Armold Final Report

Andi Fiani, Chris Steiner, Dylan Dorman, Shelby Schipper
MDS 430, 2024

Table of Contents

1. Executive Summary **4**
2. Project Introduction **4**
2.1. Problem and Opportunity 4
2.2. Background Knowledge 5
3. Stakeholder Needs **6**
3.1 Identification of Stakeholders 6
3.2 Stakeholder Needs and Requirements 6
3.2.1 Users 6
3.2.2 Keepers 7
3.2.3 Creators 8
3.2.4 Functional Range of Motion Requirements 9
4. System Overview **10**
5. Presentation of the Design **10**
5.1 Mechanical System 11
5.1.1 Armold's Spine 11
5.1.1.1 Spine Overview 11
5.1.1.2 Spine Assembly Process 12
5.1.1.4 Spine Suggestions, Notes, Improvement Opportunities 13
5.1.2 Armold's Shoulder 13
5.1.2.1 Frontal Raise Actuator 15
5.1.2.1.1 Frontal Raise Actuator Assembly 16
5.1.2.1.2 Frontal Raise Actuator Maintenance 18
5.1.2.1.3 Frontal Raise Actuator Improvement Opportunities 18
5.1.2.2 Lateral Raise Actuator 19
5.1.2.2.1 Lateral Raise Actuator Assembly 19
5.1.2.2.2 Lateral Raise Actuator Maintenance 20
5.1.2.2.3 Lateral Raise Actuator Improvement Opportunities 20
5.1.2.3 Cross-Body Actuator 21
5.1.2.3.1 Cross-Body Actuator Assembly 21
5.1.2.3.2 Cross-Body Actuator Maintenance 22
5.1.2.3.3 Cross-Body Actuator Improvement Opportunities 22
5.1.3 Armold's Elbow and Pronation 23
5.1.3.1 Elbow and Pronation Overview 23
5.1.3.2 Elbow and Pronation Assembly Process 24
5.1.3.3 Elbow and Pronation Maintenance 26
5.1.3.4 Elbow Suggestions, Notes, Improvements Opportunities 26
5.1.4 Armold's Hand 27
5.1.4.1 Hand Overview 27
5.1.4.2 Hand Assembly Process 28
5.1.4.3 Hand Maintenance 31
5.1.4.4 Hand Suggestions, Notes, Improvement Opportunities 31
5.1.5 Armold's Control Panel 32

5.1.5.1 Control Panel Overview..... 32
5.1.5.2 Control Panel Assembly Process..... 32
5.1.5.3 Control Panel Maintenance..... 32
5.1.5.4 Control Panel Suggestions, Notes, Improvement Opportunities..... 32
5.2 Electrical System..... 33
5.2.1 Armold's Electronic System..... 33
5.2.1.1 How Everything Works..... 33
5.2.1.2 Specifics Regarding Motors Used/Armold's Current Budget..... 37
5.2.1.4 Electrical System Suggestions, Notes, Improvement Opportunities..... 38
5.2.2 Control Panel Electronic System..... 38
5.3 Communication System..... 39
5.3.1 Communication System Overview..... 39
5.3.2 Communication System Maintenance..... 39
5.3.2 Communication System Suggestions, Notes, Improvement Opportunities..... 39
5.4 Software..... 40
5.4.1 Software Overview..... 40
5.4.1.1 armold.py..... 41
5.4.1.2 robot.py..... 41
5.4.2 Software Maintenance..... 42
5.4.3 Software Suggestions, Notes, Improvement Opportunities..... 42
6. Evidence of Performance..... 43
6.1 User Requirement Performance..... 43
6.2 Keeper Requirement Performance..... 44
6.3 Creators Requirement Performance..... 44
7. Operating Armold..... 46
7.1 The Basics (Startup and Operation)..... 46
7.2 Troubleshooting..... 46
8. Recommendations and Final Notes..... 46
8.1 Additional Information..... 46
8.2 Additional Armold Improvement Ideas..... 47

1. Executive Summary

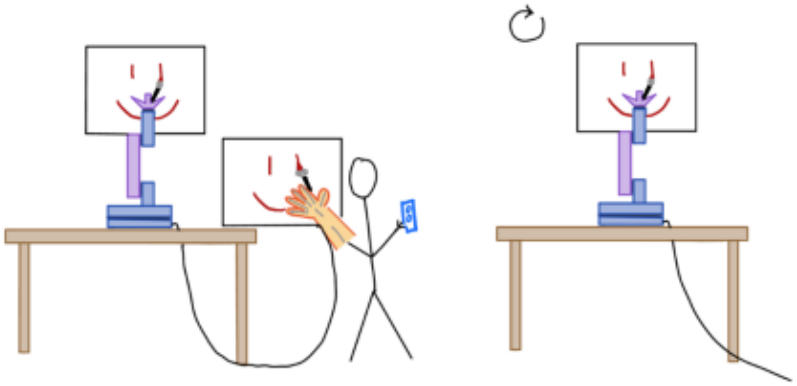
Armold is a humanoid robotic arm and control panel system sponsored by Rose-Hulman. He will remain at Rose-Hulman indefinitely as an interactive display, where he may be operated for the enjoyment of current and prospective students or other members of the Rose-Hulman community. The specifics of his final resting place are yet to be determined, though it is likely he will remain in J102, the Engineering Design Studio classroom. Overall, the goal of Armold is to provide a fun and professional source of inspiration to Rose-Hulman students. The team wants to show them what they are capable of creating when putting their acquired engineering skills and knowledge to the test. Armold is also intended to be a showpiece for incoming students visiting Rose-Hulman on tour. We hope that seeing Armold inspires prospective students to express their creativity through the pursuit of similar engineering projects.

2. Project Introduction

2.1. Problem and Opportunity

In manufacturing and other fields, there is a need for robots that are versatile, adaptable, and easy to use. It is undeniable that human arms and hands are more capable- at least, in terms of versatility and fine motor skills- than robots as they exist today. As such, many are interested in developing a humanoid robot, though this is an incredibly daunting task due to the complex nature of the arm and motions it is capable of.

Since a year-long senior capstone project is not the appropriate forum to create a robot from scratch which is fine-tuned enough for use in industry, our team settled on creating a "proof-of-concept" version of this robot which we have endearingly dubbed "Armold". Our vision for Armold was largely based on the ideas presented above. We wanted him to resemble a human arm in structure, and to be able to perform various motions which could be repeated and replayed on a loop. Initially, we were hoping to create a control sleeve that the user would simply wear to operate it. In our minds, that was the peak of user-friendliness.



The initial vision of Armold. The image depicts a user wearing a control sleeve and drawing a smiley face, while Armold performs the same motion as the user.

However, due to time constraints, our team chose to forgo the control sleeve since it introduced so much complexity. Instead, we settled for a control panel that users manipulate to control Armold.

Our "proof-of-concept" version of Armold will remain at Rose-Hulman as a display to showcase both an example of a Multidisciplinary Capstone project and to boast the Engineering Design program. This project represents what Rose students can create when they apply the engineering skills they have learned over their time at Rose-Hulman. Overall, Armold aims to:

- Engage prospective students in a fun way
- Show current students what they are capable of creating
- Promote the Engineering Design program

2.2. Background Knowledge

The creators of Armold had a limited (no) amount of robotics knowledge before pursuing this project. Since the majority of the team was composed of Engineering Design students, we *did* have experience managing a project from beginning to end and being resourceful. It must be noted that many decisions were made based on information: found online, obtained from Rose students and staff, and experience testing. The team acknowledges that there are likely better ways to design, build, or wire the system. As such, we encourage future persons on the project to question why something was done the way it was. One should not hesitate to make improvements where one sees fit as long as one understands the purpose behind the initial design.

3. Stakeholder Needs

3.1 Identification of Stakeholders

The list below describes the stakeholders involved in this project:
Users: Individuals who interact with Armold via the control system.

- Creators
- Rose-Hulman students
- Rose-Hulman prospective students and family
- Rose-Hulman faculty
- Rose-Show participants

Keepers: Those responsible for Armold's long-term storage and use.

- Rose-Hulman
- Dr. Brackin and Dr. McCormack
- Rose-Hulman Health and Safety
- Rose-Hulman Maintenance Staff

Creators: Those who design, develop, and manufacture Armold.

- Armold team
- MDS professors
- Mentor professors
- Mentor students
- Legacy professors and students who continue to make improvements

3.2 Stakeholder Needs and Requirements

Since the stakeholders are grouped by category of:

1. Users,
2. Keepers, and
3. Creators,

stakeholder needs can be identified per group rather than per individual stakeholder. Each stakeholder need has associated stakeholder requirement(s). Similarly, each requirement has an associated evaluation metric(s). Some of the evaluation metrics are subjective.

3.2.1 Users

Need	Requirement(s)	Evaluation Metric
The system must be intuitive to use.	The touchscreen will provide clear options for the user on how to operate the control system.	Pass/Fail on having buttons such as *Record*, *Loop Recording*, and *Mirror Motion* can be pressed to record or loop a recorded motion, or live mirror motion.
	The control system will have labels representing which part of the arm is controlled via each sensor.	Pass/Fail on control panel labels for Elbow, Cross-body, Lateral Raise, Frontal Raise, and Pronation.

The control panel board (part with sensors, not touchscreen) UI is user-friendly and easy to understand.	The control panel must have clear labels to communicate how the controls affect Armold's motion to the user.	Dials have labels for the motion that they perform.
Armold and control panel systems can withstand semi-regular use.	The control panel must be durable enough for regular use by college-age students.	Pass/Fail on not breaking during the Rose Show demoing.
Interaction with the system must be enjoyable.	Armold must be capable of recording and performing motions.	Pass/Fail for recording. Pass/Fail for performing recorded motions.
	The control panel must be “fun” to use.	Pass/fail for general likability of the control panel’s controls.
The user can tell if the system receives their input	The UI provides some type of feedback to the user which lets them know what state it is in (waiting for input v.s. received input)	Pass/fail on implementation of this
Armold must be able to move in real-time to a user’s input on the control panel.	Armold moves relative to the input from the control system. For example, input to the sensor which controls lateral raise movement means Armold performs lateral raise movement.	Pass/fail. Test with code. < 3-second delay.
The system is safe to use.	Armold will be fully enclosed during the operation.	Pass/fail.
	Armold will have signage explaining operation and safety precautions.	Pass/fail.
	Armold will have an E-stop on the arm’s electronic system, stopping all motors from moving at once.	Pass/fail.

3.2.2 Keepers

Need	**Requirement(s)**	**Evaluation Metric**
Armold and the control system can be relocated to a permanent location for display.	The project can be moved around and the robotic arm must be small enough to keep on a table.	The total project must be able to be broken down into components that can be carried or put onto a cart.
		Armold’s bounding box dimensions (LxWxH) must fit within 34.5"x33"x32" (+/- 1” in any direction) volume when in the home

		state. Note that this does not include an enclosure.
Users must be able to access more information on Armold to understand the purpose of the display.	The purpose and explanation of Armold must be accessible when the system is used.	A document named “Armold’s Purpose” has been created and can be displayed next to his final creation.
The project must be a stand-alone display.	The system's entire code and UI must be incorporated within the project and not rely on any of the team’s laptops.	The system can be run without a laptop connection.
The robotic arm must be safe (safe such that people are not harming it, and it is not harming people) while stored.	The robotic arm must be kept in a safe, locked enclosure that can be seen through, so it is not easily accessible or damaged.	The enclosure around Armold must be within acceptable size TBD by the art curator and/or Jake Campbell and locked 24/7 unless unlocked by Rose-Hulman staff.

3.2.3 Creators

Need	**Requirement(s)**	**Evaluation Metric**
The system should be able to take input data and output a display (that does not use the physical robotic arm) such that the creators can understand and visualize the motion that Armold would theoretically perform when given these inputs.	Create some methodology to read and demonstrate position data, for example, a graphic with simple lines that represent links of the arm and move the “forearm” accordingly when that potentiometer is activated.	Pass/Fail the system has a graphics display that the creators can use to easily understand how Armold will move when given a specific input.
Armold must be anthropomorphic; his structure and motion should have a resemblance to the structure and motion of a human arm. Note that this need does not extend to realism in terms of appearance.	The code will limit the range of motion of the elbow, hand, shoulder, fingers, and cross-body movements so they will have an operating range of motion similar to that of a human and will not self-collide.	The range of motions detailed in Table 1 below is achieved.
	A person unfamiliar with the project should be able to see Armold and recognize he is an imitation of a right human arm.	Pass/fail on general recognition.
	The length of the arm from the shoulder must be short enough to clear the base of the 80/20 stand	The arm should be ≤ 28 inches.

	(when fully extended) that Armold is mounted to.	
The code will allow sensors to control motors on the arm.	The control system will consist of the following outputs per sensor type. These outputs will act as inputs to the Armold arm brain system: - Linear potentiometers: Controls the five micro servos for Armold's fingers - Rotational potentiometers: Controls the 25 Kg*cm servo for cross-body, the 40 Kg*cm servos for elbow, pronation, and lateral movement, and the stepper motor for frontal raise movement.	Pass/Fail on sensors controlling assigned motors.
The arm should be able to lift light objects for general tasks.	The arm is durable and strong enough to lift objects ≤ 1lb.	Pass/Fail on lifting 1lb.

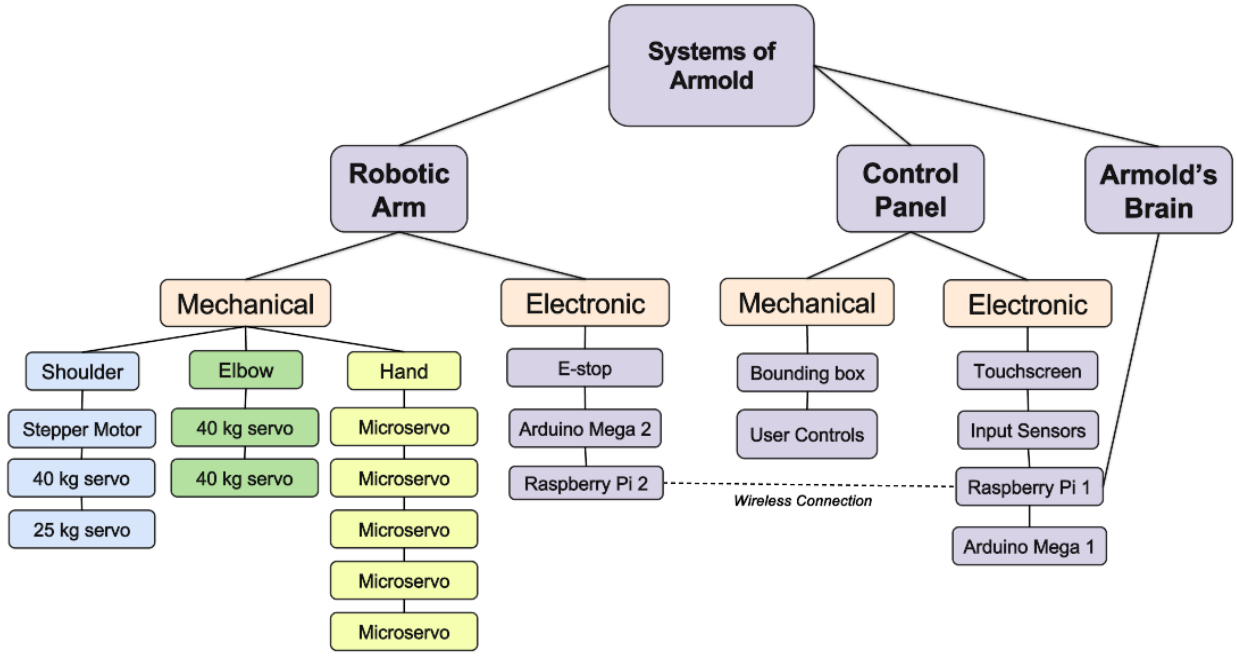
3.2.4 Functional Range of Motion Requirements

Type of motion	**Range of motion (degrees)**
Cross-body	≥ 45 +/- 5
Shoulder rotation	≥ 200 +/- 5
Shoulder lateral raise	≥ 90 +/- 5
Elbow	≥ 135 +/- 5
Wrist Pronation	≥ 180 +/- 5
Fingers	Open/closed positions

Table 1. Range of Motion of Parts of the Arm

4. System Overview

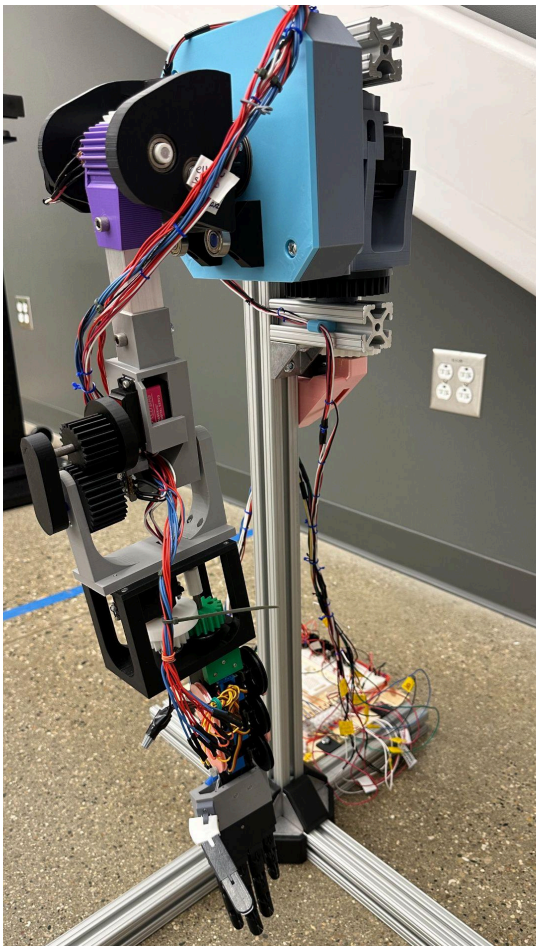
Armold is a large system that can be broken down and considered in terms of subsystems. A high-level view of this is shown in the chart below. For the most part, the team considers the robotic arm and the control panel to be separate systems, each of which has its own mechanical and electronic systems. Besides this, the third primary system is "Armold's Brain" which refers to the code on the Raspberry Pi's (both on the arm and control panel) and wireless communication method, MQTT.



