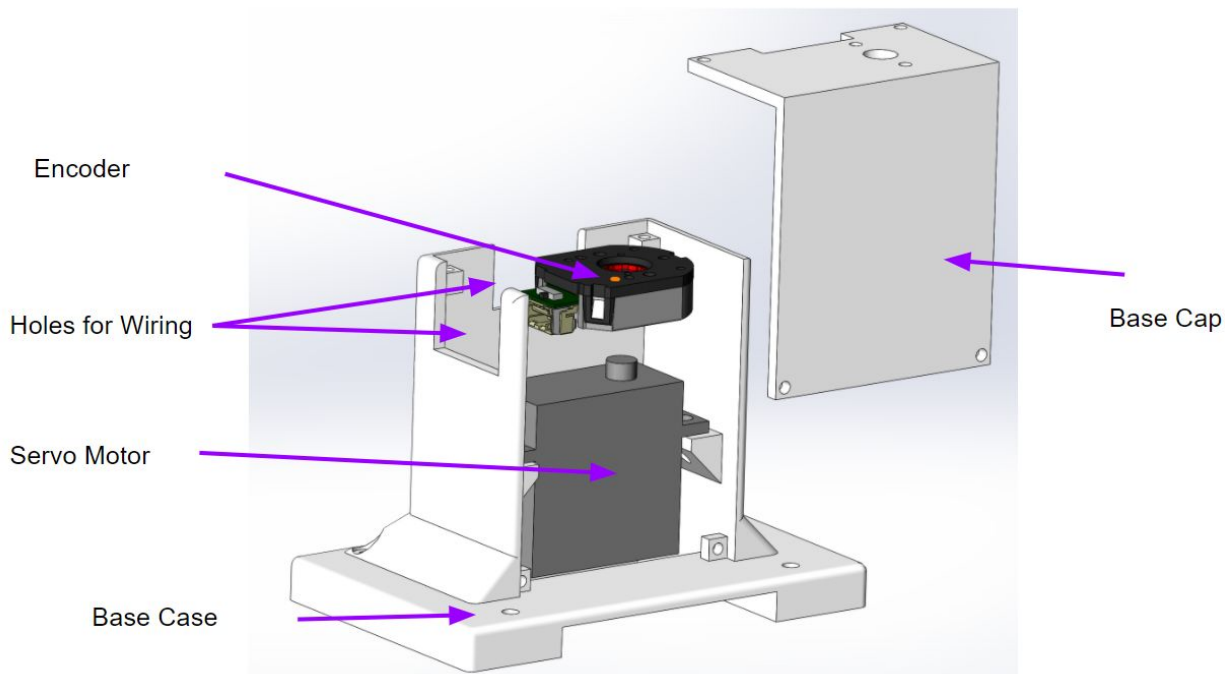


Figure 3: Base Link

The base link includes:

- Ten 3M Bolts
- One Servo Motor
- One AMT222A Encoder
- One 24T Servo Shaft Coupler
- One 0.25" Diameter aluminum D-shaft
- One base case piece
- One base cap piece

End Piece

The “end piece” is a simple x-shaped, 3D printed part that is located at the end of the arm and mounted to the final twist link with an aluminum D-shaft. This part has two main purposes; mounting a button that opens and closes the gripper, as well as allowing the user to rotate the final encoder located in the final twist link. This is crucial because the Kinova Gen3 arm has a camera and/or gripper located in this position so the user needs a way to rotate this link in order to use the camera or gripper.

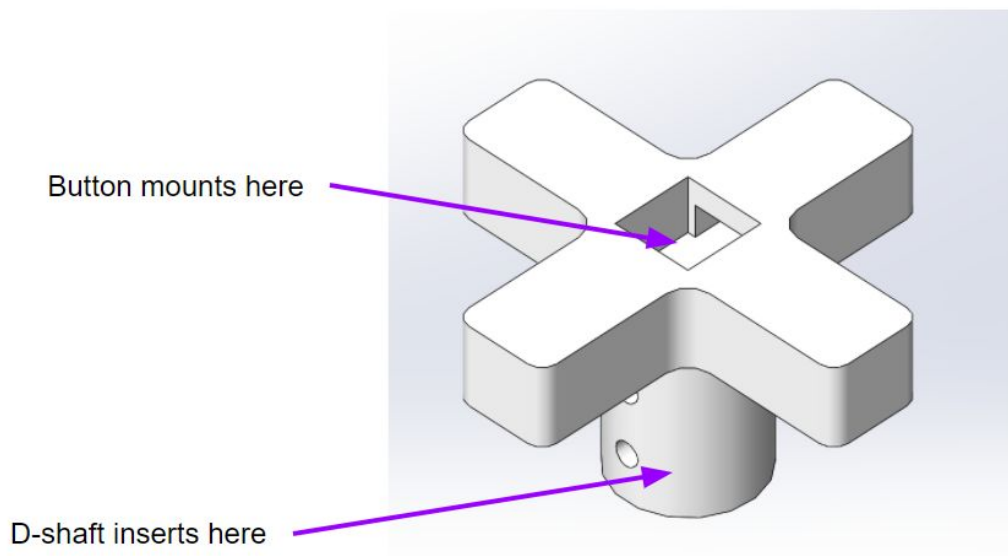


Figure 4: End Piece

Connections

There are two distinct connection types in the arm. There is a “twist to rotary” connection and a “rotary to twist” connection. The twist to rotary connection is very simple as it only requires a D-shaft running from the twist link and a fitting for it on the cap of the rotary link. The rotary to twist connection is more complex as it utilizes the U-shaped extension which creates two points of connection. One of the two connection points is between the servo horn and the arm of the U-shaped extension. This is where the servo in the rotary link directly moves the twist link as the twist link is fastened to the servo horn with four small screws. The other connection point is used to stabilize the motion and strengthen the connection. This connection is made with one ball bearing which is secured in the rotary link, a fitted insert for the shaft on the U-shaped extension, and a shaft running through them.

Motor Selection

Three different types of motors were initially considered for motor selection; servos, steppers, and dc motors. After conducting research, it decided to move forward with servo motors as the motor selection. One clear advantage of servo motors, was the weight advantage they had over the other to selections. Since they are lighter than DC and stepper motors, it meant motors with smaller torque factors could be used throughout the design of the arm. If heavier motors were chosen, all of the motors would have to have higher torque ratings, making it more difficult to manipulate the puppet arm by hand. Another advantage of choosing servo motors is no additional hardware is needed. They do not require a motor controller or motor driver to control them through a microcontroller. Instead, a PWM signal directly from a microcontroller can be

used. This also eliminates additional hardware needed in the arm, minimizing the design for weight and form factor.

Application Attributes for Stepper vs. Servo					
Application Requirements	Stepper	Servo	Application Requirements	Stepper	Servo
Highest Torque Density	✓		Detent Torque	✓	
Largest Torque and Speed Range		✓	Inertia Loads up to 30:1 (J_load : Jm)	✓	Possible
Open-Loop (typical)	✓		Inertia Loads up to 200:1 (J_load : Jm)	Possible	DD+(R / L)*
Low Speed (up to 1000 rpm)	✓	✓	Fast Corrections Against Disturbances / Commands		✓
Medium Speed (1000 - 3000 rpm)	Possible	✓	Peak Torques Available > Continuous Capability		✓
High Speed (> 3000 rpm)		✓	Highest Resolution		✓
High Torque at Low Speed (< 1000 rpm)	✓	✓	Highest Input Voltage Range		✓
High Bandwidth (BW) Response Times		✓	Simplest Integration	✓	
Point-to-Point (simple / modest)	✓	✓	Ideal for Fixed Loads	✓	
Coordination between Axes	Pseudo	✓	Highest Product Through-Put		✓
Highest Acceleration / Deceleration		✓	Highest Efficiency		✓
Hold Positon without 'Hunting'	✓				

* R= Rotary, L=Linear

Figure 5: Stepper vs Servo Comparison

When choosing servos, two types of servos were considered for the two different joint types, rotary and twist as described in the previous section. These selections were based on the mechanical constraints of each joint of the physical Kinova Gen3 arm. On the joints with a finite rotation, a rotary joint was used with a standard servo, with a finite range of motion. In a twist joint, a continuous servo was chosen to mimic the infinite range of motion on the Kinova Gen3 arm. On the standard servo, it can provide position feedback without additional hardware and for the continuous servos, an absolute encoder will be used to provide position feedback.