The subsystems of Armold will be discussed in depth in the "Presentation of the Design" section of this report.

5. Presentation of the Design

5.1 Mechanical System



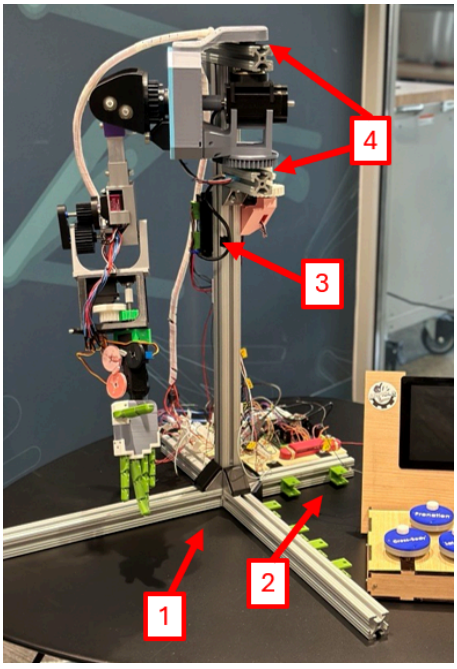
The final mechanical design of Armold contains ten actuators. These actuators allow Armold to replicate movements a human arm could make, with the exception of a saddle joint wrist. Nine of the ten actuators are servo motors and the remaining actuator is a stepper motor. The stepper motor actuator is on the shoulder. It allows for frontal raise motion. The nine servo actuators are distributed as follows:

- One cross-body actuator
- One lateral raise actuator
- One elbow actuator
- One pronation actuator
- Five finger actuators

The motors all have wires which run up the arm then down the spine to the base of the arm. They are twisted together and wrapped around the arm to avoid getting pinched while the arm is moving.

The current state of the design (May ‘24) is pictured to the left. This section of the report discusses the design of each joint in depth and details how the actuators are used to accomplish the necessary range of motion. Note that the team designed all parts of Armold. A consequence of this is that his assembly required many (all) custom parts. The majority of Armold’s parts were 3D-printed of PLA filament in our Rose-Hulman shops, including structural parts and gears. Additive manufacturing was used to keep costs down and to further show the possibilities of 3D printing to students. Unfortunately, there is a tradeoff in terms of durability.

5.1.1 Armold’s Spine



5.1.1.1 Spine Overview

Armold is mounted on a stand made of Aluminum 80/20 and connectors which we call the “spine”. The spine has longer legs on some sides to prevent tipping when Armold is in use. The positioning of the longer legs allows Armold to be a right arm. The shape of the 80/20 is convenient for mounting things. Our team designed and 3D-printed various mounting parts. Some other notable things about the spine (as corresponding to the image on the left):

1. Parts which make up the spine; 80/20 base and vertical pieces
2. Connectors that allow mounting of ¾” thick (or other thicknesses) wooden boards to hold Armold’s electronics
3. Mounting of the stepper motor driver
4. Mounts with bearings to support and constrain the entire moment of the arm from top and bottom (the stepper motor)
5. Note- the spine is held together by 90 degrees 80/20 1.5 in width connectors. There are 9 total connectors.

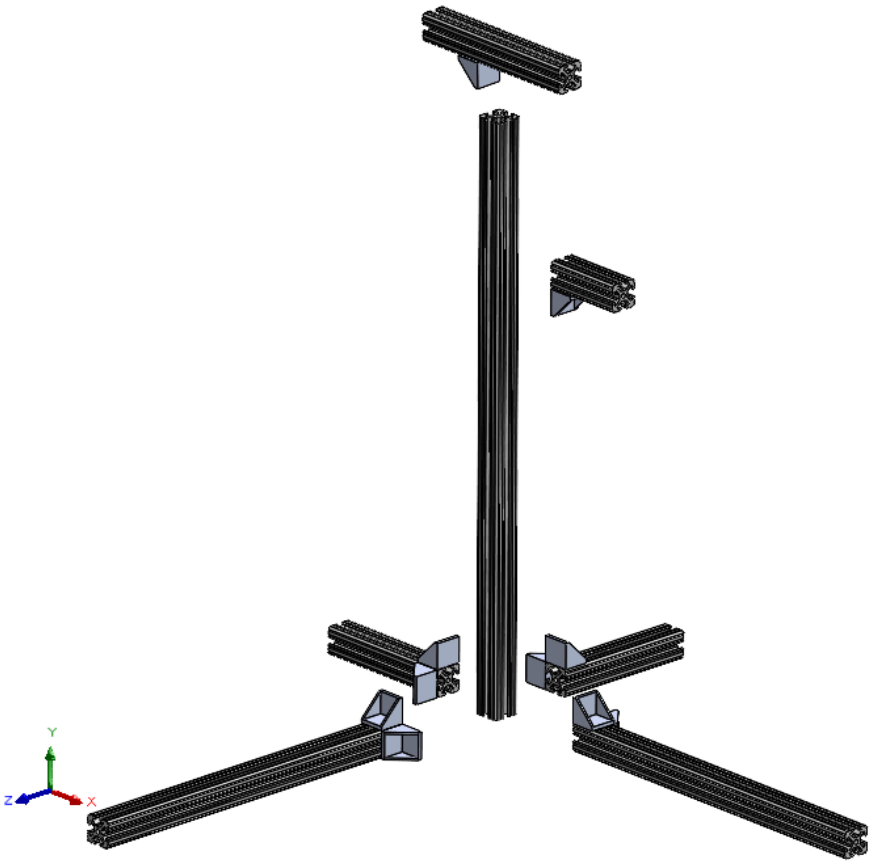
5.1.1.2 Spine Assembly Process

The spine is composed of six 80/20 pieces and nine 90-degree connectors. Each connector holds two 80/20 parts in place with a slot and screw component. A 5mm Allen wrench is used to loosen and tighten the screws. The table below is a BOM showing all the lengths of 80/20 stock used, where they are used, and what they are used for.

Quantity	Cut Length (in)	Location/Description	Purpose
1	32	Upright Spine	This piece creates height to get Armold away from the ground to mimic a human arm mounted to a torso.
2	20	Long Base Legs	The long base legs prevent Armold from tipping when reaching forward and outward.
2	8	Short Base Legs	The short base legs are for stability.
1	4.5	Shoulder Lower Support	This piece holds the entire arm assembly upright on a large ball bearing. Supports the downward force of the arm.
1	10	Shoulder Upper Support	This piece connects to the top of the arm and supports a majority of the moment caused by the arm's weight away from the point of contact to the shoulder lower support.

Spine Assembly:

The spine assembly is fairly self-explanatory. Something that may be notable and not clear in the image to the right is that the upright spine (the longest piece) goes all the way to the ground/table. It does not rest on top of any base pieces. The base pieces also rest on the ground, and each is mounted to a face of the upright spine.



Another couple important notes is that firstly, the shoulder lower and upper supports must be parallel to one another. Note that while the upper shoulder support is constrained as it is connected to the top of the upright spine, the position of the lower shoulder support piece is not dependent on the spine but rather the stepper motor. It will “find its own height” when it is attached to the arm. Secondly, it is necessary that the base legs which form the quadrant of space that Armold tends to operate in are longer such that Armold does not tip over while in use.This orientation is also critical for Armold to be a right-armed robot (sorry lefties!).

5.1.1.3 Spine Maintenance

Throughout our design iterations and wiring prototypes, the spine has experienced moments of clutter consisting of wires, 3D printed wire holder supports, tape, and other miscellaneous things. At the end of the lifespan of the project, any vestigial remnants of things on the spine were removed to create more room and improve cleanliness.

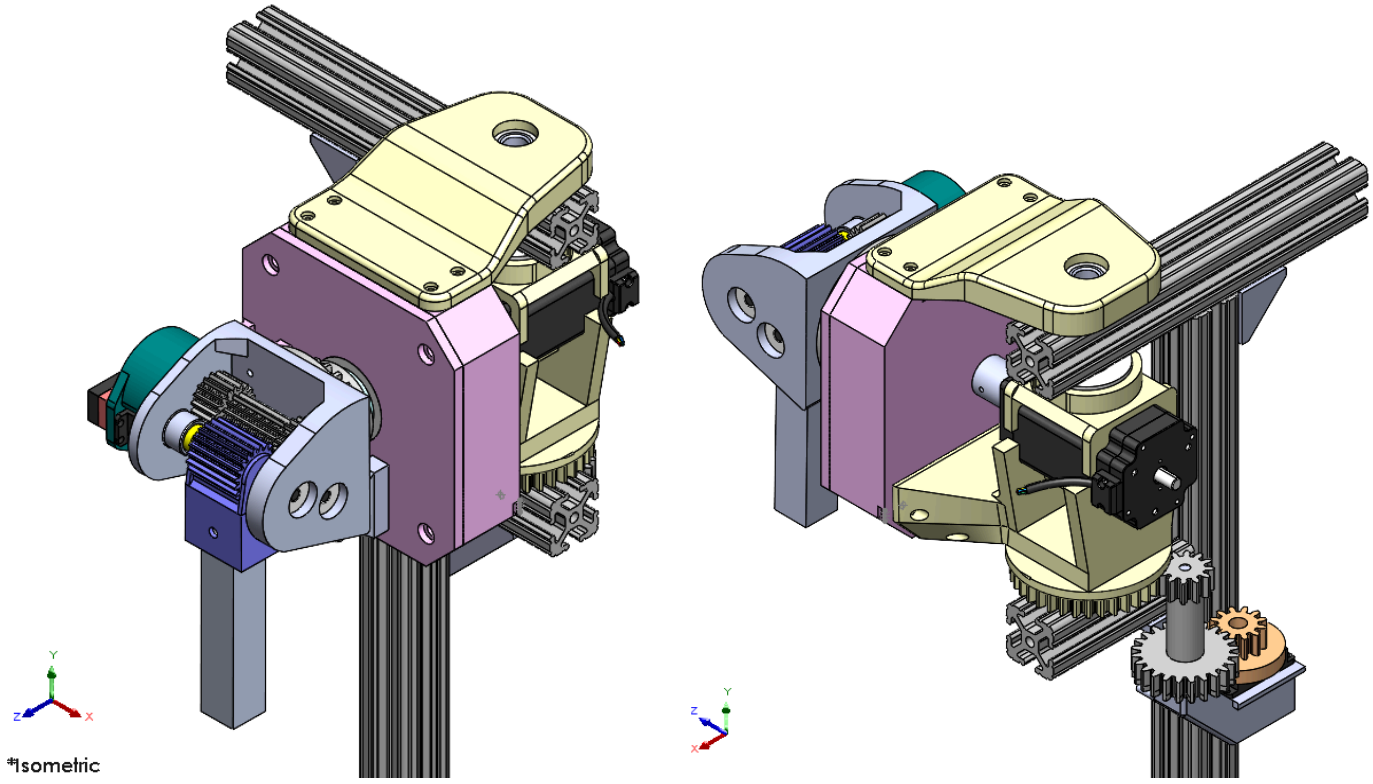
5.1.1.4 Spine Suggestions, Notes, Improvement Opportunities

During our static calculations to determine how long the legs need to be to prevent tipping, we did not take the weight of the legs into account. The additional mass acted as a factor of safety as the mass of the legs would move the center of mass of the arm closer to the upright spine. The effect of this additional mass has become even more dramatic since we have attached boards with wiring and power supplies to the base. Therefore Armold's legs do not necessarily need to be as long as they are if the weight of the spine and attachments to the base themselves are taken into consideration. This could decrease the ground footprint Armold needs to be stationary in the future.

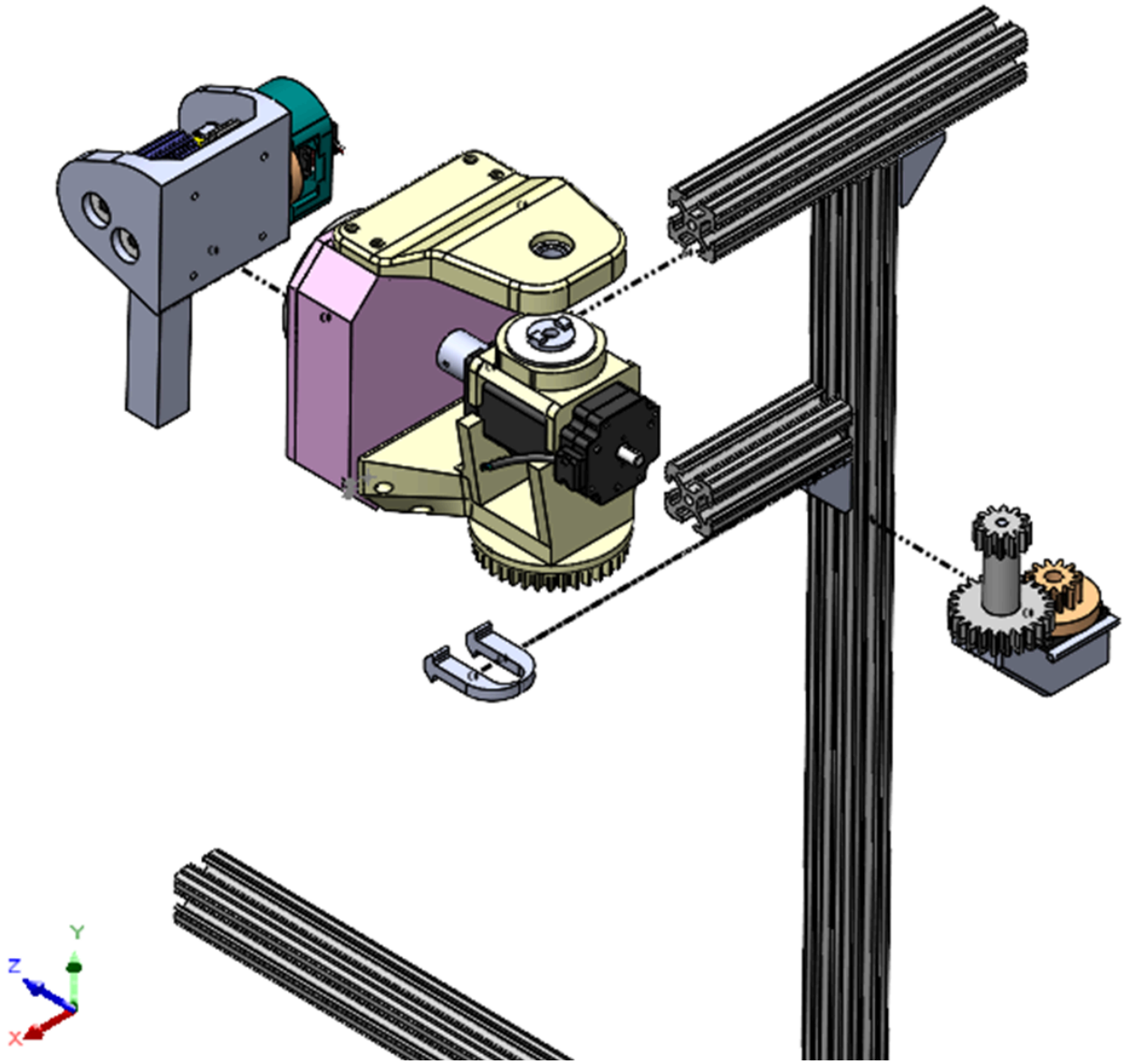
Another recommendation is to incorporate some kind of handle or transportation mechanism. A handle is not required once Armold becomes a successful display, but we see that being a long time away. Portability during these prototyping stages is crucial, and he has become less and less portable as the year has progressed. Finding a way to lighten the load or manage the bulkiness of Armold would make transport time more pleasant and efficient.

5.1.2 Armold's Shoulder

Armold's shoulder contains 3 actuators. These actuators allow for humanoid motions such as frontal and lateral raises, as well as cross-body reaching. The shoulder is also the connection point between the arm and the stand, and as such, this section will likely include pieces of the stand a bit as well which is why the stand assembly was introduced first. In this section, the shoulder will first be introduced as a full assembly with several subassemblies. From there, the subassemblies pertaining to each actuator will be documented and described. The complete shoulder assembly is pictured below.