In order to ensure the motors were strong enough to hold the puppet arm in a set position, torque calculations had to be done to each joint where a motor was being placed. This ensured the motor being chosen for each joint, would have enough torque to hold the arm in place past it. Using the following equation, the torque applied at each joint was found.

$$T = Fr \text{ where } F = mg$$

Equation 1: Torque Equation for Each Joint

The torque applied at each joint is equal to the force times the radius of motion, where force is equal to the mass of the parts being supported by the coefficient of gravity. The torques for each joint were found iteratively through the arm, starting at the end of the arm and working towards the base. This meant each motor was chosen specifically for what it had to hold. The mass for each joint contained the mass of the 3D printed part, any motors and encoders contained in the

joints further up the arm, and hardware to mechanically attach the arm, which included bearings, D-shafting, and shaft collars.

Motors for each joint were chosen starting at the end effector of the arm, and then working towards the base. This was done to minimize the size and torques of the motors at the end of the arm. This method was chosen after talking with last year's RoboPuppet team, which encountered problems with motors with excessive torque near the end effector. This made it difficult to manipulate the puppet arm by hand.

For the torque calculations, the length was assumed to be half of the length of each linkage. This provided a generalized distribution of where the weight for each piece was located. Also, the mass for each 3D printer part was assumed to be 50% to account for the 3D printing fill. When deciding each servo motor, a factor of safety of approximately 1.5 to 2 could be achieved with each motor to ensure there would be ample torque for the motor to support the linkages towards the end effector. Initially, the lower torque option will be used to minimize resistance while manipulating the arm. If more torque is needed, the higher torque value on each motor will be used.

The following table shows the calculated torques and motor specification chosen for the design. A python script was made to efficiently calculate each torque specification, where the additional masses could be added for each joint to account for any necessary hardware and the motor mass from the previous joint.

Part	Torque Required (from python script calculation) (Oz-in)	Torque of chosen motor (Oz-in)	Motor Mass (g)	Motor Voltage (V)
6	-	18.05	9	4.8
5	3.75	20.94	8	4.8
4	11	38.8	41.7	4.8
3	41.8	54.15	31	4.8
2	70.1	111.09	53	4.8
1	141	208	67	4.8
Base	202.6	240	60	4.8

Figure 6: Table of Torques and Motor Selections

The motors models chosen for each joint are listed below.

Part	Motor
Base	Bilda Torque
1	SC - 1268SG Black
2	HSR - 2648 CR
3	HS - 225MG
4	HSR- 1425 CR
5	HS - 53

Figure 7: Servo Model Choice

Design Iterations

Iteration 1: “Scaled” Arm

Both iteration 1 and iteration 2 were designed with the same intentions. Most, if not all major features and design goals such as friction reduction and ergonomic design were part of these iterations. The main and only difference between iteration 1 and 2 is that iteration 1 was designed to be scaled by a factor of $\frac{1}{2}$ to the Kinova Gen3 arm. This means that the distance from one rotary joint to one twist joint in the design would be half the length of the distance from a twist joint to a rotary joint in the Kinova arm. While this was a nice feature, the negatives heavily outweigh the positives. The larger and heavier body and links needed to provide a proper scaling resulted in high torque forces and a boxy/unappealing design.

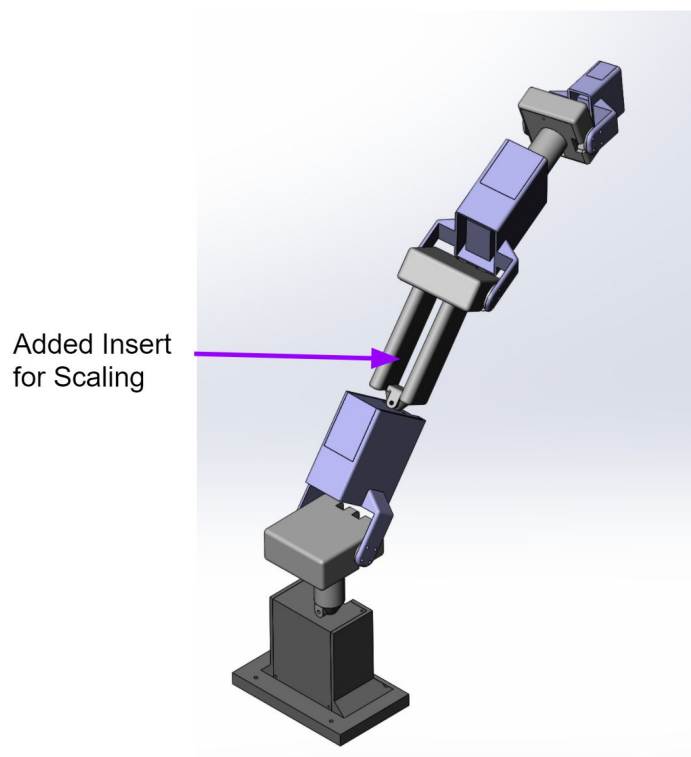


Figure 8: Iteration 1 Design

Iteration 2: “Un-Scaled” Arm 1.0

Due to the negative qualities of the scaled arm, an unscaled arm design was chosen to be more desirable for this project. Even though iteration 2 is not specifically scaled to the Kinova arm, it still has the same range of motion with all of the joints exhibiting the same angular movement, and in addition to this, the arm has the same degrees of freedom.

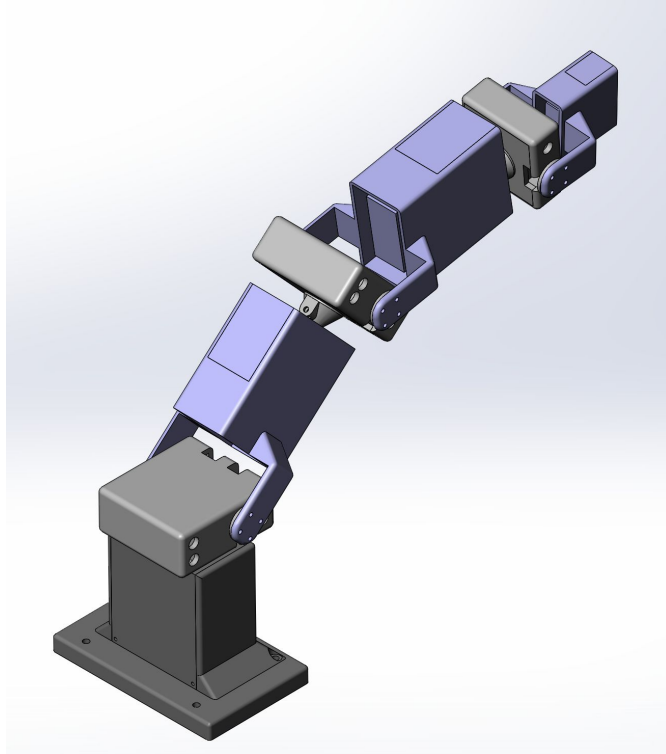


Figure 9: Iteration 2 Design

Iteration 3: “Un-scaled arm” 2.0

After working and experimenting with the initial build of iteration 2, several design flaws and multiple changes were made to ensure the arm’s performance was improved to desired performance. The most major change made was a modification to the rotary link which allowed for the mounting and installation of a hall encoder to be used to track the movement of the servo in said rotary link. Another major change was the “U-shaped extensions” which extruded off the bottom of the twist links were extended so that the twist link could clear the rotary link with no interference and the arm could bend in both directions. Along with that, several more cutouts were made in the cases and caps for each link type as there was more space needed for wiring as well as more access points for fastening bolts and screws. Also, a small x-shaped piece was added at the end of the arm to hold a button and rotate the encoder located in the final twist link. And finally, the final twist link was made larger to house an AMT222A Encoder.

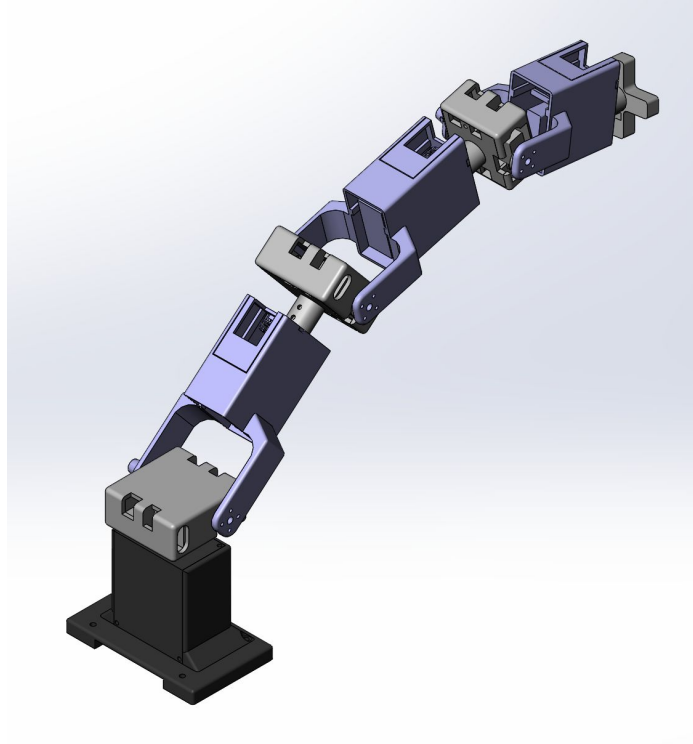


Figure 10: Iteration 3 Design

Design Issues

As stated in the previous section regarding Iteration 3, the assembly and testing of the first prototype (iteration 2) revealed multiple design errors that were corrected in iteration 3. This section will recap those errors as well as give more insight into them and their solutions.

The first and most major error is that the servo motors located in the rotary links had no system in place to track their rotation. It was initially concluded that the servos themselves would have hardware in place to do this but upon further inspection, it was determined that Hall Encoders would be needed to track their rotation. This meant that the rotary links would need an additional structure inside the case piece to mount and house the hall encoder. This mounting plate can be seen in Figure 1.

Another very important change connects directly to the first error described. In order for the Hall Encoder to track the movement of the rotary servo, the rotation of the entire connected twist link needed to be tracked. This was done by making it so the D-Shaft connecting the two links would rotate with the twist link and therefore have the same angular displacement. In order to do this, the hole on the U-Shaped extension that initially housed a ball bearing was modified to only hold the D-Shaft and lock it into place. This was done by reducing the hole diameter from ten mm to seven mm and adding an area for a set screw. This mounting hole can be seen in Figure 2.

A third crucial design change was one also made on the U-shaped extension of the twist link. The arms of this extension were made longer so that the twist link could undergo rotary motion in both directions. In the iteration 2 prototype, it could only rotate fully in one direction and the other direction's rotation was quickly cut short with interference between the rotary and twist link. By extending the length of the arms of the extension, more clearance space was created and rotation in both directions became possible.

Finally, a couple more minor errors were found. One error was that there were not enough holes for wiring to flow through. Several more gaps were added into the caps and cases of each link to allow for more economical organization and placement of wires. Another error was that that final twist link was not large enough to hold an AMT222A Encoder. This was simply fixed by increasing the size of the link case to that of twist link 4.

Mechanical Analysis

Torque Calculation Script ([torques.py](#))

In order to calculate the torques required for each servo, a Python script was utilized. The script was designed to calculate the required torques iteratively down the arm, starting with the servo in the end effector and ending with the servo in the base. Referencing equation 1, the script allowed the user to input the total mass associated for the torque equation.

The mass required for each servo includes the total mass of all parts in any of the above linkages, including 3D printed parts, other servos, encoders, shafts, bearings, shaft collars, and any other small parts. The script automatically inputs the mass of the 3D printed parts, which was listed if they were composed of solid PLA, and scaled it by a factor of one-half to account for printing density of the part. This scaling factor also accounted for some over-estimation, since most 3D-printed parts have an in-fill of 25%, meaning their total mass would be one-quarter of the solid mass. This meant the calculated torques based on the mass variable would be greater than required, ensuring each servo would meet the needed specification. The total mass of the remaining parts, excluding the 3D printed parts, can be manually added by the user for each iteration of the calculation process. Finally, the mass of the parts already calculated were also automatically added to the next step of the interaction process.

The script also automatically inputs the length required for each torque calculation. The length corresponded to the overall length between linkages, meaning where the parts connected to one-another. The overall length was then reduced by a scaling factor of one-half to account for the fact that the average mass in each part was roughly in the center of the part. This scaling factor also ensures there would be a small over-estimation of the required torque.

After the user has inputted the mass of each new part iteratively down the arm, the script outputs the required torque for each servo. These torque values can then be safely used to select a servo for each joint. By choosing servos specific to their required torque, the friction in each joint when moving the arm will be reduced, making the manipulation of the RoboPuppet arm easier. Also, the torque required for the end joints, will be much less than near the base, which means smaller servos, which tend to be lighter, can also be used, further reducing the total weight of the arm.

Minor Design Changes

While testing with the RoboPuppet arm, some minor design changes had to be added. The majority of them had to do with the overall support of the arm. The most notable were around the shaft connections. Due to the torque on the shaft between the base joint and the first linkage, this became a weak point in the arm. The arm tended to flex on the shaft connecting these two parts. In order to compensate for this, the cap of the 1st part, connecting the shaft from the base, was extended where the cap fit into the shaft. This allowed for two set-screws to be used to secure it to the shaft. It also meant more material was contacting the shaft, resulting in less bending at the connection. A bearing was also interested into the cap in the base to secure the shaft laterally, reducing the amount it could tilt between the connection from the base servo to the first linkage. After these changes were made, the torque applied to this joint no longer was a problem and the shaft stayed upright, as it was supposed to be.

Another weak point was the screws being used to attach the twist joints to the servo horn in the rotary joints. The screws used initially were not strong enough to firmly attach the twist joint to the servo horn. The size of these screws were increased to secure the connection between these parts. It also reduced the amount of play in each joint when the arm was placed in “hold” mode.

Electrical Design

The electrical design of this system was based on the 2019-2020 iteration of the RoboPuppet MQP’s design. An overview of the wiring connections is shown below in Figure 11.