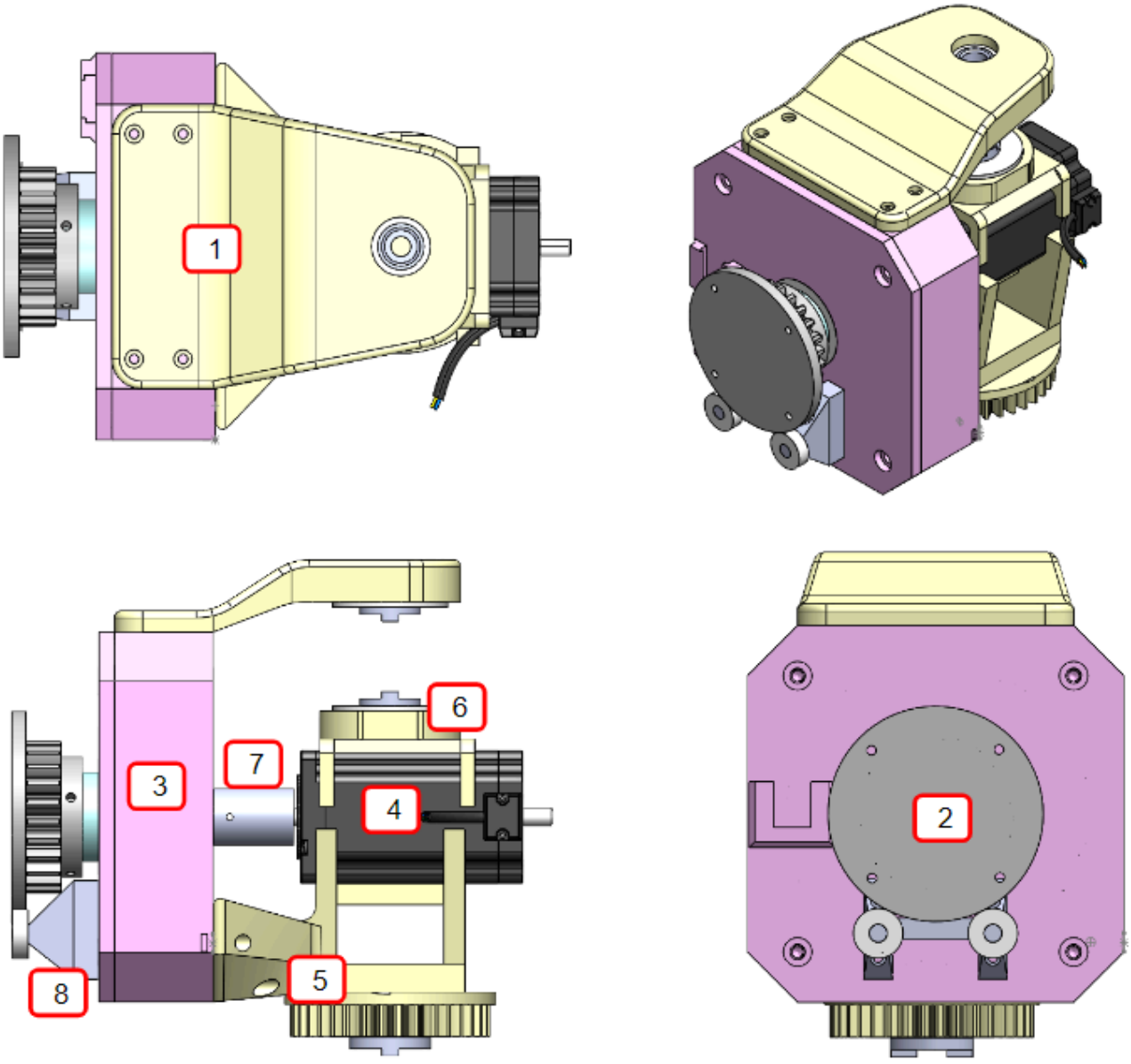


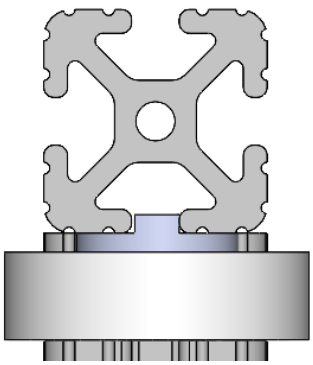
The image below depicts how the following subassemblies fit together: frontal raise actuator, lateral raise actuator, cross-body actuator, and the spine. The two parts that look like half of a “U” are clips to constrain the cross-body actuator.



5.1.2.1 Frontal Raise Actuator

The frontal raise actuator is the foundation of the shoulder. It holds the stepper motor and a large planetary gearbox to support the high moment when the arm is extended and holding up to one pound. It is also large because it must mount the arm to the stand securely. The three yellow parts house the bearings that connect the arm to the spine. Below is a numbered list of the major parts depicted and their purpose for the frontal raise shoulder assembly.





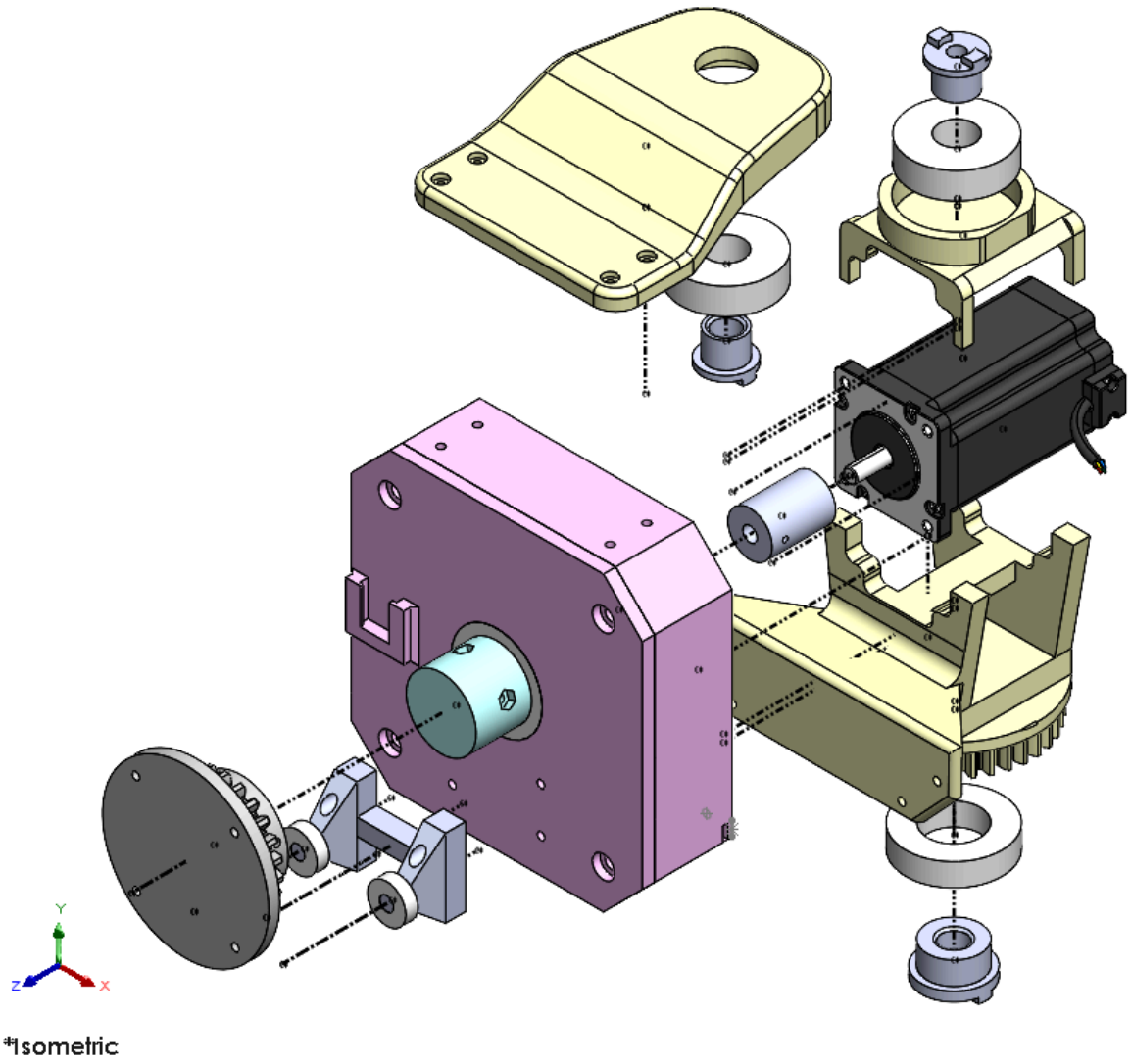
The bearings' inner diameter is a tight fit to the 3D-printed part that slides in and secures to 80/20. The same hardware of a metal part that slides into the 80/20 and connects to a screw to tighten the 90-degree supports for the spine down is used to secure the bearing to the stand. This allows for the arm to be removed from the spine easily. As you can see from the right-side view above, all of the slides into the 80/20 are parallel so that the shoulder assembly can slide in and out of the spine's lower and upper shoulder supports.

Number	Part	Description
1	Top Arm Support	This piece supports the moment caused by the arm's weight.
2	Lateral Base Gear	This piece is the bridge between the frontal raise actuator and the lateral raise actuator. The reason for its gear design is for the future potential to have a feedback sensor off to the side.
3	Frontal Raise Gearbox	This is the casing and inner gear for the planetary gearbox that supports the moment caused by frontal raise motion, driven by the stepper motor.
4	Stepper Motor	Drives frontal raise actuator.
5	Base Bearing and Motor Saddle	A multi-use part that holds the stepper motor in place, connects the lower shoulder to the spine, supports the frontal raise gearbox, and its gear connects to the cross-body actuator.
6	Upper Motor Saddle	Holds the motor in place and connects to the spine.
7	Motor Couple	Connects stepper motor shaft to gearbox's first stage sun gear.
8	Frontal Support Bracket	Supports the moment act of the lateral base gear. Before this piece was implemented there was pinching happening inside the gearbox.

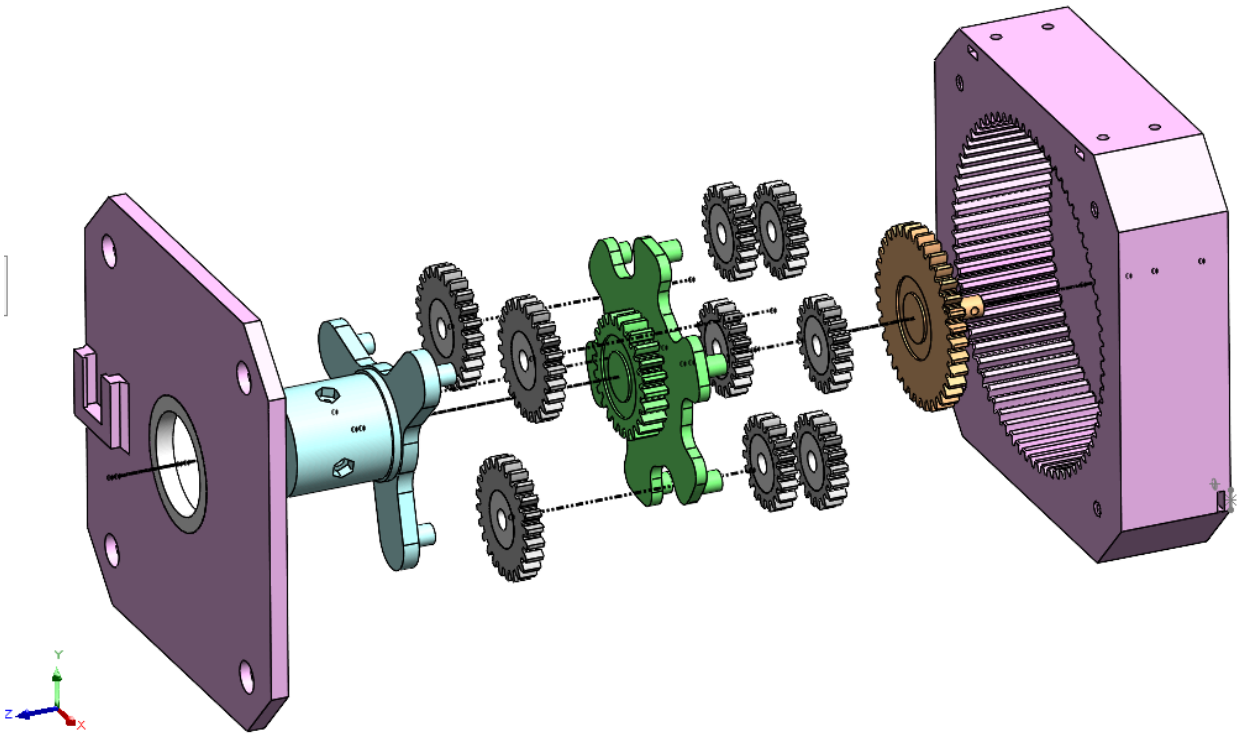
Many more parts will be discussed in the next section.

5.1.2.1.1 Frontal Raise Actuator Assembly

We will split up the front raise actuator assembly into two segments; the gearbox, and everything else. To begin, below is the overall assembly of the frontal raise actuator in an exploded view. In addition to the numbered parts above, there are some bearings and bearing-to-spine connectors. Screws used to assemble these parts are not shown in our Solidworks assemblies as the files already had a lot of parts and mates. The four screws used to connect the gearbox lid to the gearbox are ¼-20 and the screws used for the rest of the assembly used various M3 screws and 6-32 x ¾" screws as well.



The gearbox contains a two-stage planetary gear system. This system is pictured below in an exploded view. Our newer models of the shoulder do not contain these internals for simplicity, so the casing is an older part file, however, the internals are the same. The first stage begins with a sun gear (orange) attached via a small rod to the motor couple. The first sun gear's stage has a gear ratio of 3, driving the middle crank (green) to the next state with a gear ratio of 4 for a total gear ratio of 12. Each planet gear has a small ball bearing in the center to greatly reduce friction within this 3D-printed gearbox. Both stages of the planetary gearbox ring gear are stationary and inside the gearbox casing. This design was inspired by the internal gearbox of a handheld power drill.



5.1.2.1.2 Frontal Raise Actuator Maintenance

The frontal raise actuator is a rigid structure, with only a few maintenance points to mention. Concern comes up from time to time with the internals of the gearbox as it is a high torque gearbox with 3D printed gears and the inside cannot be seen easily. From experience, we recommend removing the lateral base gear (which would require removing the arm from the lateral raise actuator when fully assembled) and then removing the 4 ¼-20 bolts holding the cover on. This process takes about 3-5 minutes. Another noteworthy maintenance tip is that the stepper motor can be tested without being attached to the arm simply by removing the pin that connects the motor coupled to the first stage sun gear. This helps troubleshoot wiring and coding issues or redesigns.

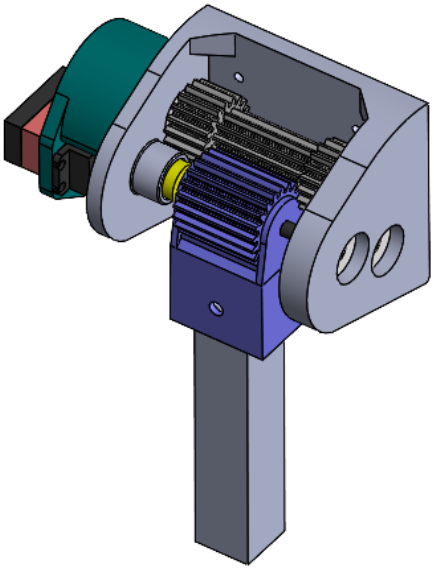
5.1.2.1.3 Frontal Raise Actuator Improvement Opportunities

There are a lot of opportunities our team saw in this actuator that we were unable to get to, including:

- The slim bearing between the gearbox lid and output crank tends to slowly walk out of place. Redesigning the lid to have a front covering the bearing would remove this issue.
- This actuator is extremely bulky due to the planetary gearbox. Planetary gearboxes are known for being compact, but not when they are 3D printed. The large teeth and necessary bearings added a lot of size to the shoulder we were not expecting. We were considering maybe having a worm gearbox for the future but there is no back drive which would be unideal. Therefore the torque required could be reassessed since the motor on the final prototype is twice as strong as the motor we had at the time of this actuator's design.
- Due to the room for a feedback sensor on the lid, the output crank sticks out far and the lateral raise actuator is about 2 inches away from the gearbox lid. This distance adds bulk and increases the moment the shoulder needs to support.

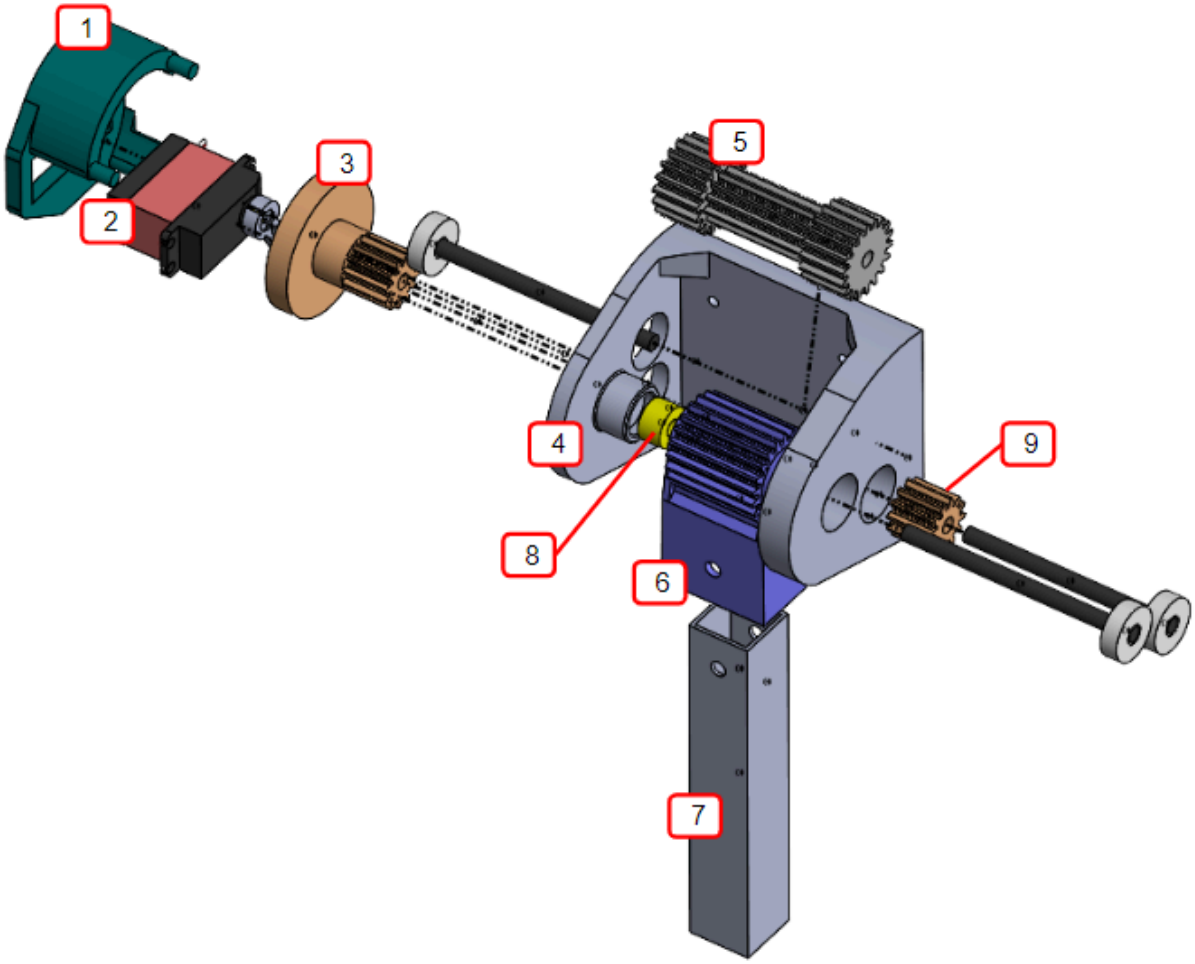
5.1.2.2 Lateral Raise Actuator

The lateral raise actuator allows for motion laterally from the shoulder. It also connects the shoulder to the elbow via a 1-inch square stock. Four screws are used to connect the main enclosure of the lateral raise actuator to the lateral base gear previously shown on the frontal raise subassembly.



5.1.2.2.1 Lateral Raise Actuator Assembly

The exploded view with significant parts labeled is pictured below. Each numbered part has a name and description in the table following the exploded view. The motor and motor housing were designed for easy disassembly in the case that maintenance is required. This system gets assembled mostly horizontally. The only requirement is to first place screws into the top through holes because they will not fit in after the long combination gear is in place. The long combination gear is the first gear to put in place as it is difficult to slide into place after the others. The gears have to be hovered in the middle of the casing and the rods being pushed in from the sides will go through the gears to mount them in place. Each ball bearing has a small adaptor piece to mount the shafts to the bearings that are not shown in the CAD model. There is one shown for reference later on in the cross-body actuator assembly.



Number	Part	Description
1	Motor Housing	Holds the servo motor in a fixed position concerning casing.
2	40 Kg*cm Servo Motor	Drives actuator
3	Servo Gear	Connects servo motor to gearbox
4	Lateral Raise Casing	Contains the majority of the actuator
5	Long Combination Gear	Acts as the idler gear and connects two gear stages
6	Output Gear	Connects to the upper arm output
7	Upper Arm	1 in square stock acts as Armold's "bone" to connect shoulder to elbow
8	Custom Bushing	Machined in place of a bearing for the tight space
9	Detached Servo Gear	Same gear as on the servo but added to the other side to help make the gearbox more uniform.

5.1.2.2.2 Lateral Raise Actuator Maintenance

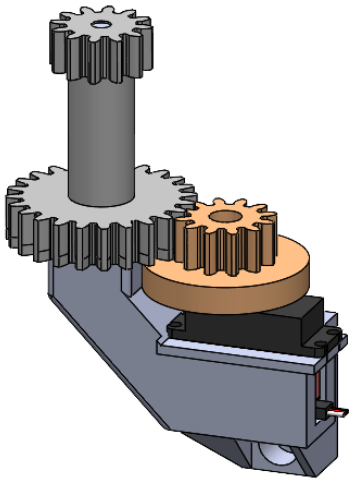
The motor and motor housing easily detach from the casing for electrical and range of motion testing. The rest of the gearbox is a hassle to disassemble, however, it has proven to be reliable in the past. One issue we have had is the output gear has cracked near the area where it attaches to the stock in the past, however a quick redesign with thicker walls seemed to fix the problem. It is still something to be wary of.

5.1.2.2.3 Lateral Raise Actuator Improvement Opportunities

The servo motor and motor housing fall out of the assembly constantly. This was one of Armold's largest flaws. Currently, we have a 3D-printed C-clamp holding it in place. This was a temporary solution which held things together and allowed us to test the actuator. However, it proved not to be sturdy enough and will slide out of place, especially when the front raise actuator is in use. Our idea of an ideal solution would be to redesign the casing such that it uses screws to attach the motor housing to the casing. The tricky part of that design would be designing for assembly with the gears being in place while still being able to access screws.

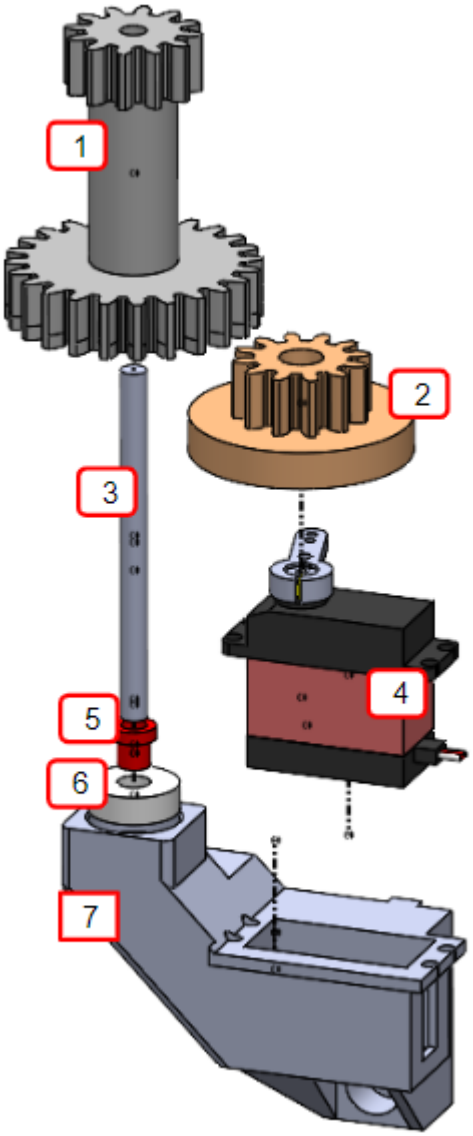
Another problem we have with the lateral raise actuator is the amount of play. The play in the gearbox, specifically the output gear and the combination gear makes it so the arm easily wobbles a significant amount. We do not have any great ideas to fix this while still using 3D-printed gears but it is an obvious weak point in the arm.

5.1.2.3 Cross-Body Actuator



The cross-body actuator is perhaps the least human-like on the arm and has the smallest responsibility. However, if done correctly in the future, the cross-body actuator can play a key part in humanoid movements. This actuator is located on the upright spine and is not mounted to the shoulder at all. It utilized two small hooks that slide into the lower shoulder support 80/20 piece on the spine to hold the tall gray gear upright. The tall gray gear is in contact with the base of the frontal raise actuator which allows this servo to rotate the whole arm about the upright axis.

5.1.2.3.1 Cross-Body Actuator Assembly



Each part of the cross-body actuator is listed in the table below with the rows corresponding to the number callout in the exploded view to the left. This assembly is assembled completely top down which allows for quick assembly and disassembly. This is very helpful when it comes to maintenance.

Number	Part	Description
1	Idler Gear	Connects input gear to output gear. This output gear is on the base part of the frontal raise actuator.
2	Input Gear	Connects to the servo motor
3	Shaft	¼ shaft of rotation
4	25kg Servo Motor	Drives the actuator
5	Adapter	Connects ¼ inch shaft to 8mm ID ball bearing
6	Ball Bearing	Reduce rotational friction
7	Cross-Body Actuator Housing	Holds it all together and mounts to the upright spine.

5.1.2.3.2 Cross-Body Actuator Maintenance

Maintenance for this actuator is simple. Unscrewing the one screw holding this onto the upright spine results in this actuator being completely detached from the system. This allows for easy home position corrections and testing. We have also needed to reprint some of the hooks holding the idler gear in place due to them snapping in the past.

5.1.2.3.3 Cross-Body Actuator Improvement Opportunities

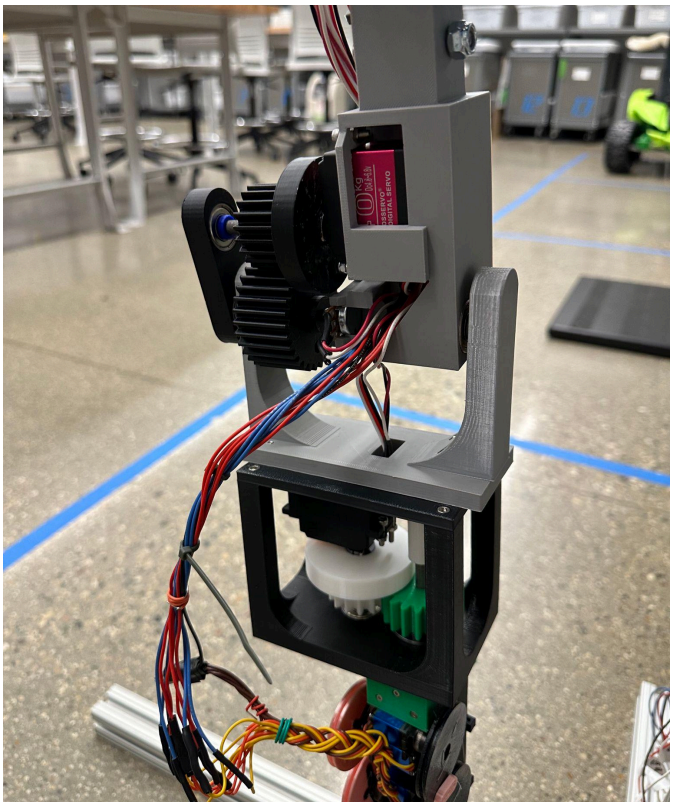
Currently, the cross-body motor has gear misalignment issues due to the top of the idler gear not being properly mounted. Redesigning this movement to be better constrained could be more energy efficient for the motors and cause more precise movements. We also believe a stronger servo may be necessary as the actuator's movements are inconsistent.

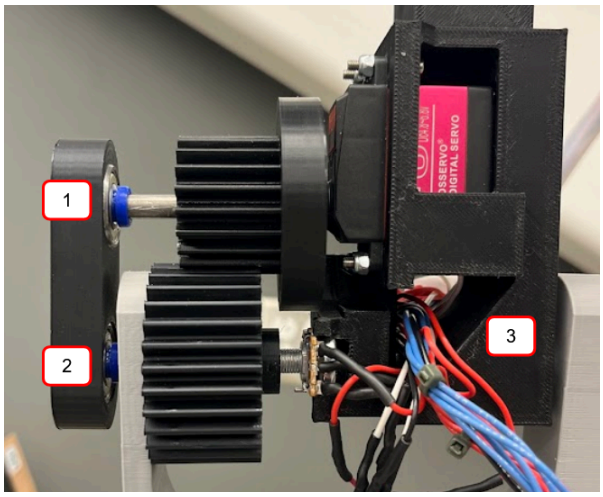
Another improvement which could be made is redesigning the hooks which constrain the tall gray gear such that they are stronger and/or utilize a bearing to reduce friction.

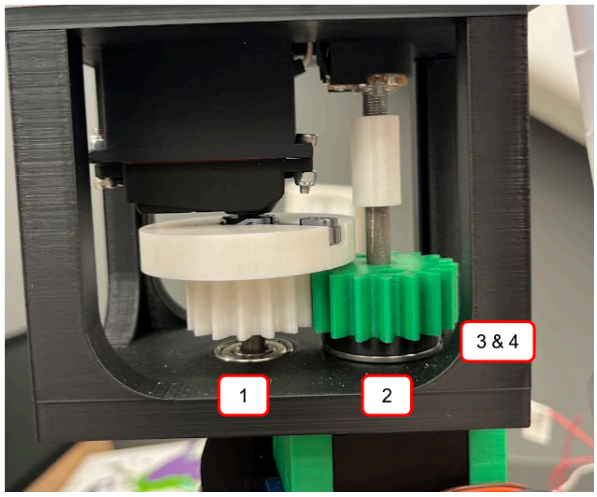
5.1.3 Armold's Elbow and Pronation

5.1.3.1 Elbow and Pronation Overview

Armold's elbow and pronation system is made out of 3D printed parts, two 40 Kg*cm servos, and two 10kΩ potentiometers. The elbow assembly is connected to the shoulder with a piece of 1"x1" square aluminum tubing and is secured with a ¼" bolt and a locknut. It uses a gear ratio of 12:18 to create more torque when lifting the arm. The pronation assembly does the same thing with a gear ratio of 15:18. Servos are held into place with M3 screws and locknuts in both assemblies to prevent transverse vibration from loosening them over time. ¼" stock and adapters for bearings are used as shafts to ensure uniform rotation. Ball bearings and roller bearings are used in various locations in the assemblies, which can be seen in the images below. In the elbow assembly, shown on the left image below, ball bearings 1 and 2 keep the gears in the elbow aligned while allowing them to rotate along their axes. The third ball bearing on the elbow is inside the black 3D-printed servo holder. This allows the elbow to bend (rotate) without friction. In the pronation assembly, shown on the right image below, bearings 1 and 2 are ball bearings, allowing the pronation gear shafts to stay aligned and rotate without friction. Needle bearings 3 and 4 fit between the pronation output to prevent friction while maintaining a compressed fit.

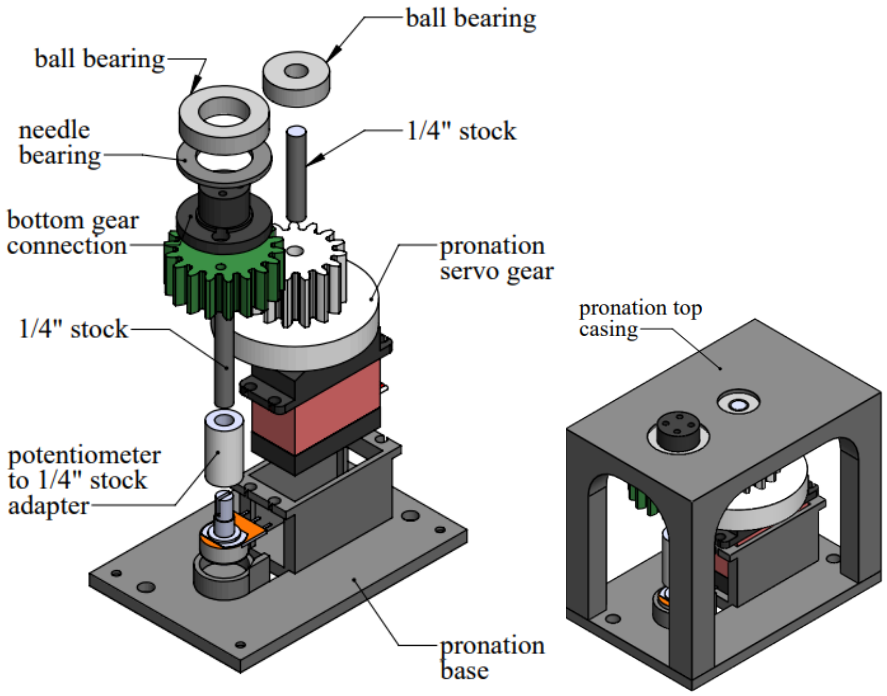






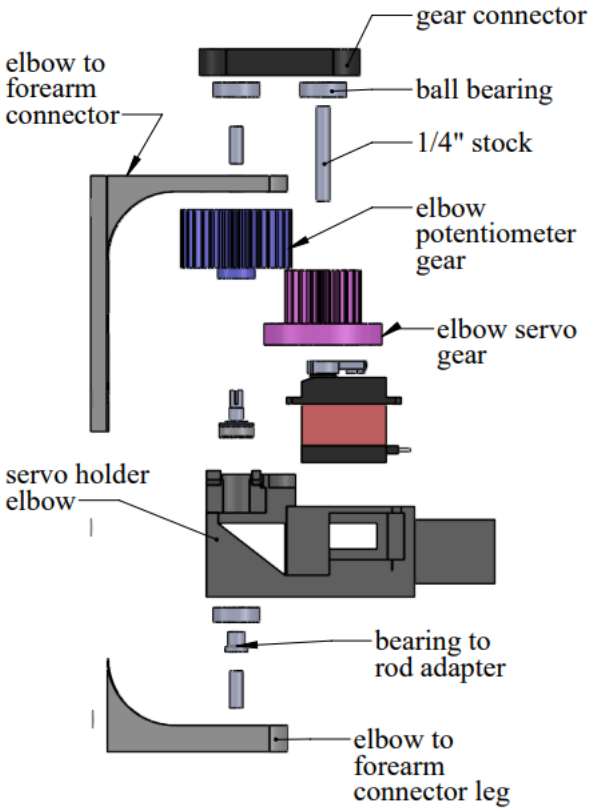
5.1.3.2 Elbow and Pronation Assembly Process

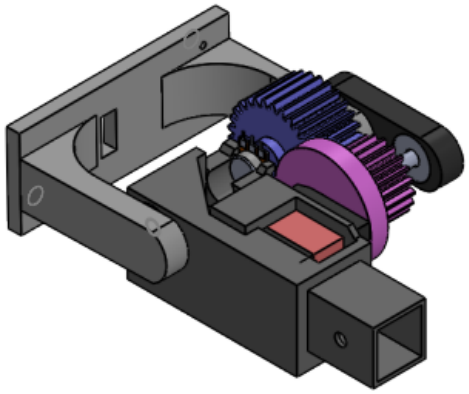
Because this large midsection of the arm can be broken into 2 major assemblies, they will first be described separately in their assembly processes and then brought together in a third set of instructions.