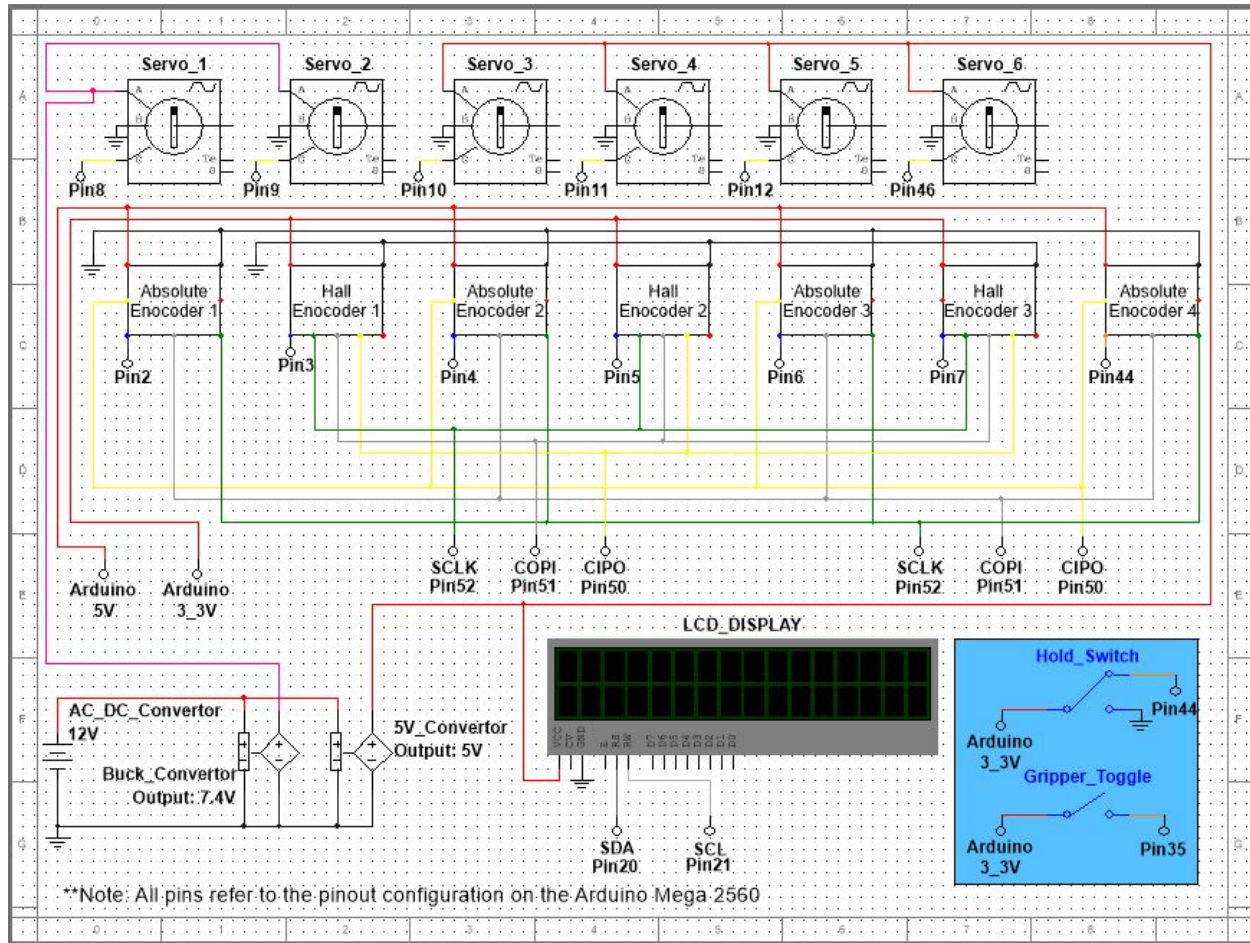


Figure 11: Hardware Schematic

When wiring the robot, it was decided to keep a color-coded system to ensure reliability and ease of use among all of the electronic components. This system was very useful given that all the encoders used SPI communication, and therefore had similar wiring requirements. The colors used are shown below, as well as in the schematic shown in Figure 11.

- Red: Power
- Black: Ground
- Blue: Chip Select
- Yellow: COPI
- White: CIPO
- Green: SCK

For each type of encoder a wiring system was devised to use one wire from each encoder's power, ground, COPI, CIPO, and CLK signals to lead to the Arduino. This reduced the total length of wiring compared to having independent lines for all of these signals from each of the encoders. This system was implemented separately on the hall encoders and the absolute encoders. Keeping each type of encoder separate also helped to isolate and debug issues relating to each type of encoder.

Microcontroller Subsystem

Sensors

RoboPuppet uses absolute encoders on four of the seven joints. On the twist joints with continuous rotation servos, the servos are not able to provide this data. Therefore, it was needed to choose absolute encoders that would be able to provide positional data for those joints. A AMT222A encoder was chosen for this due to its small size and straightforward implementation, which is pictured in Figure 12. They were placed on the shafts coming out of the base and twist joints.



Figure 12: AMT222A Absolute Encoders

The AMT222A Encoder has a 12 bit resolution, which can be read in using an SPI interface. This encoder has an accuracy of 0.2 degrees.

The second type of encoder used is a hall encoder. It was chosen due to its small form factor, which enabled it to be placed in each of the rotary joint housings. The hall encoder uses a microchip that senses the change of the magnetic field on a diametric radial magnet as it rotates

above the sensor. The magnet was attached to the end of the shaft that is a part of the rotary joint enclosure. The shaft is positioned such that the magnet is between 0.5 to 3mm from the chip, and centered above it. The hall encoder chosen is pictured in Figure 13.

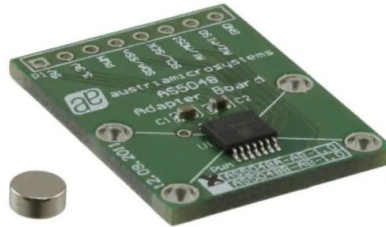


Figure 13: AS5048A Hall Encoders

The AS5048A Encoder has a 14 bit resolution, which can also be read in using an SPI interface. This encoder has an accuracy of 0.2 degrees.

Microcontroller

In order to choose a microcontroller, it had to meet the minimum requirements, listed below.

- 7x PWM pins
- SPI pins (SCK, COPI, CIPO)
- 7x Chip Select pins
- 4x Digital I/O pins
- 5V input pin compatible

The Arduino Mega 2560 and Teensy 3.5 microcontrollers both fit this requirement set. The Arduino Mega was then chosen for this project due to its easy integration with Visual Studio IDE. It has enough pins to handle all the required peripheral devices and has 5V tolerant input pins, which is required to read the PWM output of the servos. A diagram of the microcontroller, with all of the pinout assignments, is shown in Figure 14.

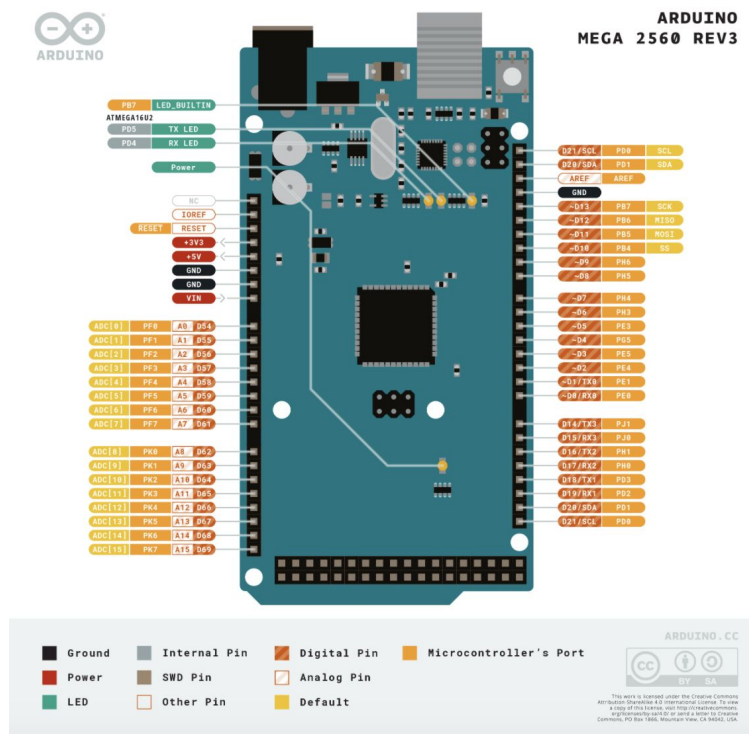


Figure 14: Arduino Mega 2560 Pinout Assignments

Power Distribution Subsystem

The system ran at three different voltage levels. The breakdown of each module and their power requirements are shown below.

Component	Voltage (V)	Current (mA)
Servos	4.8 - 7.4	150 - 2,000
Absolute Encoders	5	16
Hall Encoders	3.3 - 5	15
Microcontroller	5	45

Figure 15: Table of Module Power Requirements

In order to achieve these distinct voltage levels, a 12V power source connected to a 5V regulator and a Buck Converter set to 7.4V. The main power to the system is provided by a 12V 360W

power supply, as shown in Figure 16. This power supply takes 120V wall power and converts it to 12V.



Figure 16: 12V DC Power Supply

A 5V, 3A power supply, shown in Figure 17, takes in power from the 12V rail and converts it to 5V to power servos three, four, five, and six.

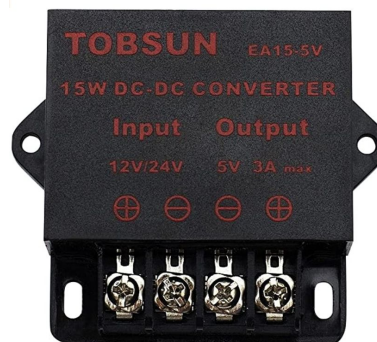


Figure 17: TOBSUN 5V DC Power Supply

An adjustable buck-boost converter is also included to provide power to the servos. This converter takes power from the 12V power supply and can output 0-32V at up to 240W. This module will be configured to output 7.4V to power servos one and two at the base of the arm to achieve the necessary torque requirements.

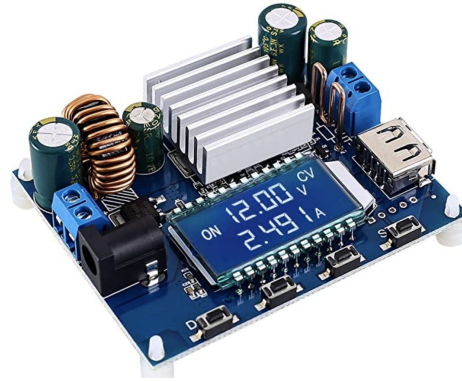


Figure 18: WHTDS DC Buck Boost Power Supply Module

The absolute encoders are powered from the Arduino Mega's 5V supply. This means regardless of a power supply issue, the encoder will always always read in the absolute position of these joints. The hall encoders are powered by the Arduino Mega's 3.3 supply for the same reasons. The circuitry for the hold and gripper toggles are also powered by the Arduino Mega's 3.3V supply.