Pronation Assembly: (Images Above)

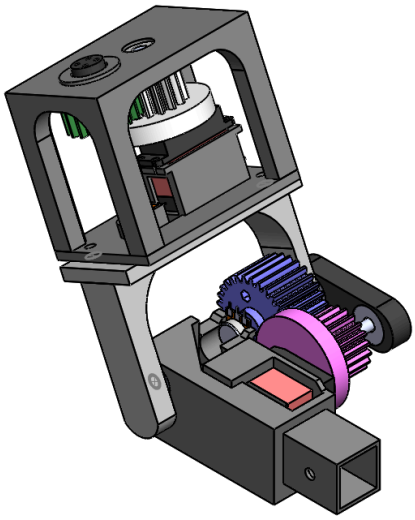
1. Begin by placing the servo motor in the pronation base part and thread the wires through the hole located near the center of the part.
2. Secure the servo motor in place with four M3 screws and four locknuts.
3. Push the pronation servo gear onto the servo motor arm and secure it with an M3 screw through the center of the gear.
4. Press-fit a 10kΩ potentiometer into the pronation base part and thread the wires through the same hole as the servo motor wires.
5. Connect a 1.6” piece of ¼” stock to the potentiometer to ¼” stock adapter part.
6. Connect a 1.375” piece of ¼” stock to the pronation servo gear.
7. Press the other end of the adapter onto the knob of the potentiometer.
8. Screw the bottom gear connection part onto the pronation potentiometer gear with M3 screws.
9. Press the green gear into the exposed end of the ¼” stock.
10. Push the two ball bearings into the holes on the pronation top casing part.
11. Place one needle bearing onto the bottom gear connection part.
12. Gently place and possibly superglue four M3 nuts into the holes in the legs of the pronation top casing part.
13. Place the pronation top casing part over the gears, lining up the ¼” stock and bottom gear connection part with the bearing holes.
14. Screw four M3 screws through the base of the pronation base part into the legs of the pronation top casing part.
15. Place a rod to bearing adapter part on the exposed ends of the ¼” stock coming from the pronation servo gear to ensure it has a snug fit with the inside diameter of the ball bearing.





Elbow Assembly: (Images Above)

1. Begin by press-fitting two ball bearings into the gear connector part.
2. Press two rod to bearing adapters into the inside diameter of the ball bearings. Set this sub-assembly aside.
3. Press another ball bearing into the side of the servo holder elbow part. Then press a rod to bearing adapter into that bearing.
4. Place a servo in the servo holder elbow part, threading the wires through the triangular hole, and secure it into place with four M3 screws and locknuts.
5. Push the elbow servo gear onto the servo motor arm and secure it with an M3 screw through the center of the gear.
6. Press-fit a 10kΩ potentiometer into the servo holder elbow part and thread the wires through the same hole as the servo motor wires.
7. Place nuts inside the hex-shaped holes on the elbow potentiometer gear. Press the elbow potentiometer gear onto the end of the potentiometer, lining up the gear teeth with the elbow servo gear
8. Line up the holes on the elbow to forearm part with the holes on the elbow potentiometer gear and connect them with M3 screws.
9. Place a 0.7-inch piece of ¼” stock in the side of the servo holder elbow with the bearing and rod to bearing adapter.
10. Attach the forearm to elbow connector leg to the piece of stock and screw it into the forearm to elbow connector with M3 screws.
11. Place a 0.7-inch piece of ¼” stock through the elbow potentiometer gear and a 1.8-inch piece of ¼” stock through the elbow servo gear.
12. Attach the gear connector sub-assembly to the end of the ¼” stock.



Bringing Assemblies Together:

1. Connect the forearm to the elbow connector to the pronation base part with two screws.

5.1.3.3 Elbow and Pronation Maintenance

After prolonged use of Armold, it is recommended to check the lock nut positions. The locknuts ensure the servo motors stay in place and do not loosen over time due to vibration in the assembly. Lock nuts should rarely need tightened, most likely due to the erosion or deformation of 3D prints.

It is also recommended to perform regular checks on 3D print stability. Look for cracks, layer separation, or other breaks in prints. As PLA material can weaken from prolonged stress or temperature changes, it is important to ensure there are no defects.

Bearings may need to be cleaned and regreased if the rotation of either assembly seems to need more power than usual. It is good practice to check these and regrease them during regular maintenance on the arm.

5.1.3.4 Elbow Suggestions, Notes, Improvements Opportunities

There are a few recommended improvements for the midsection of the arm. The first major improvement would be replacing our motors with stronger ones. Motors that are strong enough to rotate or lift sections of the arm like the elbow or pronation would create a much simpler design, reducing the bulkiness of the elbow and pronation assemblies. This would also reduce the chance of gear misalignment, additional friction from PLA parts, and mechanical failures. Our team believes that something as strong as 80 Kg*cm servos would drastically increase the performance of the system. However, it is important to note that power must be considered when changing motors. Amperage was one of the largest challenges of this project and it should be taken into mind by future project team members.

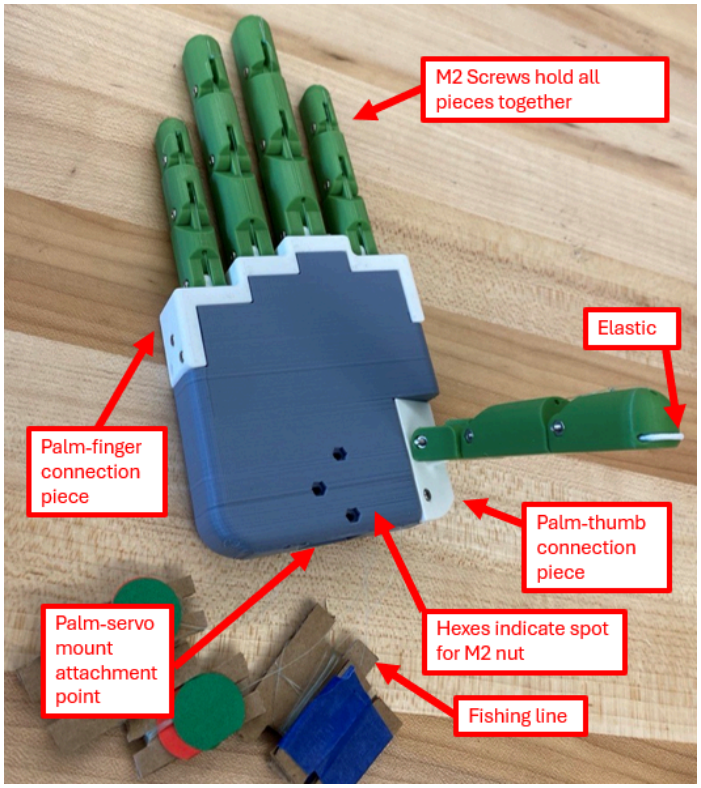
Another improvement could be replacing the potentiometers in this system with rotary encoders. Potentiometers were used in hopes of implementing feedback control, however, when tested, they proved to be jittery and unreliable for consistent values. Using rotary encoders would provide more accurate data on the real position of the arm. In this model, potentiometers were used as an axis of rotation. Unfortunately, this can damage them over time and prove to be mechanically unstable. In future designs, if gears are still used, we would recommend running a separate gear off of the system that does not aid in the lifting or rotating power.

5.1.4 Armold's Hand

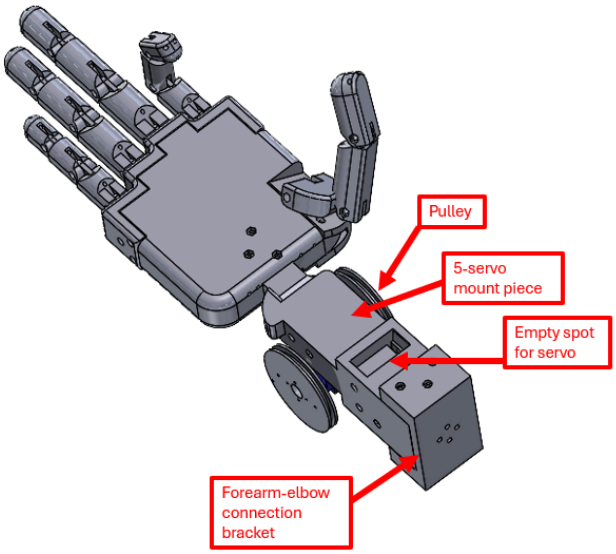
5.1.4.1 Hand Overview

Armold's hand system is made out of 3D-printed parts, 5 micro servos, and M2 screws and nuts (lengths 6mm-20mm).

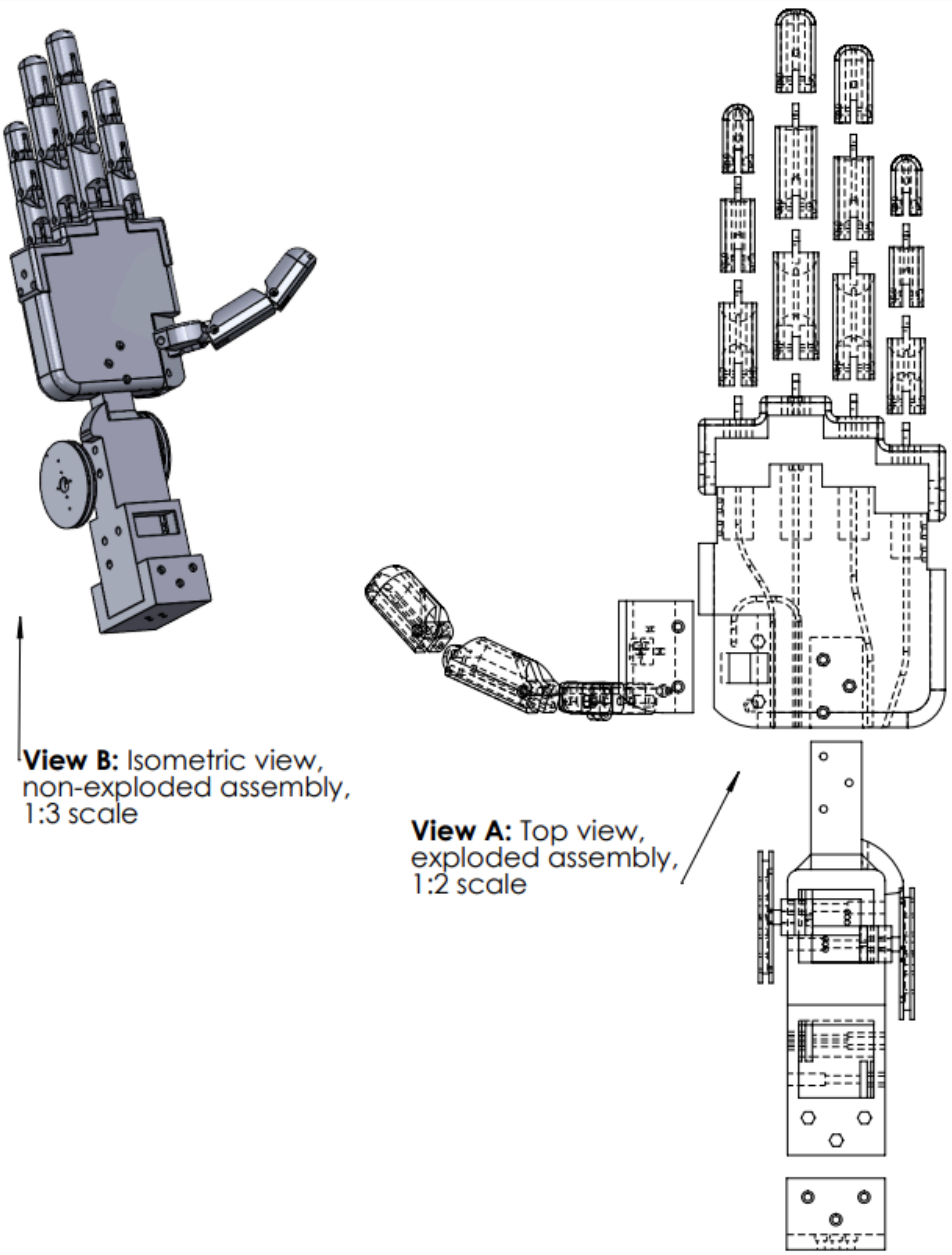
The fingers of the hand can open and close (and achieve positions in between). They are moved via the pulling of a fishing line which is tied to the top of each finger and runs through the front of the fingers. The fishing line runs through the fingers and palm (these string channels can be seen more clearly in View A of the engineering drawing below) before it wraps around a pulley that is connected to a micro servo arm. When the string is pulled, the finger achieves the close position. When the string is released, the finger returns to its "home" (open) position since there is elastic that runs down the back of the finger which acts like a spring.



The picture to the left demonstrates what the hand assembly looks like including the forearm pieces. Note that only two of five pulleys are included in this image. This image shows the proper orientation of parts for assembly.



5.1.4.2 Hand Assembly Process



Above, a screenshot of the engineering drawing for the hand assembly is shown. The actual engineering drawing can be found in the team's GitHub repository. There is no certain order to assemble the hand, so the process below will detail the best way we found to assemble the hand. The entire hand assembly uses M2 screws of various lengths. It is highly recommended to work with a full set of lengths 6mm - 20mm near you. Before attaching any part, a screw can be dropped into the countersunk side to determine the appropriate screw length to use. It may be helpful to reference View A above throughout assembly.

1. Assemble the fingers independently of one another. Each finger contains 3 phalange pieces. All phalanges are connected using M2 screws. The shortest screw which could still fully thread through the nut was used. Hexagon countersinks are for the nuts, while round countersinks are for the screw heads. The naming convention used:
    - Fingers are referenced with the first letter of their names (T = thumb, I = index, M = middle, R = ring, and P = pinky)

- Phalanges are referenced depending on how far out they are from the palm. “Base” refers to the piece closest to the thumb, “middle” refers to the middle piece, and “tip” refers to the fingertip

At the end of this step, all five fingers should be assembled but not connected to anything else.

2. Attach fingers to their finger-palm connection pieces.

   2.1 Attach the index through pinky fingers to the finger-palm connection piece by screwing them on. Note that this can be done in any order but it is easiest to connect the middle finger, then the ring finger, then the pinky finger. The index finger can be attached at any point since it screws in from the opposite side.

   2.2 Attach the thumb to the thumb-palm connection piece by screwing it on.

3. It is time to set up the elastic in the fingers. Cut five pieces of elastic. They should be approximately the length of the finger doubled, plus a bit of extra. Starting on one side at the bottom of the finger, run the elastic through the finger connection piece up until it comes out at the top of the finger. I found success in doing this by holding the finger straight, but digits can be bent as necessary if the elastic needs to be re-aligned to get strung through. Once the elastic comes out of the fingertip, bend it back the other way in a U-shape and run it down the other side of the finger. When you have both ends at the bottom under the finger-connection piece, tie the elastic in a double knot. You want it quite taught, but not too much. There is a bit more art than science to this process; iterate between tying the elastic and bending the finger until the finger snaps back at an acceptable rate. When this is done and the elastic is either double or triple-knotted, cut the elastic so the ends are ~1/4 ” or so. The palm piece has hollow cut-outs to accommodate the elastic. Put either end of the elastic together and wrap with tape. This will make the next assembly step easier.
4. Attach the finger-connection pieces to the palm piece.

   4.1 There are spots for nuts on the palm piece. If the print is tight enough, the nuts will stay in on their own. If not, superglue the nuts in. Alternatively, place the nuts in and then put a piece of masking tape over the entire face. This will still enable the screws to poke through and thread with the nuts.

   4.2 Attach the finger-connection piece which contains the index-pinky fingers to the palm piece. Screw from either side evenly to ensure all nuts catch (there is a bit too much space between the palm and finger-connection pieces… not a problem, but assemble accordingly).

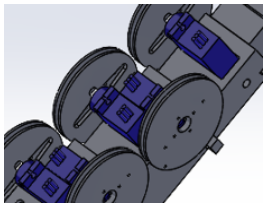
   4.3 Attach the thumb-palm connection piece to the palm piece.

5. It is time to run the fishing line through the fingers. Cut five pieces of line that are 15” minimum. For each finger:

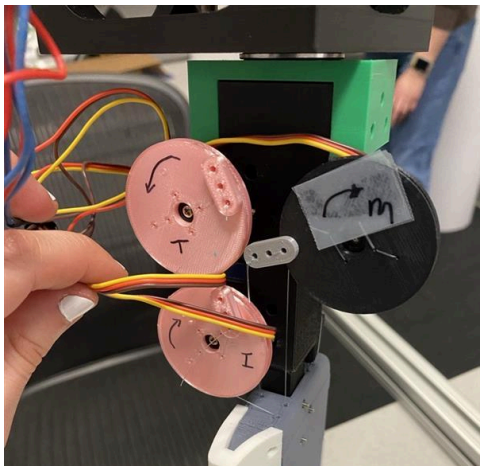
   5.1 Starting at the tip, run the line through the finger, finger-connection piece, and palm piece. Tie the line at the tip of the finger (see Armold for reference),

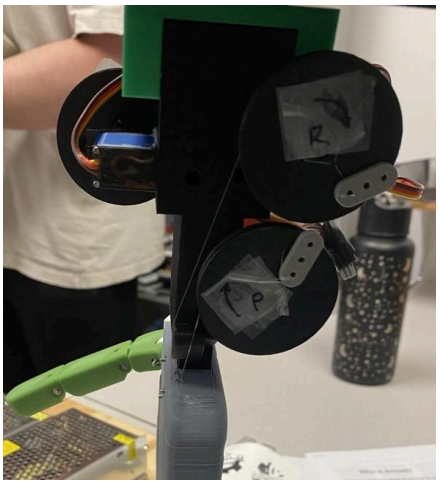
   5.2 It is recommended to manage the lines which now protrude from the palm piece. Wrapping them around something small works, or they can be taped higher on the palm piece for now.