PCB Design

In order to create a cleaner base module and reduce errors with wiring, it was chosen to create a PCB for the final design. Despite being a very simplistic design, it will help with the wiring in the base. Through testing, some issues arose with the wiring in the breadboard. It was found the jumper cables would pull out affecting reading in encoders or having a continuous ground plane. These issues led to testing and delays that could have been avoided. Once the final wiring was known, the following PCB trace layout was created in order for a board to be fabricated. The PCB incorporates all of the leads needed for two RoboPuppet arms to be controlled from the Arduino Mega 2560.

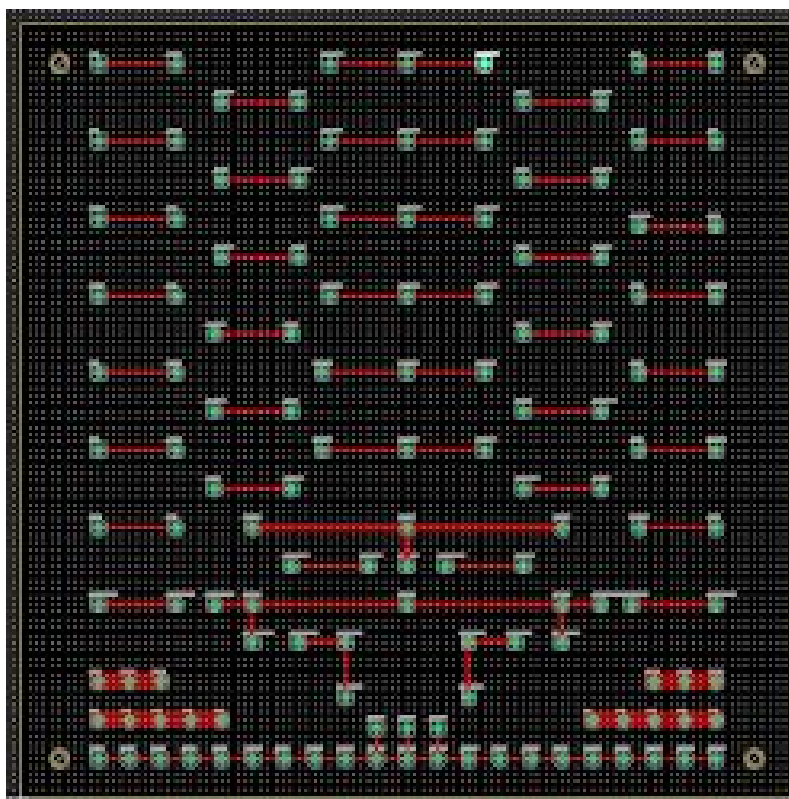


Figure 19: PCB Trace Layout

The PCB is designed to allow for more robust and efficient wiring. When the RoboPuppet arm was manipulated during testing, it caused some of the jumper cables to be slightly pulled out from the breadboard, breaking the connection. By having soldered on cables, this problem will be avoided. It also will isolate any possible disconnects between jumpers and headers to the junction between the arduino and PCB.

Another benefit is that the silkscreen layer is labelled where each connection should be placed, making it easy to identify which wires go to which part on the arm. This will eliminate the need for the use of “flags” on the wires connecting to the breadboard for labelling purposes; these flags created clutter on the breadboard.

When designing the PCB, a few important factors were kept in mind. One of the most important was to keep a continuous ground plane to ensure all the grounds are connected. On the PCB, all ground connectors are unified with a single trace, meaning only one ground pin has to then go to the Arduino. Another important factor was to limit the SPI connections, CIPO, COPI, and CLK, to short traces on the board. This is due to the fact the SPI signal can degrade over longer traces. Along with this, the PCB ensures that the power lines are kept a significant distance away from the SPI connections to avoid interference issues. The PCB is designed to have each arm’s

connections on one half of the board. This will help with overall organization and troubleshooting. With each arm on one side of the board, any electrical problems that occur will be able to be easily isolated.

Overall, the PCB has all the required connections to be compatible with two RoboPuppet arms and to connect back to the Arduino Mega 2560. This includes all the connections for the servos, encoders, gripper toggle, hold switch, and LCD screen. The PCB will streamline and make more robust the wiring system for the RoboPuppet Arm.

Firmware ([Firmware/](#))

The RoboPuppet embedded firmware runs on a Teensy 3.5 microcontroller. The firmware was written in C++ and uploaded to the Teensy Microcontroller to interface with the hardware. This was developed in Visual Studio Code using the platformio.ini plugin. The firmware will handle reading joint angles from the encoder hardware, reading gripper button values, and maintaining communication with the ROS nodes via serial. The firmware consists of a main function as well as several subsystems.

Main Control Program ([src/Main.cpp](#))

The main functions consist of Arduino `<setup>` and `<loop>`. The setup function initializes all of the firmware subsystems. The loop function repeatedly tells each individual subsystem to update and then sends their data to the simulation.

Firmware Subsystems ([sub/](#))

The RoboPuppet firmware is divided into subsystems for code organization and readability. The subsystems are listed and briefly below in Table 20.

Name	Description
Abs Encoders	Reads angles from rotary encoders
Hall Encoders	Reads angles from hall encoders
Info LCD	Outputs debugging information to LCD screen
Buttons	Reads push button values
Servos	Sets servos to the appropriate angles
Serial Interface	Communicates RoboPuppet joint and state data to the simulation
Config	Contains configuration settings and helper functions used throughout the firmware

Figure 20: Table of Firmware Subsystem Descriptions

A diagram of how the firmware subsystems are shown below in Figure 21. The rounded blocks represent physical systems while the squares represent firmware subsystems.

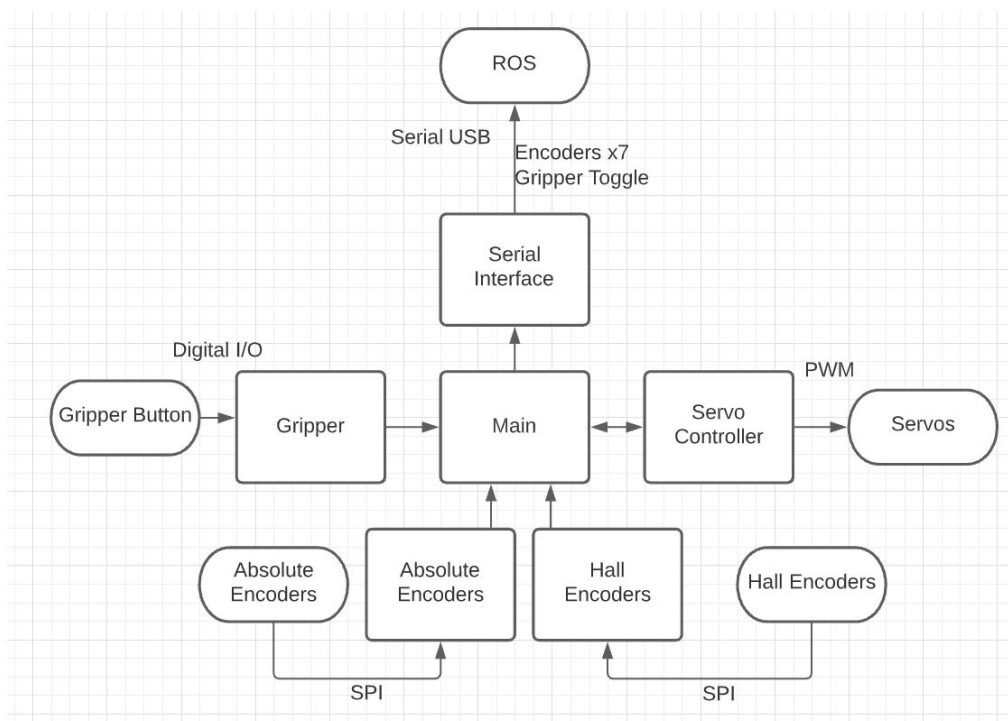


Figure 21: Block Diagram for Firmware Subsystems

Absolute Encoders Subsystem ([absEncoders/](#))

This subsystem manages the absolute encoders on the four twist joints (0, 2, 4, 6). The encoder position is received in a 12 bit value. This value, which ranges from 0 to 4096, is extracted from the 12 bit value using the methodology shown in the CUI Devices sample code for the AMT222A encoder. This position value is then mapped to an angle value between -180 and 180. This angle value is converted to radians when fed into the Trina Simulations.