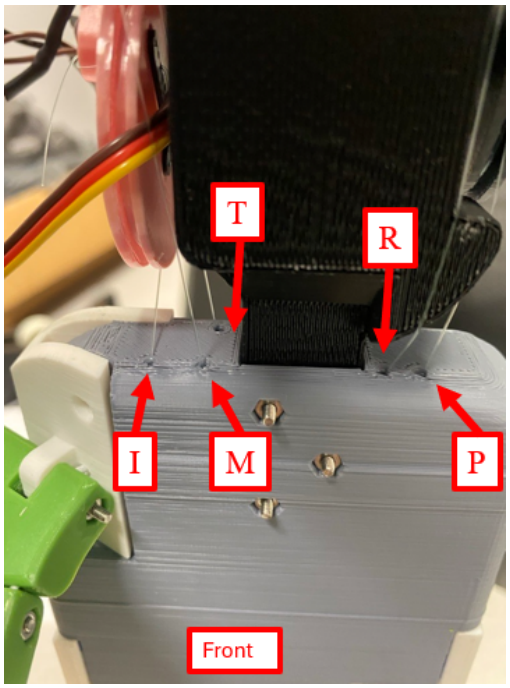
6. Attach the five-servo mount piece to the palm piece. Again, secure nuts in place before connecting pieces.
7. Attach micro servos to the five-servo mount piece. The servos will only fit in the slots in the proper orientation. The wire of the servos will always face away from the piece. 20mm M2 screws should then be used to ensure the servo motors remain within the mount. They get run through the mount piece and the wing of the servo. Attach all five servos.
8. Attach servo arms to pulley pieces. The servo arm should fit into the pulley piece cutout. From here, the fishing line is used to effectively “sew” the servo arm and pulley pieces together with the holes that line up between the two.



9. The final step is getting the fishing line connected and wrapped around the pulleys with proper slack such that servo motion causes the open/close finger motion. This step can be a bit tricky and require some iteration. It doesn't really matter which string goes to which pulley, but the way I left them on Armold was what I found to make the most sense. The pulleys are labeled with the finger they control as well as the direction the fishing line wraps around them. The line could be wrapped either way depending on what is 0 and what is 180 in the code. Labelled pulleys and string channel diagram:







10. To connect the hand to the pronation assembly, the green bracket (fore-arm elbow connection bracket) piece is used. This must first be screwed onto the pronation assembly via several screw holes in the center. Then, the black five-servo-mount piece of the hand can be placed into the green bracket and screwed on.

5.1.4.3 Hand Maintenance

So far, the hand has not had operation issues, although it is predicted that over time various components may experience the effects of fatigue.

Fishing line: The fishing lines can be replaced fairly easily, although if the fingers were initially assembled with a line long enough that it wraps around the pulley multiple times, the line does not need to be replaced. Rather, the pulley can be taken off of the servo, adjusted, and reconnected to adjust tension. If the line is not long enough, it can be replaced by cutting and tossing the old line and running a new one through the finger to the pulley.

Some of the string channels (see the wireframe in the engineering drawing) curve much more than others. Specifically, the index and thumb fingers are the most challenging to restring. I've found it much easier to do so by folding down the end of the fishing line (such that the end you are about to thread into the channel is doubled in thickness) before feeding it into the string channel. This makes the end of the string more rigid and you are able to finagle it where you need to.

Finally, note the fishing line can be adjusted both manually and with code. It is sometimes necessary to run code to move a servo from 0 to 180 (or to do so by turning on Armold and using the control panel) to see where the motion of the servo occurs such that the string and pulley piece can be repositioned. It can be tricky to work with because the hand micro servos do not have a mechanical stop; if you manually move their position, you will lose track of where their range of motion is.

Elastic: If a finger is not snapping back properly, the elastic must either be tightened or replaced. So far, it has never needed to be replaced. However, tightening and replacement have similar solutions. Either way, the finger-palm connection piece must be removed. This will enable access to the elastic and you may either untie and re-tighten or re-thread elastic.

Screws: Many of the screws, particularly on the fingers, have Loctite to ensure they do not come undone. While the finger screws are intended to act as pins, they sometimes do not. Occasionally they may start to fall out and need to be replaced or need re-loctited. If you add loctite to a nut, <u>do not</u> simply screw down as tight as you can. The 3D-printed pieces are small enough that this deforms the print and will mean the fingers can no longer bend.

5.1.4.4 Hand Suggestions, Notes, Improvement Opportunities

An important note about the hand is that 2 designs were finalized and used. The hand that is currently on Armold and used for pictures above is iteration #2. Iteration #2 was designed to be easier to assemble (it is, by far!). However, this iteration also included changes that were made to the fingers such that they could not close as far (less than ninety degrees per digit). This change was made intentionally such that gripping capabilities would be improved, but it did not seem to aid with gripping very well and, in my opinion, is aesthetically worse. *The overall takeaway from this is that the best hand would include an assembly of the iteration #2 pieces: Palm piece, 5-servo mount piece, Forearm-elbow bracket connection piece, Finger-palm connection piece, Thumb-palm connection piece; and the finger phalange pieces from iteration #1. This assembly would result in the hand build which is easiest to assemble, best functionally, and best aesthetically.*

Some other improvement ideas:

- Create a better guy-line piece used to tension the strings on the hand that is actually functional and serves more of a purpose than just acting as something to tie the fishing line to.
- Replace the elastic with another set of fishing line that runs down the back of the hand but connects to small, strong springs in the back of the palm. This would improve overall durability since the elastic wouldn't need to be tightened or replaced.
- Modify the design of the five-servo-mount piece so it is easier to take servos in and out (currently a hex nut must be placed very deep in the part and it is difficult to screw the servos in).

- Find an appropriate type of hardware to replace the finger screws with (pins somehow- this is what I wanted to use initially but couldn't find anything).
- Add better labels to the hand pulleys.
- Fix the countersinks on the palm piece so screws do not protrude from the palm.

Some just-for-fun ideas:

- Would be really neat to design a set of fingers which print already linked to one another. Not sure how difficult it would be to do so but still have enough space for the string channels.
- Add an additional range of motion to the thumb so it is able to truly give a thumbs up.

5.1.5 Armold's Control Panel

5.1.5.1 Control Panel Overview

This section will explain the potentiometer dials, sliders, and the laser cut box that holds all of the control panel components. The wiring of the control panel will be explained in section 5.2.2. The control panel is made up of a ⅛" laser-cut wood box, a frame for the touch screen, rotational potentiometers, and linear potentiometers. Both potentiometer types have large controls to make interaction with Armold more fun. The rotational potentiometers have labels for each of the 5 arm motors and the linear potentiometers have been fitted with TPU bumpers which can be seen in the image to the right.

5.1.5.2 Control Panel Assembly Process

The wooden box which holds the touch screen and potentiometers was made with Solidworks files which will be available in the GitHub. The panels were attached to one another using M2 screws and nuts. Rotational potentiometers were placed in slider pieces that allowed them to be pushed in between slits in the wooden top panel. Their controls were assembled using M2 screws, ball bearings, and 3D-printed parts which will also be available in the GitHub. Linear potentiometers were placed in their assembly, shown in the picture above, which was then screwed into the top panel as well. Red TPU bumpers were printed and press-fit onto the sliding arm of the linear potentiometers. The touch screen was slid into the angled wooden panel.

5.1.5.3 Control Panel Maintenance

Over time, potentiometers may wear out or wiring may come loose, causing sensor reading issues. It is good to regularly check the debug output which can be viewed on the touchscreen. From this debug window, the user can tell if the potentiometers are rotating at their full range of motion and that values are being read correctly. The touchscreen should also be regularly cleaned with a dry microfiber cloth to prevent fingerprint build-up and a clear display. Potentiometer dials and sliders should also be wiped down regularly.

5.1.5.4 Control Panel Suggestions, Notes, Improvement Opportunities

There are two main ideas that we feel could improve the control panel. One would be adding a front piece of wood to frame the touchscreen. Currently, if the touchscreen is pulled forward to lean at an angle, it can fall out of the laser-cut assembly. Locking it in with a separate piece would prevent this from happening. We would also recommend adding a back casing to the control panel that would still allow easy access to the battery pack ON/OFF switch but would keep the wiring system contained.

5.2 Electrical System

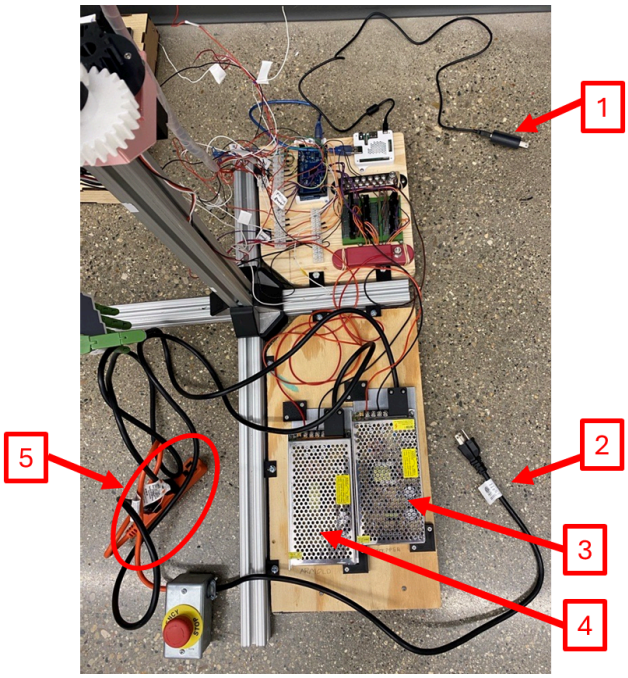
Your understanding of Armold's electrical system functions may vary greatly depending on your background knowledge of electronics and motors. Since there is no way for the team to know who may be working on Armold in the future, this section will be written under the assumption that you have very little robotics experience.

There are two main components to Armold's electrical system: the system on the control panel, and the system on the robotic arm. Each will be documented in detail here.

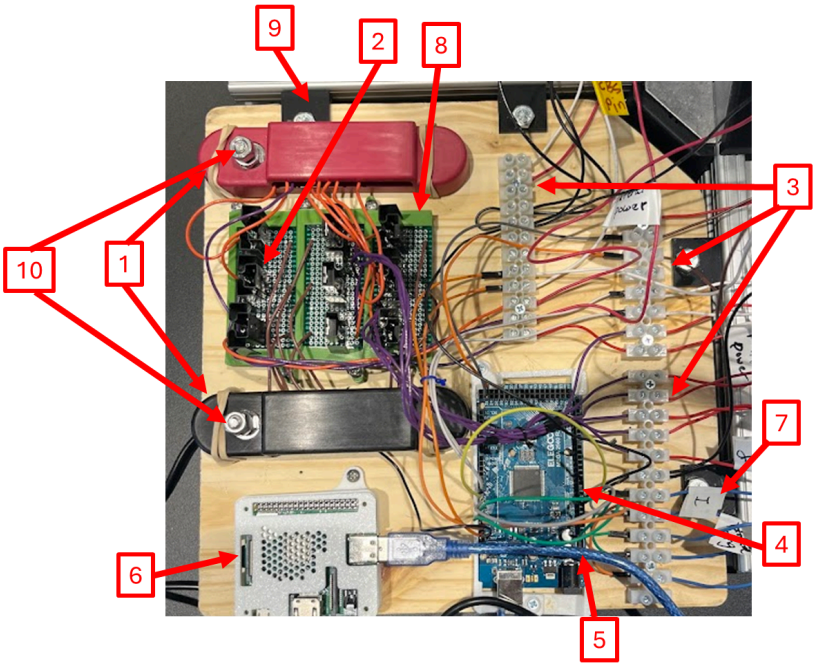
5.2.1 Armold's Electronic System

5.2.1.1 How Everything Works

Armold's electronics are mounted on the wooden boards at the base of the spine. Wires that are connected to the actuators on the arm run up the spine and are contained with a translucent cable wrap as shown to the right. The picture below shows an overview of Armold's electronic system and the arrows reference the following specific features:

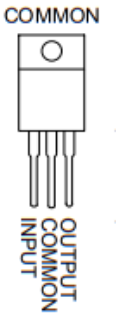


1. Plug-in to provide power to the Raspberry Pi Model 3 A+ (must be plugged in to run Armold).
2. Plug-in to provide power for all of Armold's operations. This cable runs to an E-stop which is wired to an extension cord (5). which allows all power to Armold to be cut in case of an emergency. The E-stop is wired to an extension cord which provides outlets to the two power supplies that provide power to Armold's actuators.
3. Power supply which **only** powers the stepper motor via a DM542T V2.3 motor driver that is mounted on the spine. This is a 24V 5A supply. A separate supply was used for the stepper motor since this motor requires significantly more voltage than the others. Here is a link to where it was purchased.
4. Power supply which provides power to the 9 servos on Armold. This is a 12V 20A supply. This supply was chosen for the servos since they require a low operating voltage (typically 4.8-6.8 volts) but draw a large amount of current when all operated together. 20A was not necessary but was chosen to be conservative. There is more information regarding the current necessary for Armold below. Here is a link to where this power supply was purchased.
5. The E-stop is wired to an extension cord which provides outlets to the two power supplies that provide power to Armold's actuators. The E-stop is wired to an extension cord which provides outlets to the two power supplies that provide power to Armold's actuators.



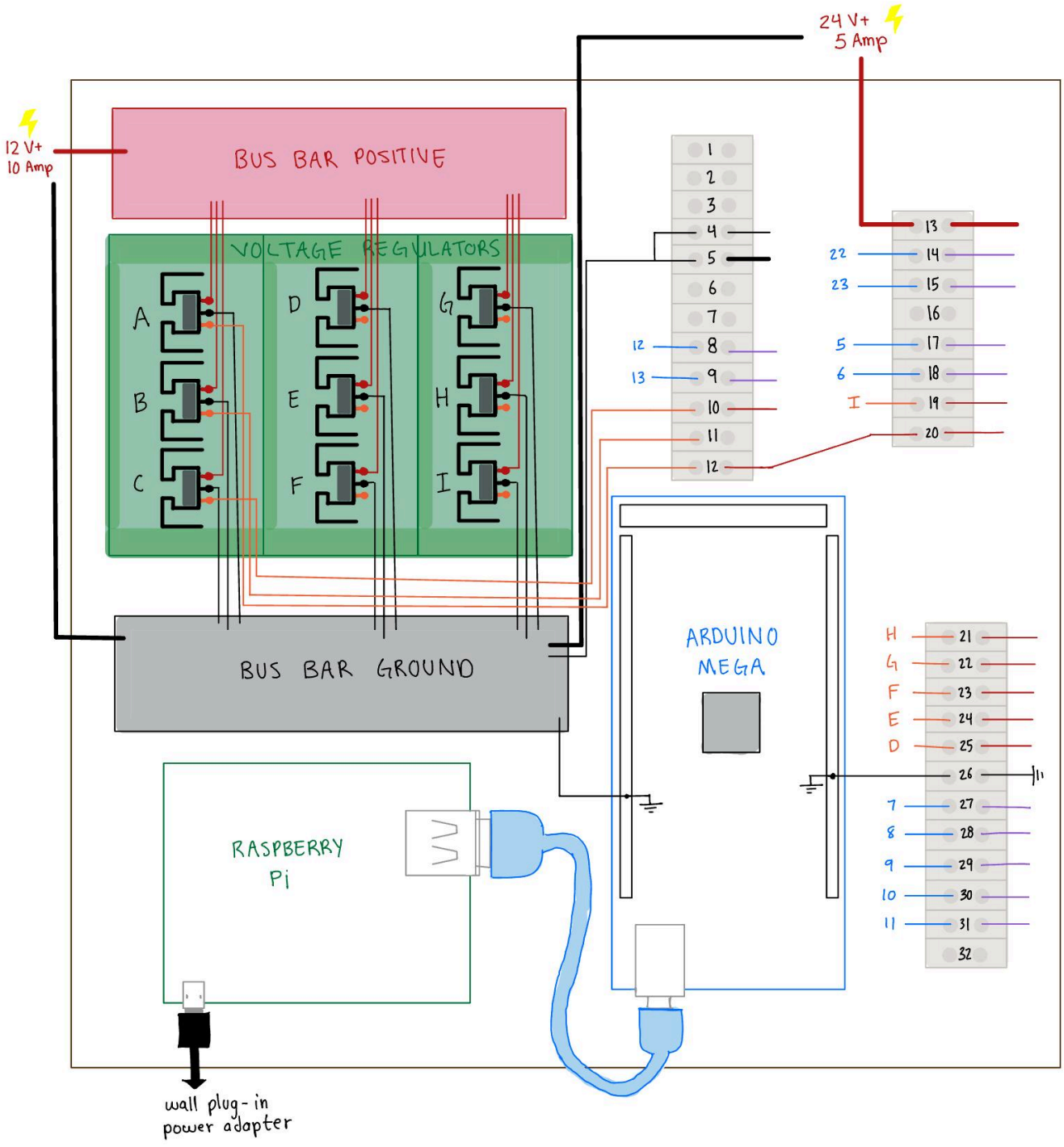
The image above demonstrates the general circuitry information. The bulk of the circuitry is mounted to the smaller wooden board shown on the left. This circuitry contains the features listed below. A user-friendly wiring diagram showing what wires connect at the white wire terminals will follow this list.