Buttons Subsystem ([Buttons/](#))

This subsystem reads both buttons present on the RoboPuppet: the button to open and close the gripper and the button to hold the puppet in place. Each button press toggles the current state of those two systems. It consists of two separate get functions, one for each function, which returns the state of the toggle for that specific function.

Hall Encoders Subsystem ([hallEncoders/](#))

This subsystem interfaces with the three rotary joints (1, 3, 5) to control the motors and read in their positional values. The init function connects each servo object to its correct pins, and initializes all angle readings to zero. The update function will return the positional value for the specific motor in degrees, which will be converted to radians when fed into the Trina Simulation. This [library](#) was used to read in values from the AS5048A Hall Encoders

Servos Subsystem ([Servos/](#))

This subsystem interfaces with the three rotary joints (1, 3, 5) to control the motors and read in their positional values. The init function connects each servo object to its correct pins, and initializes all angle readings to zero. Each servo has different PWM ranges to access their range, which are listed below.

Part	Servo Type	PWM Range
Base	Continuous	Still at 1500 uS
1	180 degree range	800 - 2200 uS
2	Continuous	Still at 1500 uS
3	180 degree range	553 - 2520 uS
4	Continuous	Still at 1500 uS
5	180 degree range	553 - 2270 uS
6	-	-

Figure 22: Table of Servo PWM Ranges

LCD Info Subsystem ([SerialInterface/](#))

The LCD Info Subsystem works as an additional debugging and informational view for the user or developer.

Serial Interface Subsystem ([SerialInterface/](#))

This subsystem manages the communication of data over serial to the python script bridge running on the host machine.

Config Subsystem ([Config/](#))

This subsystem manages the configuration of pin constants and encoder calibration helper functions throughout the firmware. This enabled the easy configuration of pins in one central file.

Serial Message Protocol

After receiving feedback from the previous year's interface between the microcontroller and ROS host machine, it was opted to write a custom protocol instead of sending ROS messages directly between the two systems. The custom protocol sends the necessary position and status data over serial from the microcontroller to a python bridge script on the host machine that publishes the received serial data to a ROS topic. This method removes the overhead required with sending a ROS message over serial.

The message itself is composed of a simple comma separated string of encoder values arranged as follows (this protocol is subject to change as further requirements are found):

Arm 1 data is the first 9 values, and a second set of data can be sent over for use with a second arm.

Data Type	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16
Arm 1 Data	Servo 1	Servo 2	Servo 3	Encoder 1	Encoder 2	Encoder 3	Encoder 4	Gripper Status	Hold Status

Figure 23: Table of Arm 1 Serial Communication Variable Types

Data Type	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16	u_int16
Arm 2 Data	Servo 1	Servo 2	Servo 3	Encoder 1	Encoder 2	Encoder 3	Encoder 4	Gripper Status	Hold Status

Figure 24: Table of Arm 2 Serial Communication Variable Types

ROS Software ([Catkin/src/RoboPuppetMQP/](#))

The ROS software is written for ROS Melodic on Ubuntu 18.04. It includes launch files, UI design files, python nodes, and support classes. The software communicates with the embedded controller through a serial interface. It is also connected to the simulated Kinova Gen3 arm in Gazebo simulation via ROS messages. In addition, users are able to run a RoboPuppet control GUI on the host machine. The GUI provides performance information of the Kinova arm and also enables users to control the simulated arm.

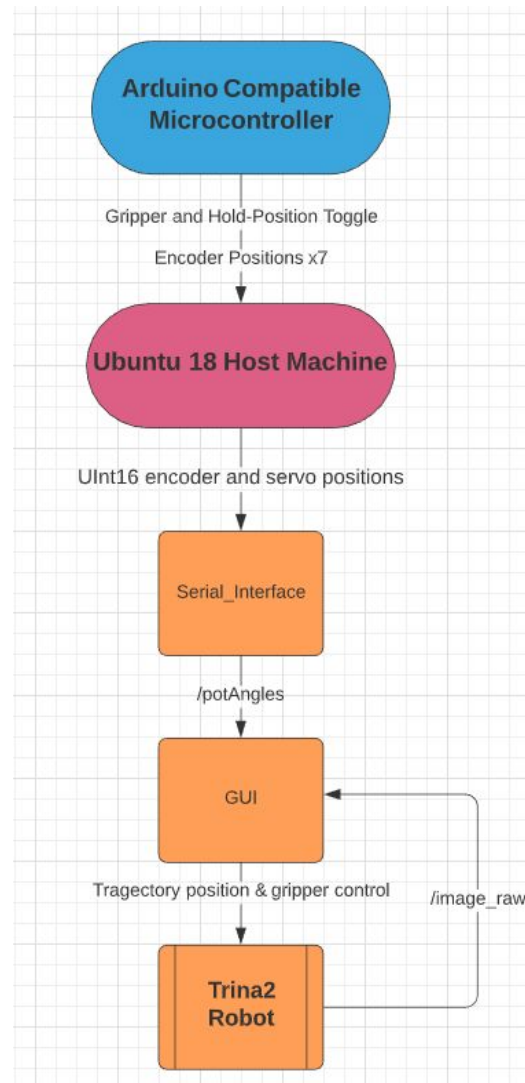


Figure 25: Block Diagram for ROS Information Nodes

The block diagram for the ROS Kinova software is shown in Figure 25. The orange squares represent different scripts that exchange the information shown on the lines by using different ros topics and nodes.

Launch File Structure ([launch/](#))

This folder contains the launch files that will load all the necessary ROS nodes for use. The ‘spawn_kortex_robot.launch’ is used to bring up the gazebo environment for single Kinova Gen3 Arm. ‘trina2.launch’ and ‘trina2_single.launch’ are both used to bring up the Trina2 robot environment. These two launch files can also be found in the Trina2 package. The purpose of these files was to keep the version the same during the project, knowing the Trina2 package was constantly updated.

ROS Source Code ([src/](#))

This section will introduce the ROS source code divided by file. The code includes ROS nodes and other support files.

Constants File ([constants.py](#))

The constants file contains a list of constants which are used by multiple other python source files.

Login Interface ([gui_login.py](#))

This is a login interface for the users of RoboPuppet. By inputting a username and password, users are able to login and get access to control the robot with RoboPuppet. There are two different types of GUI provided --- GUI Lite and GUI Pro. Depending on the user type, the system will automatically pop up the correct version. In addition, this page also provides a sign up button for the first time user and an about menu bar where users are able to get an introduction of Trina2 as well as of the RoboPuppet system.



Figure 26: User Login Interface

Sign Up Interface ([gui_signup.py](#))

This page allows users to register an account for the GUI. They will need to provide information including username, password, default robot and user type (Regular or Admin). After signing up, the system will automatically jump to the login page. Currently, all the user information is stored locally in a json file named 'userlist.json'. In the future, an online database might be a better idea because of the security and efficiency reasons.

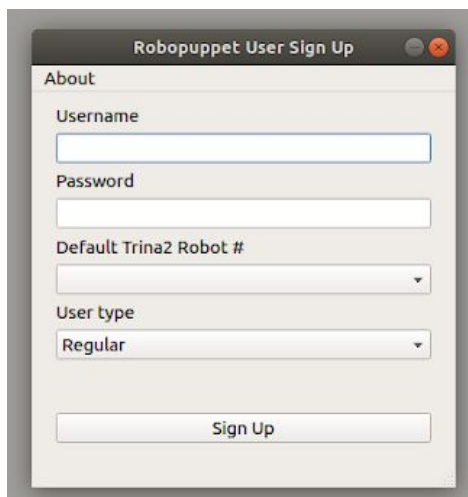


Figure 27: User Sign Up Interface

GUI Lite ([gui_lite.py](#))

This is an intuitive one page GUI for RoboPuppet. In this page, the users are able to modify RoboPuppet’s setting, including arm selection, mirror/reverse option and enable/disable connection. Depending on the arm selection, all angle, velocity and cartesian pose data of that arm will be displayed on the right. A camera view button is also provided in case users want to see the camera view. By clicking on the button, a camera view window will pop up. In this way, the user can see both windows at the same time and close the camera view at any time.

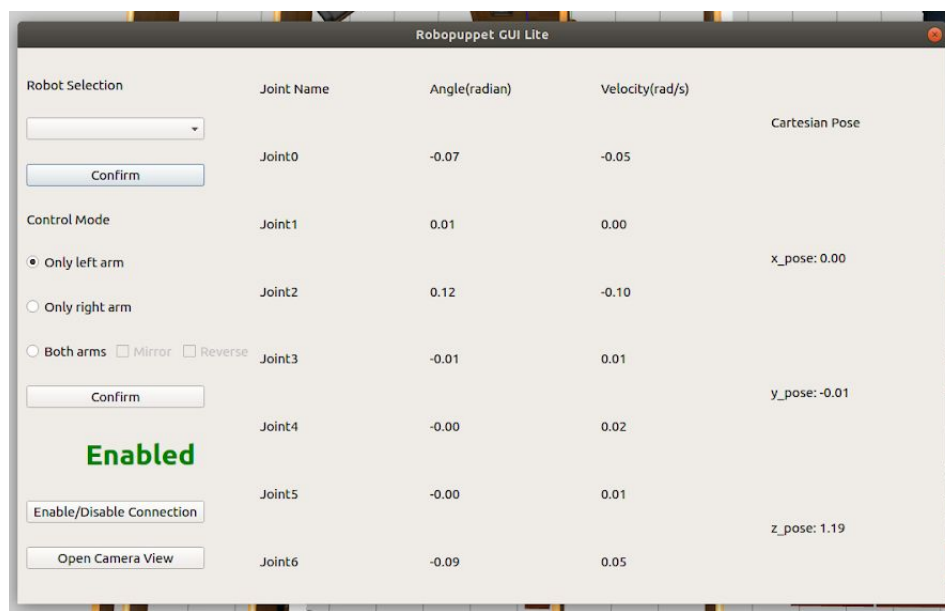


Figure 28: GUI Lite Interface

GUI Pro([gui_pro.py](#))

Compared to GUI Lite, GUI Pro is a more comprehensive and detailed interface. There are six main tabs in the interface: Info, Control, RoboPuppet Settings, Learn, Plot and Camera View.

Joint Name	Angle(radian)	Velocity(rad/s)	Cartesian Pose
Left-Arm			
Joint0	-0.08	-0.10	
Right-Arm			
Joint1	0.01	-0.00	x_pose: 0.00
Joint2	0.13	-0.03	
Joint3	-0.01	0.01	y_pose: -0.01
Joint4	-0.01	-0.01	
Joint5	-0.00	-0.00	z_pose: 1.19
Joint6	-0.10	-0.08	

Figure 29: GUI Pro Interface

Info: As shown in Figure 29, presents the basic information of both robot arms, including joint angle, joint velocity and cartesian pose of end effector. The unit here is radian and rad per second since rad per second is the SI unit of angular velocity. The update rate for the information is once every 100ms.

Control: This tab allows users to control each of the arm joints as well as each gripper. Users will need to input the desired angular position in radians and hit set angle. A ‘Home Kinova Arm’ button is also available to move the robot arms to the default position. For users to refer, a ‘Get Current Angle’ button is provided, which will automatically fill the input text boxes with current joint angle values. In addition to that, the current cartesian position of end effector is displayed. Users can also hit the ‘Get Estimated Position’ to preview the position before actually sending it. Besides that, users can control the gripper using the slider on the bottom right corner. There are 10 steps for each slider.

RoboPuppet Settings: This tab allows users to modify the settings related to RoboPuppet. Users can choose the arm they want to control --- left, right or both. They can also enable/disable the RoboPuppet depend on the situation.

Learn: This tab allows users to record and replay a set of robot arms motions. Users will need to hit the record button to start recording and stop button to stop recording. Then, hitting the play button will replay these motions.

Plot: This tab provides visual performance history for the users to view. Users are able to see the plot of joint angles and velocities over time through plots.

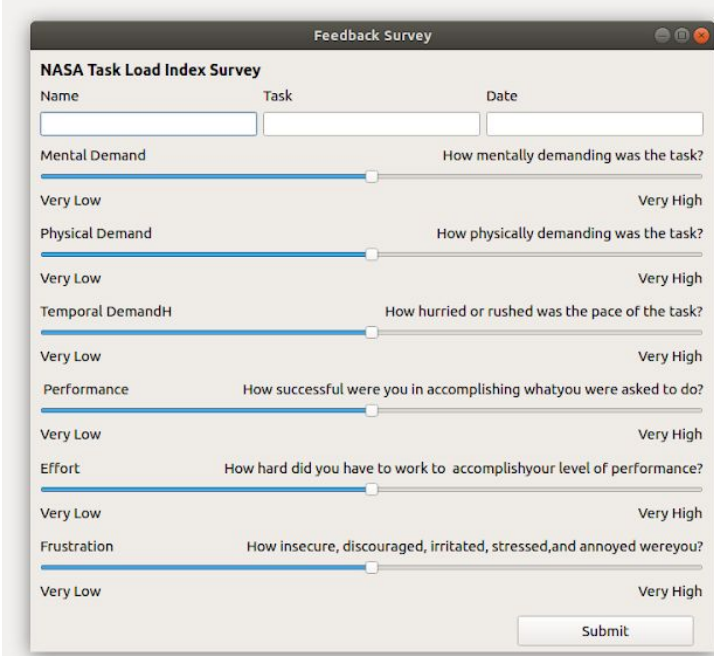
Camera View: This tab provides camera views from three cameras on the robot --- main camera, camera on the left arm and camera on the right arm.

Kinematics Class ([kinematics.py](#))

This class contains all transformation tables for each joint and provides a function to calculate the forward kinematics for the Kinova Gen3 arm.

Feedback Survey([gui_feedback.py](#))

This is a page which reserves for the potential user testing in the future. The evaluation method we choose here is the NASA Task Load Index(NASA-TLX). This is a widely used way to rate the overall workload of a certain task through six different aspects --- Mental Demands, Physical Demands, Own Performance, Effort, and Frustration. Users are able to access this page from the login page and the result will be stored in 'survey_result.json'.



The image shows a screenshot of a web-based survey interface titled "Feedback Survey". The main heading is "NASA Task Load Index Survey". At the top, there are three input fields labeled "Name", "Task", and "Date". Below these are six horizontal sliders, each representing a different aspect of the task load index. Each slider is labeled with a category on the left and a question on the right. The categories and questions are: "Mental Demand" (How mentally demanding was the task?), "Physical Demand" (How physically demanding was the task?), "Temporal DemandH" (How hurried or rushed was the pace of the task?), "Performance" (How successful were you in accomplishing what you were asked to do?), "Effort" (How hard did you have to work to accomplish your level of performance?), and "Frustration" (How insecure, discouraged, irritated, stressed, and annoyed were you?). Each slider has "Very Low" on the left and "Very High" on the right. A "Submit" button is located at the bottom right of the form.

Figure 30: NASA-TLX Feedback Survey

GUI Design Folder(ui/)

This folder is a collection of the ui design files for all the GUIs introduced above. There are two types of files in this folder one is .ui file and one is the corresponding .py file. The .ui files are written in XML format which is used to provide the layouts and components for the GUI interface. It can be opened and edited through Qt Designer. In this way, the developers will not need to type any code, they can simply use the design function in the Qt Designer and the code will automatically be updated.

The .py files are python versions of .ui files. It is generated through the command 'pyuic5 <filename>.ui -o <filename>.py'. Every time you modify the UI design, you will need to manually update this python file. The reason for this extra step is to make it easier for the ui design class to be imported into the GUI node. It will also be easier for the developer to call the UI components from a python class.

Trina2 Environment

The Trina2 Simulation environment was also used in addition to the Kinova Simulation. This simulator allowed us to imagine the Kinova Arm in a more realistic hospital setting, as shown in Figure 31. This was integral to testing the usability and effectiveness of the RoboPuppet controller.