1. Power and ground rails. All components are connected to these rails with the exception of the stepper (though that does utilize the ground, but not the power). The rails used are rated to handle 300V/150A; **well** above the necessary power for Armold and are linked here. The top cover is held down with rubber bands but reveals 12 terminals per rail when the cover is removed.
2. LM7805 regulator chips (pictured right) and their heat sinks. There are 9 total; one per servo motor. A datasheet for the LM7805 is linked here. This chip is responsible for stepping down the 12V from the power supply to 5V before providing power to the servos. Each chip has a maximum current rating of 1.5A, which is why one chip was used per actuator (when tested, the actuators were found to pull a peak current of approximately 1.3A under load). The regulator chips are soldered to perf boards. There are 3 wires per regulator chip. A red wire connects the 12V power rail to the input terminal of the chip. A brown wire connects the common (ground) terminals of the chip to the ground power rail. A purple wire connects the output (5V output) terminal of the chip to a servo motor via a wire terminal. Note that there is one faulty solder where the red and purple wires were switched, so in this case, the red wire is connected to the servo while the purple wire is connected to the power rail.



3. Wire terminals were utilized so connections to the motors did not need to be soldered for flexibility (enables easy changing of power supplied or data). Purple 5V output wires run from the LM7805 regulator chips to the terminals, which are connected to power wires for servo motors on the arm. Wire terminals also facilitate the connections between the Arduino Mega (used to control the servos) and the servos.
4. An Arduino Mega was used to control the motions of the servo motors. This was utilized since Raspberry Pis only has analog pins and because the Mega has a large number of connection spots. Some online research also showed that controlling the servos with a Pi may be more likely to cause jitters, so an Arduino was used.
5. A USB A/B cable is used to connect the Arduino Mega to the Raspberry Pi 3 A+. This cable enables serial communication between the Arduino and the Pi, which is necessary because the Pi receives the information from the control panel and then communicates it to the Arduino before the Arduino moves the servos.
6. A Raspberry Pi 3 A+ is used to wirelessly receive information from a second Pi on the control panel. This process is elaborated on in the communication section.
7. Electrical tape on the wires descending from the actuators is used for labeling.
8. 3D-printed mounts are used in place of standoffs to hold the perf boards with the regulator chips above the wooden board.
9. 3D-printed pieces are used to mount the wooden board on the 80/20 base.
10. The larger screw/nut on the power and ground rails are the connection terminals to Armold's power supply. 16 gauge wire is used to connect the power supply to these terminals.

All of these wiring connections in a tight location create a not-so-easy on the eyes scene. In an attempt to organize our wiring we developed a wiring diagram sketch with labels for each voltage regulator and wire terminal. These labels correspond to the wiring they are connected to in the tables following the wiring diagram below. Some consistencies in the wiring diagram below are red wires are power, black wires are ground, orange wires are (purple in the real world) are outputs from voltage regulators, and blue wires are connected to the arduino.



This table goes along with the wiring diagram above to help declutter the picture of wire connections.

Wire Terminals			Voltage Regulators	
Label Number (T)	**Input Wire (from controls/power)**	**Output Wire (to the arm)**	**Label Letter**	**V-Reg output location**
1	vacant	vacant	A	V+ Pronation servo (T12)
2	vacant	vacant	B	T11
3	vacant	vacant	C	V+ Cross-body servo (T10)
4	Ground (T5)	Ground wires from Arm	D	V+ pinky figer micro-servo
5	Ground Bus Bar	Ground wires from Arm	E	V+ ring figer micro-servo
6	vacant	vacant	F	V+ middle figer micro-servo
7	vacant	vacant	G	V+ index figer micro-servo
8	Arduino pin 12	Lateral raise servo logic	H	V+ thumb micro-servo
9	Arduino pin 13	Cross-body servo logic	I	V+ Elbow servo (T19)
10	V-Reg C	V+ cross-body servo		
11	V-Reg B	vacant		
12	V-Reg A	T20		
13	24V power supply V+	Stepper Motor Driver V+		
14	Arduino pin 22	Stepper direction digital signal (on motor driver)		
15	Ardunio pin 23	Stepper step digital signal (on motor driver)		
16	vacant	vacant		
17	Arduino pin 5	Elbow servo logic		
18	Arduino pin 6	Pronation servo logic		
19	V-Reg I	V+ elbow servo		
20	T12	V+ pronation servo		

21	V-Reg H	V+ thumb	
22	V-Reg G	V+ index finger	
23	V-Reg F	V+ middle finger	
24	V-Reg E	V+ ring finger	
25	V-Reg D	V+ pinky finger	
26	Arduino ground	Ground from arm	
27	Arduino pin 7	Index finger logic	
28	Arduino pin 8	Middle finger logic	
29	Arduino pin 9	Thumb logic	
30	Arduino pin 10	Pinky finger logic	
31	Arduino pin 11	Ring finger logic	
32	vacant	vacant	

5.2.1.2 Specifics Regarding Motors Used/Armold's Current Budget

Powering Armold has been a big challenge faced in this project: what is the best way to do it? Servo motors have an operating voltage range, and pull as much current as necessary while operating. When choosing how to power a servo, it is important to provide voltage within the operating range, but also to ensure that it is able to pull as much current as it needs. This is not challenging when you are looking to power one or two motors at a time, but since Armold has ten motors, he requires a large amount of current. The current budget provided below is a crude estimate based on testing done in fall of 2023 where the servo motors were loaded to their stall torque. As this was done the peak observed current was noted. Overall, the micro servos and larger servo motors have similar current demands. Current information for the stepper motor has never been determined since, firstly, we aren't sure how to do this, and secondly, it is powered from its own supply anyways.

It is important to note that the stall torque of a servo motor increases as you move towards the higher end of the operating voltage. All motors were initially powered with 5V to minimize total current draw. It was then found that the lateral raise servo would stall, so it's power was increased to 7V by using another supply (another improvement opportunity is to replace the lateral servo 5V regulator chip with a 7V regulator chip).

In general, the team worked to remain overly conservative in terms of current draw which is why the components are powered with a supply that is rated up to 15A. It is always important to remain aware of the total current draw (between Armold and the control panel as well) to avoid blowing a breaker. From the numbers below, it seems that we are pretty safe. At one point, current values were drawn for the control panel but a current budget was not created as the current required by the sensors is so low relative to Armold that it wasn't worth pursuing further.

			Technical Details of Interest				
Section of Armold	**Motion on Armold**	**Actuator Type**	**Stall Torque (Kg*cm)**	**Operating Voltage (V)**	**Supplied Voltage (V)**	**Approximate Operating Current (A)**	**Approximate Peak Current (A)**
Shoulder	Frontal raise	Stepper motor	unknown	uknown	24	unknown	unknown
	Lateral Raise	Servo motor	40	4.8 V - 6.8 V	7	0.8	1.3
	Cross-body	Servo motor	25	4.8 V - 6.8 V	5	0.8	1.3
Elbow	Elbow bend	Servo motor	40	4.8 V - 6.8 V	5	0.8	1.3
	Pronation	Servo motor	40	4.8 V - 6.8 V	5	0.8	1.3
Hand	Thumb	Microservo	3.1 - 3.5 (at 4.8V and 6V, respectively)	4.8 V - 6.0 V	5	0.5	1
	Index	Microservo		4.8 V - 6.0 V	5	0.5	1
	Middle	Microservo		4.8 V - 6.0 V	5	0.5	1
	Ring	Microservo		4.8 V - 6.0 V	5	0.5	1
	Pinky	Microservo		4.8 V - 6.0 V	5	0.5	1
Totals						5.7	10.2

5.2.1.4 Electrical System Suggestions, Notes, Improvement Opportunities

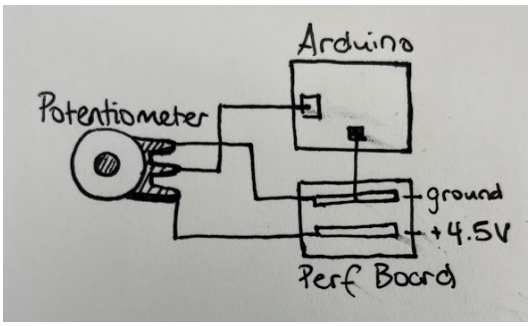
One improvement that we believe could benefit the electrical system would be to change out the voltage regulator chips on the motors for adjustable ones. Currently the voltage regulator chips are set to convert 12V to 5V. However, the servo motors can operate at a range of 4.8V to 6.8V and this also changes the amount of torque that the motors output. Replacing the chips with adjustable ones can allow power to be distributed where necessary for the current motors, however, the current will still need to be considered as the system is limited due to the power supply outputs.

Another recommendation would be to add “checkpoints” onto Armold, or places where you can easily test electrical connections. Nothing like this is implemented currently so if an actuator were to fail due to electronics, it would be quite difficult to determine where the failure is.

5.2.2 Control Panel Electronic System

The control panel’s electrical system consists of a Raspberry Pi 4B, an Arduino MEGA, a battery pack, a Sunfounder touchscreen, a 12V 8A plug-in power supply, 5 rotational potentiometers, and 5 linear potentiometers.

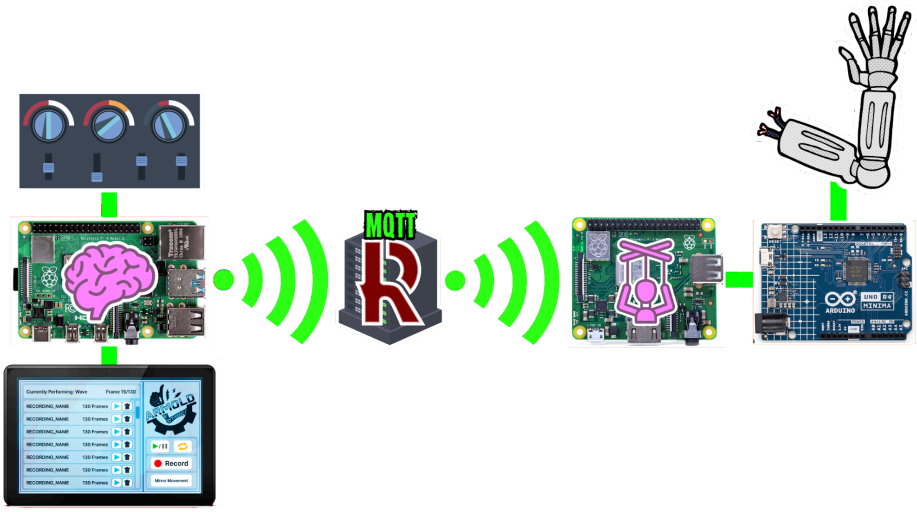
Each potentiometer is wired to a ground rail and a voltage rail on a perf board. The perf board is soldered to a battery pack with 3 AA batteries that provide about 4.5V of power to the potentiometers. This method was used due to the battery pack being available at the time. This has also allowed the project to be run on one less large power supply (which is optimal since there are so many) while maintaining the ability to still switch the power ON/OFF. In the future, this could be changed to a 5V supply. The potentiometers are also connected to the Arduino which communicates with the Raspberry Pi, as explained in section 5.2.1.1. As these potentiometers which are assigned to actuators on the arm are rotated or slid, the corresponding part on the arm moves. The amount that these motors move is determined by values assigned within the code. The Sunfounder touchscreen, Raspberry Pi, and Arduino are all powered off of each other via serial connection and using the 12V 8A power supply.



5.3 Communication System

5.3.1 Communication System Overview

As Armold is intended to wirelessly transmit instructions from its control panel to its robotic arm, two different Raspberry Pis are utilized to handle both running the software of the system as well as transmitting data between one another. The method by which the two Pis communicate is called paho-MQTT, which is a robotics- focused protocol in which one device publishes data to a server, and another device subscribes to that server to receive its data. As of now, this method enables Armold to seamlessly transfer data from one Pi to another with negligible data loss and/or latency. The following diagram depicts the way in which the devices and the server transfer data:



5.3.2 Communication System Maintenance

Currently, the server which Armold utilizes to publish and read back data from is located on Rose-Hulman's campus and is managed by EIT. This server was previously only used for a robotics class, and was disabled when the class was out of session; however, it was brought back online permanently for Armold's use. So long as both Pis are able to connect to the Rose-Hulman network (RHIT Open wifi is fine), they should be able to connect to the server to transmit data. If the issue arises in which the control panel's Pi is stuck on "searching for its arm," it may be the case in which EIT has shut the server down temporarily. To fix this issue, simply contact EIT and have them confirm that it is online.

5.3.2 Communication System Suggestions, Notes, Improvement Opportunities

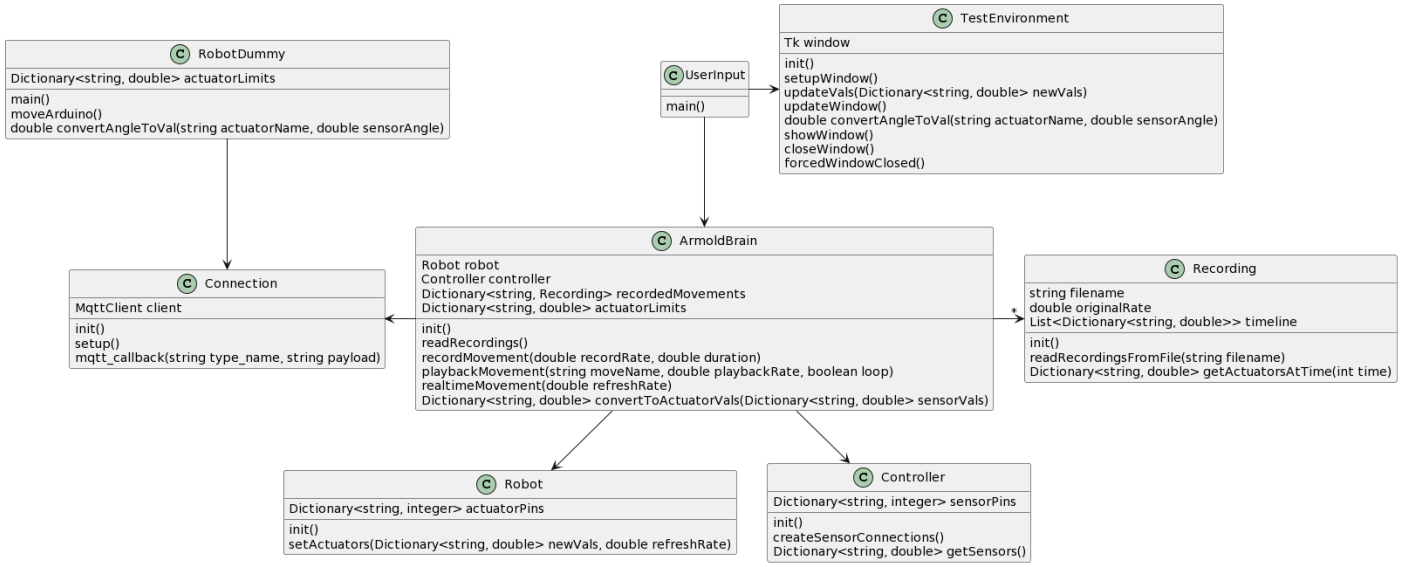
While the Rose-Hulman MQTT server works just fine, there are some minor issues in using it that could be improved on in the future. For starters, any Rose-Hulman student able to connect to the network is also able to publish data to the MQTT server, and thus would be capable of overwriting any data that Armold's control panel is trying to send to its arm (as a worst-case scenario, a malicious student could force Armold to be stuck in a less-than-savory pose indefinitely). This is definitely a security risk- the only real workaround would be to either switch to a more secure method of wireless communication or have EIT require devices to be whitelisted on the server before they are able to publish information/restrict publishers to the "Armold" topic to only the control panel's MAC address. Another issue is that currently, the code does not handle any sort of timing-out on connections. For example, this means that if the control panel is able to connect to the server but the transmission rate is extremely slow, it could slow down and lag the entire system if every single "frame" of data takes longer than it should be allowed to send. Currently, the team hasn't found this to be a substantial issue, however, it would definitely be helpful for future members of the Armold team to look into it. We would recommend perhaps switching the system to use a more direct peer-to-peer method of data

transfer (such as sockets; which are fairly straightforward to implement - changing to use this implementation in favor of MQTT was not done by our team during this final quarter due to us only becoming familiar with it near the end of the quarter and trying to get what we had done working, but we imagine it would only take a week or two to make the switch in the future).

5.4 Software

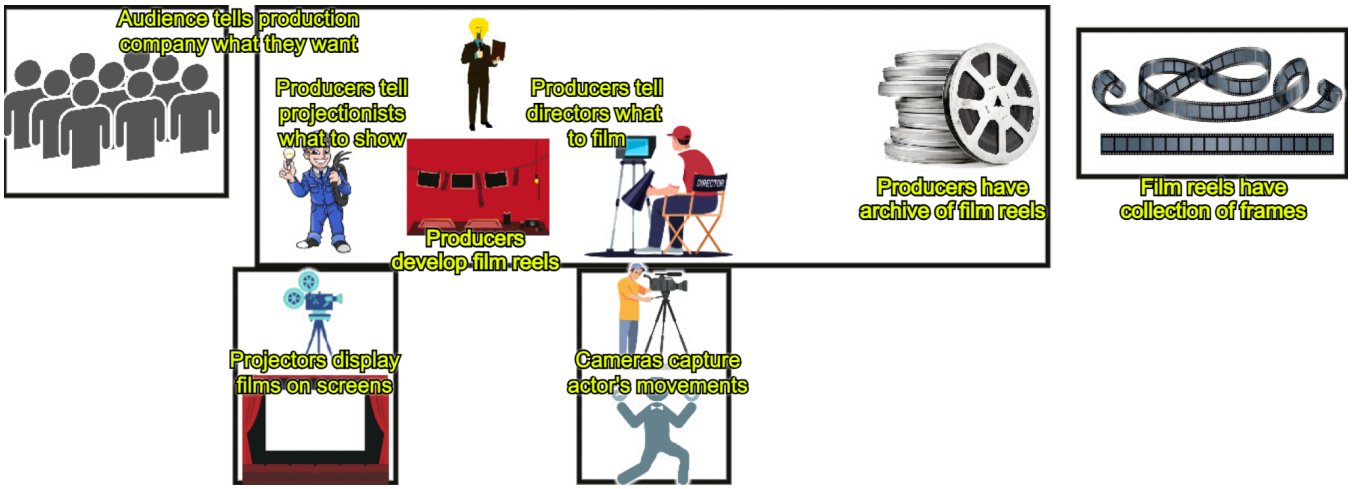
5.4.1 Software Overview

Armold's control panel and arm each have their own unique programs written in Python which handle everything from receiving instructions to recording data to interacting with the physical components of the system. Each Raspberry Pi of the system has all of the code available from Armold's Github repository downloaded and runs their respective program upon boot (armold.py on the control panel Pi and robot.py on the robotic arm Pi). Each program references the pyFirmata package, enabling the Raspberry Pi to have full control over their Arduinos. The software of the system as a whole matches the following UML diagram:

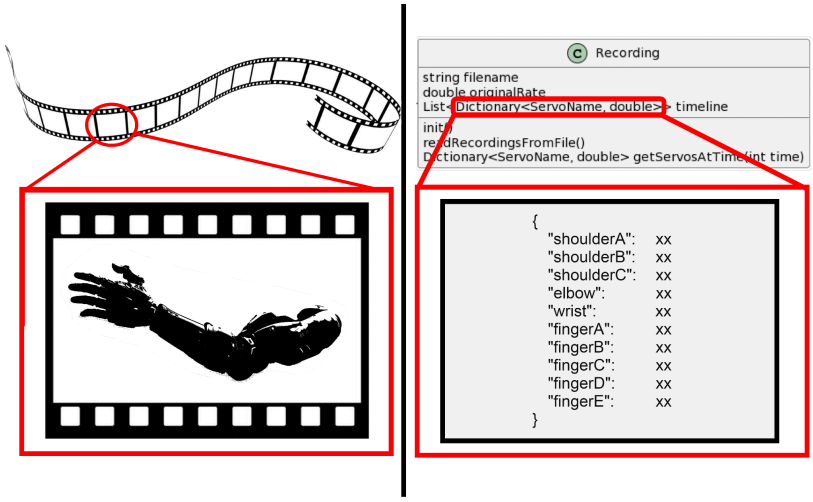


The program robot.py encompasses the RobotDummy class above (although there is no literal "RobotDummy" class in the code; this is simply a figurative class) and includes a single reference to the Connection class. The program armold.py contains the rest of the classes ("UserInput" is another figurative class; this essentially denotes how the GUI has power over all other classes) as well as one other reference to the Connection class. This reference to the connection class in each program handles the Communication system covered in the previous section.

The design of the software can be understood by thinking about it as a film production process. The user (audience) lets the software through the GUI (the producer) what they want the arm to do (what movie they want to see). Then, the software (the producer) either begins to read data from the control panel sensors (sends a director to record an actor) or pulls a recording from memory (pulls a film reel from the collection). As "frames" arrive, the software (producer) converts the sensor values to valid servo/motor values (develops the film) and sends the "frames" over to the robotic arm's software (a movie theater projectionist) which then signals the arm to move to match the "frame" (projects the frame of film onto the screen). If the user wishes to record a motion (add a film to the collection), the software (producer) would then additionally save the "frame" to a new recording (splice the new film "frame" onto a new film reel). After receiving user input (audience feedback), this whole process occurs incredibly fast, allowing for the motion of the arm (film on screen) to occur smoothly.



The actual recordings (film reels) contain a list of "frames" in a timeline. These "frames" contain all of the positions of where each of the joints of the arm (posture of an actor) should be at any given instantaneous moment in time. Similar to film, the high speed at which these "frames" are cycled provides the illusion of smooth motion.



5.4.1.1 armold.py

The software which runs on the control panel (armold.py) uses Tkinter to display its GUI for users. This program has one primary loop which seeks user input on any of its buttons and performs the appropriate action for each one: mirroring real-time movement from the control knobs to the arm, recording and saving movements from the control knobs, and playing recorded movements on the arm. Essentially, once a button is pressed, the system breaks away from its main loop and performs a secondary loop where it requests the appropriate data (positions from a saved recording or positions of control knobs from the connected Arduino) for a frame (1/30 second) and publishes the frame to the MQTT server; repeating this loop 30x a second until told to stop doing so by the user before returning to its main loop again.

5.4.1.2 robot.py

The software that runs on the robotic arm (robot.py) simply follows an infinite loop of checking if new data is published to the MQTT server and then tells its Arduino to move each servo or stepper motor to the new target positions. Rather than jumping straight to the new positions, the software first checks that the position is within the preset value range, limiting if needed, and smooths the motion of the arm over the full duration of the frame rather than moving all servos/motors to the correct position instantaneously. Additionally, the software permanently records the

position of the stepper motor at any given time such that if a power outage were to occur, on reboot it would be able to move back to its home position without causing damage.

5.4.2 Software Maintenance

5.4.2.1 Connecting to Raspberry Pi's

If maintenance needs to be performed on either Raspberry Pi in which the software runs, an SSH connection can be established through a user's terminal by using the hostname and static IPs "pi@ArmoldPrimary" (for the control panel) and "pi@ArmoldSecondary" (for the robotic arm) with the password "Armold." Once connected, users can use basic Linux commands to move through the directory and perform operations as needed ("cd foldername" to enter a folder and "ls" to list all files within the directory).

5.4.2.2 Ending a process

As the python programs on either Pi are set to automatically run on boot, in order to stop a program on either device from running (in case modifications to either device need to be made and the running code makes doing so difficult), simply connect to the respective Raspberry Pi using the method above and enter "ps -A" to list all running processes and then enter "kill #" replacing the # with whatever the PID of the running "python" program is.

5.4.2.3 Updating software

After modifying the code and committing/pushing changes to Github, simply connect to both Raspberry Pi's using the method above and enter "git pull" in the terminal while inside of the "Armold" directory.

5.4.2.4 Changing Arduino Pin Numbers

To change the pin number of a sensor/servo on the Arduino to which either device is connected to, simply open "pins.txt" and replace the number associated with the respective device.

5.4.2.5 Adding new sensors/servos or modifying range of motion

Within "armold.py," and "robot.py," add the name of the new joint to each of the lists at the very top of each program with the respective values for each list. Then, add the joint to "pins.txt" using the method above. So long as all the values are accurate and physically possible, the system should "just work."

5.4.3 Software Suggestions, Notes, Improvement Opportunities

The software of the system was made with flexibility and modularity in mind. As such, it is easy to modify and add as many new joints as needed. However, it could be made easier: currently, both "armold.py" and "robot.py" have duplicate dictionaries at the top of their programs. These dictionaries could be moved to a new file and read at runtime, similar to "pins.txt." Additionally, while the GUI of the system works just fine, it could also be improved with better practices within the code; as it was mostly done in an effort to "get it working." It's a bit chaotic to read and probably not super flexible in terms of being future-proof, but it was made organized enough to work and has comments everywhere explaining what it's doing. Finally, the GUI seems to "flicker" some whenever it refreshes/redraws itself after a change performed by a user, so fixing that would be cool. Other than those improvements listed in the communication section, these are the only real concerns the team has identified currently!

6. Evidence of Performance

This section expands on the requirements sections from above, explaining in notes whether or not we passed each requirement with the evaluation metrics.

6.1 User Requirement Performance

Requirement	Evaluation Metric(s)	Status and Notes
The touchscreen will provide clear options for the user on how to operate the control system.	Pass/Fail on having buttons such as *Record*, *Loop Recording*, and *Mirror Motion* can be pressed to record or loop a recorded motion, or live mirror motion.	Pass. These buttons and present and work.
The control panel must have clear labels to communicate how the controls affect Armold's motion to the user.	Pass/Fail on dials have labels for the motion that they perform. Elbow, Cross-body, Lateral Raise, Frontal Raise, and Pronation.	Pass. Labels are present on rotational potentiometers.
The control panel must be durable enough for regular use by college-age students.	Pass/Fail on not breaking during the Rose Show demoing.	Pass. The control panel had no issues.
Armold must be capable of recording and performing motions.	Pass/Fail for recording. Pass/Fail for performing recorded motions.	Pass. Motions can be recorded.
The control panel must be “fun” to use.	Pass/fail for general likability of the control panel’s controls.	Pass. Rose show feedback was positive.
The UI provides some type of feedback to the user which lets them know what state it is in (waiting for input v.s. received input)	Pass/fail on implementation of this	Fail. The UI does not currently let the user know which state it is in.
Armold moves relative to the input from the control system. For example, input to the sensor which controls lateral raise movement means Armold performs lateral raise movement.	Pass/fail. Test with code. < 3-second delay.	Pass. Less than a 1-second delay.
Armold will be fully enclosed during the operation.	Pass/fail.	Fail. Enclosure still needs to be built.
Armold will have signage explaining operation and safety precautions.	Pass/fail.	Pass. Armold has a document labeled “Armold’s Purpose”.
Armold will have an E-stop on the arm’s electronic system, stopping all	Pass/fail.	Pass. An Estop has been created and is used in the powering process.

motors from moving at once.		

6.2 Keeper Requirement Performance

Requirement	Evaluation Metric(s)	Status and Notes
The project can be moved around and the robotic arm must be small enough to keep on a table.	The total project must be able to be broken down into components that can be carried or put onto a cart.	Pass. Armold can be carried by a stronger person or placed on a cart.
	Evaluation Metric: Armold's bounding box dimensions (LxWxH) must fit within 35"x33"x32" (+/- 1" in any direction) volume when in the home state. Note that this does not include an enclosure.	Armold currently measures at 34.5"x33"x32.5"
The purpose and explanation of Armold must be accessible when the system is used.	A document named "Armold's Purpose" has been created and can be displayed next to his final creation.	Pass. Document exists.
The system's entire code and UI must be incorporated within the project and not rely on any of the team's laptops.	The system can be run in the absence of a laptop connection.	Pass. The system does not work on laptop usage.
The robotic arm must be kept in a safe, locked enclosure that can be seen through so it is not easily accessible or damaged.	The enclosure around Armold must be within acceptable size TBD by the art curator at a later date and be locked 24/7 unless unlocked by Rose-Hulman staff. A kill switch must also be accessible to users or staff in case of electrical issues.	Fail. Enclosure does not currently exist.

6.3 Creators Requirement Performance

Requirement	Evaluation Metric(s)	Status and Notes
Create some methodology to read and demonstrate position data, for example, a graphic with simple lines that represent links of the arm and move the "forearm" accordingly when that potentiometer is activated.	Pass/Fail the system has a graphics display that the creators can use to easily understand how Armold will move when given a specific input.	Fail. The system does not have a graphics display but does have a debug window that reads the positions of the potentiometers on the control panel.

The code will limit the range of motion of the elbow, hand, shoulder, fingers, and cross-body movements so they will have an operating range of motion similar to that of a human and will not self-collide.	The range of motions detailed in Table 1 below is achieved.	Fail. The arm is capable of moving to the values shown in the table, but motors will need to be moved and values reprogrammed.
A person unfamiliar with the project should be able to see Armold and recognize he is an imitation of a right human arm.	Pass/fail on general recognition.	Pass. People recognized the arm at the Rose Show.
The length of the arm from the shoulder must be short enough to clear the base of the 80/20 stand (when fully extended) that Armold is mounted to.	The arm should be ≤ 28 inches.	Pass. The arm measures at 27.5”
The control system will consist of the following outputs per sensor type. These outputs will act as inputs to the Armold arm brain system: - Linear potentiometers: Controls the five micro servos for Armold’s fingers - Rotational potentiometers: Controls the 25 Kg*cm servo for cross-body, the 40 Kg*cm servos for elbow, pronation, and lateral movement, and the stepper motor for frontal raise movement.	Pass/Fail on sensors controlling assigned motors.	Pass. Sensors control assigned motors.
The arm is durable and strong enough to lift objects ≤ 1lb.	Pass/Fail on lifting 1lb.	Pass. The arm can lift 1lb items.

7. Operating Armold

7.1 The Basics (Startup and Operation)

1. Plug in the two power supplies on the arm to the Estop extension cord.
2. Plug in the third power supply to the Estop extension cord and set it to 6.8 volts. Attach the nodes of this power supply to the red wire labeled “lateral power” and one of the screws on the ground bus bar.
3. Plug in the Raspberry Pi on the arm.
4. Plug in the control panel power supply and move the switch on the battery pack to the ON position.
5. Wait for the arm to move and the motors to power on.
6. When the arm “wakes up”, you may begin live control, recording motions, or playing pre-recorded motions.

7.2 Troubleshooting

- Because the Arduino to the arm is powered by the connections to the Estop, if the Estop is pressed, the Arduino will reset. This causes an error in the Raspberry Pi. In this case, unplug the Raspberry Pi for a few seconds and allow it to reset itself. The arm will shortly come back to life.
- The lateral raise servo can be removed easily for assembly purposes, however, this causes the servo to sometimes slip out of place while moving. This can be fixed by rotating the assembly and placing it back in once the arm has been lowered. A temporary solution has been put in place with a 3D-printed clamp and a rubber band.
- There may be times when the finger will no longer bend and this is usually due to the fishing line slipping off of the pulley. To fix this, start by unscrewing the pulley from the micro servo. The pulley should be labeled with the proper direction to wrap the string in and have the string be taught while the finger is in it’s open position. If it is not labeled for some reason, it can be helpful to have another person operate the servo from the control panel so you can see which direction the line should be wrapped; mentally noting that the string should be taught when you screw the pulley on, and then when the servo moves it should pull the string tighter so the finger achieves the close position. It is always ideal to wrap the fishing line while the finger is in the “extended” position (the linear potentiometer is pushed up). Once the line is wrapped on, place the pulley on the micro servo. The servo arm fits tight enough that you can test if the slack is properly set by operating the finger before screwing the pulley piece on, for easier adjusting. Once the tension is right, rescrew the pulley on to the micro servo. Test the extend and grip motions on the control panel.

8. Recommendations and Final Notes

8.1 Additional Information

- CAD files and other important documentation will be left behind in a GitHub repository. Code will also be available in case bugs appear later and Pi’s or Arduinos need to be reflashed.
- Add information on enclosure design ideas (insert Andi’s sketches and information on design ideas)
- If any additional information is needed, feel free to reach out to one or all of us through our Rose-Hulman emails. Some of us will be checking these regularly if communication on the project is needed.
    - Andi Fiani (fianiar1@rose-hulman.edu)- best for hand questions, will be at Rose in fall 2024 and available for questions
    - Chris Steiner (steineca@rose-hulman.edu)- best for CS questions
    - Dylan Dorman(dormanda@rose-hulman.edu) - best for shoulder and spine questions
    - Shelby Schipper (schippsk@rose-hulman.edu) - best for elbow questions

8.2 Additional Armold Improvement Ideas

These are additional ideas or thoughts that we had on the project to improve it in the future so that it may be safer, easier to use, and stronger:

- Create a box or lid to case the electronics at the base of the arm. The wires are a bit overwhelming for users to look at and make the project look less professional. However, be sure that ventilation isn't an issue as some of the voltage regulator chips can get hot as they're used for long amounts of time. Easy access to the wiring should also be considered in case of emergencies or maintenance.
- Currently the control panel debug only checks the values of the potentiometers on the control panel by displaying the degree values. To better know where the arm should be, a graphical debug output, showing an animation of the arm moving could create a better idea for the user on where the arm would be moved.
- Implementing feedback control with rotary encoders could also aid in a more in-depth debug window.
- The GUI currently does not permanently save some pre-recorded motions. If the creators divide that the display wants to have permanent pre-set motions that cannot be deleted, these would need to be made permanent so that any user cannot delete these important motions.
- Our team has noticed some misalignment issues causing buttons near the bottom of the touchscreen to be difficult to press. This could be due to calibration issues or something with the code related to the screen.
- Another important improvement could be machining parts like the elbow or pronation mechanisms out of lightweight metal as the 3D-printed assemblies can easily snap or break. Gears could also be ordered or possibly machined as 3D-printed gears are not very accurate and can often bind due to inaccurate dimensions.
- At times the robotic arm ceases to move and the system has to go through a "cool off" period due to motors or voltage regulator chips overheating. This is most often seen in the elbow and could be due to a couple of reasons. The motor may not be strong enough to efficiently move the arm without stalling a bit. The motor itself could also only be able to withstand a certain amount of use before needing to cool down. Looking into the powering issues associated with the motors or purchasing stronger motors could fix this issue.
- At the moment, the covers for the power and ground bus bar rails are secured with a rubber band. Making these look more professional would be a positive addition to the system. However, the covers should be easy and quick to remove in case a wire becomes disconnected or needs to be fixed.
- Currently, there is not start-up screen on the GUI. Having a start-up screen that shows basic instructions on how to operate Armold safely, Arnold's purpose, and prompts the user to turn the battery pack on the back of the system to the ON position could be a useful addition to the system. This also ensures that the user must view the information before operation.