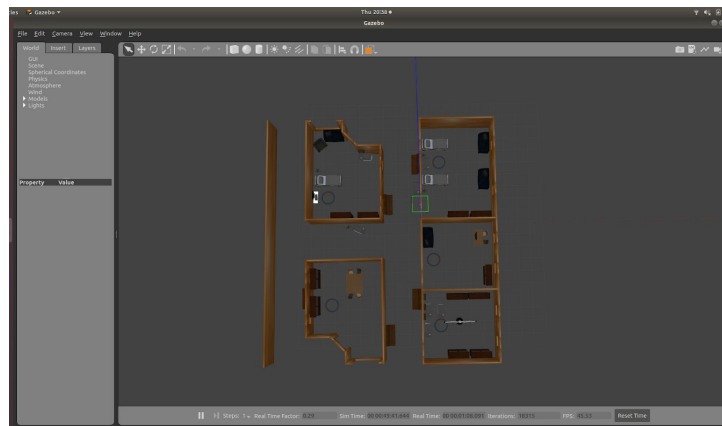


Figure 31: Basic Trina2 Environment

ROS Topics and Services

ROSTopic	Type	Description
trina2_1/right_arm_cam/color/image_raw	Image	Camera data from right arm camera
trina2_1/left_arm_cam/color/image_raw	Image	Camera data from left arm camera
trina2_1/main_cam/color/image_raw	Image	Camera data from main arm camera
trina2_1/right_arm_joint_{1..7}_position_controller/command	Float64	Desired joint position for right arm in radians
trina2_1/left_arm_joint_{1..7}_position_controller/command	Float64	Desired joint position for left arm in radians
trina2_1/right_arm_robotiq_2f_85_gripper_controller/gripper_cmd/goal	GripperCommandActionGoal	Desired gripper position for right gripper from 0-1 (0 is open, 1 is close)
trina2_1/left_arm_robotiq_2f_85_gripper_controller/gripper_cmd/goal	GripperCommandActionGoal	Desired gripper position for left gripper from 0-1 (0 is open, 1 is close)
/potAngles	JointPositions	Angles from the RoboPuppet in degrees (-180 to 180)

Figure 32: Table of Ros Services and Topics

There is only one ROS service used in this project, which is ‘gazebo/get_joint_properties’. This service is used to get the joint position of the joints on the robot arm. For example, if you want the position of the first joint of the left arm, you will send the parameter of ‘trina2_1/left_arm_joint_1’.

The previously mentioned Python bridge script publishes the data received from the puppet to a ROS Topic called potAngles using the custom message JointPositions. Another Python script takes the values published to this topic and pushes them to the joints in the Gazebo simulation.

Conclusion

This year, the system was redesigned for use with the Kinova Gen3 Robotic Arm. This redesign included the creation of a new puppet arm to match the Gen3's 7 degrees of motion, a rewrite of the firmware code to fit the new puppet arm and different electronic components, and a redesigned ROS GUI to provide information and control to users. It was not possible to attempt any user testing with the new system due to the COVID-19 pandemic.

Part of the team (Mario Castro, Francesco De Leo, and Isabella Feeney) will continue the project into C term 2021 and their goals include:

- Creating a second arm
- Incorporating vibration feedback
- Creating a base to mount both arms and all the electrical components
- Moving the hold mode switch to the gripper of the arm

Appendices

A) Acknowledgements

We would like to thank our advisors Jane Li, Michael Gennert, and Jie Fu for their extensive support on this project. In addition, team members Michael Sidler and Tabitha Gibbs of the 2019-2020 RoboPuppet MQP team have been extremely helpful in answering questions about previous iterations of this project as well as in troubleshooting any issues we encountered.

B) Kinova Launch Instructions for Ubuntu 18.04

Git clone and setup TRINA-WPI-2.0 before using the RoboPuppet Arm:

<https://github.com/hiro-wpi/TRINA-WPI-2.0>

Setup

1. Go to your ros workspace
2. Enter 'src' folder
3. Create a new package
4. Enter that package
5. git clone <https://github.com/CluelessTimeTraveler/RoboPuppetMQP.git>

Connect to Serial Port

1. Connect the RoboPuppet via USB port
2. Check the port name, if it is not '/dev/ttyACM0', then modify line31 in the SerialToRosMessage.py file accordingly.
3. use roscore and rosrund SerialToRosMessage.py to see if the serial connection succeed

Use RoboPuppet

1. If everything set up correctly, use roslaunch RoboPuupetMQP trina2.launch
2. Hit the play button in Gazebo
3. Login use 'admin admin' or 'user user' or sign up one

Potential Problems during setup

1. Missing Module. If you meet this problem, try apt-get that module, and double check the instruction in Trina2 repository
2. Serial port disabled, use command 'sudo chmod 666 /dev/ttyACM0' or find the appropriate port name

C) Important Links

Github: <https://github.com/CluelessTimeTraveler/RoboPuppetMQP>

D) Parts List

Part Name	Part Link
Part 5 Motor	https://www.servocity.com/hs-53-servo/
Part 4 Motor	https://www.servocity.com/hsr-1425cr-servo/
Part 3 Motor	https://www.servocity.com/hs-225mg/
Part 2 Motor	https://www.servocity.com/hsr-2648cr-servo/
Part 1 Motor	https://www.servocity.com/sc1268sg-be-high-torque-digital-servo-black-edition/
Base Motor	https://www.servocity.com/2000-series-dual-mode-servo-25-2/
Base Motor Board	https://www.servocity.com/3102-series-dual-mode-servo-programmer-1-1/
Encoders	https://www.mouser.com/ProductDetail/CUI-Devices/AMT222A-V?qs=17cgNqFNU1iC5HUmXUbG%252Bw%3D%3D
D-Shaft	https://www.servocity.com/0-250-1-4-x-2-00-stainless-steel-d-shafting/
24T Servo Shaft Coupler	https://www.servocity.com/c1-spline-servo-to-1-4-shaft-coupler-set-screw/
24T Servo Shaft Coupler	https://www.servocity.com/3f-h25t-spline-servo-to-1-4-shaft-coupler-set-screw/
Bearing	https://www.sparkfun.com/products/13012
Printer Filament	https://www.amazon.com/Printer-Filament-SUNLU-Dimensional-Accuracy/dp/B07XG3RM58/ref=sr_1_3?crid=YCMEWCSBOE8V&dchild=1&keywords=1.75mm+pla+filament+1kg&qid=1600890188&srefix=1.75mm+%2Caps%2C169&sr=8-3
Shaft Collar	https://www.servocity.com/0-250-1-4-aluminum-clamp-collar/
5V DC Converter	https://www.amazon.com/BINZET-Converter-Regulator-Regulated-Transformer/dp/B00J3MHRNO/ref=pd_bxy_img_2/141-3771859-7239818?_encoding=UTF8&pd_rd_i=B00J3MHRNO&pd_rd_r=b4043d65-aa82-4bcb-b5db-bea5e2c790ff&pd_rd_w=j0yMH&pd_rd_wg=6nKZk&pf_rd_p=ce6c479b-ef53-49a6-845b-bbbf35c28dd3&pf_rd_r=03RFSZMWZ6F110C6DYZR&pvc=1&refRID=03RFSZMWZ6F110C6DYZR
12V Power Supply	https://www.amazon.com/SUPERNIGHT-Universal-Transformer-Industrial-Automation/dp/B007MWNF5Q
Voltage Converter	https://www.amazon.com/DROK-Converter-5-3V-32V-Regulator-Transformer/dp/B078Q1624B/ref=pd_lpo_23_t_1/141-3771859-7239818?_encoding=UTF8&pd_rd_i=B078Q1624B&pd_rd_r=f0308010-8dad-4529-bf86-44

	34720f05d8&pd_rd_w=CpnJj&pd_rd_wg=KOTTF&pf_rd_p=7b36d496-f366-4631-94d3-61b87b52511b&pf_rd_r=FGA58W2M3JP5TX70JFAN&psc=1&refRID=FGA58W2M3JP5TX70JFAN
Teensy 4.1	https://www.amazon.com/PJRC-Teensy-4-1-Without-Pins/dp/B088D3FWR7/ref=sr_1_1?dchild=1&keywords=teensy+4.1&qid=1601581863&sr=8-1
Arduino Mega 2560	https://www.amazon.com/ARDUINO-MEGA-2560-REV3-A000067/dp/B0046AMGW0
Header Pins	https://www.amazon.com/MCIGICM-Header-2-45mm-Arduino-Connector/dp/B07PKKY8BX/ref=sr_1_3?dchild=1&keywords=header+pins&qid=1601581895&sr=8-3
30g wire	https://www.amazon.com/TUOFENG-Wire-Solid-different-colored-spools/dp/B07G2SWB19/ref=sr_1_3?dchild=1&keywords=22g%2Bwire&qid=1601582800&sr=8-3&th=1
Magnetic Encoders	https://www.digikey.com/en/products/detail/ams/AS5048A-TS_EK_AB/3188612
Screws	https://www.amazon.com/Stainless-Hexgon-Socket-Metric-Assortment/dp/B07VZTSHB4/ref=sr_1_5?crd=3ACLAJP4EVIN4&dchild=1&keywords=m2+m3+screws&qid=1603413248&srefix=m2+m3+s%2Caps%2C192&sr=8-5
Absolute Encoder Molex Connectors	https://www.mouser.com/ProductDetail/Molex/502578-0600?qs=3OKVfsn1b5Ax%252B4TT0aiBNw%3D%3D