An Accessible, Adaptable, 6-axis, Low-Cost 3D Printed Robotic Arm Solution

Frank Lucci

Basis Shavano San Antonio Campus

International Science and Engineering Fair

April 22, 2024

Abstract

Robotic arms are useful in a variety of applications: manufacturing, de-palletization, artificial intelligence research, etc. However, on average, a typical robotic arm with 6-axis capabilities can cost many thousands to millions of dollars depending upon the payload, accuracy, repeatability and workspace size. Additionally, if an arm needs to be utilized for a task requiring different constraints than originally intended, expensive modifications are needed to meet requirements., High initial costs and expensive modifications deter many independent researchers, small scholastic institutions, hobbyists, and others with limited budgets from having access to the possibilities that robotic arms offer. To address this issue, an adaptable, 6-axis, low-cost 3D printable robotic arm that is compatible with Robotic Operating System (ROS) was designed, built, and tested for those who can't afford industrial arms but who still need a comparable utility. The robotic arm was designed to have a maximum reach of 800mm at a standard length that it could be easily extended and modified for specialized tasks. Additionally, the robot was designed to have a repeatability of under 2mm, a payload of 1kg at the standard maximum length, an accuracy below 1cm, and a cost under $800. When refined and tested at the maximum length, the average repeatability was 1.13mm. The accuracy ranged from 0.07cm to 3.58cm with an average of 1.38cm. The maximum maneuverable payload was 0.644kg and the max holding load was 1.45kg for lifting at 800mm. Finally, the price, adaptability, and compatibility with ROS were successfully implemented.

Table of Contents

Introduction

Robotic arms are used everywhere: industrial manufacturing, pick and place tasks, painting, AI research, the list goes on. However, on average, a typical industrial robotic arm with 6-axis capabilities can cost anywhere from depending upon the payload, accuracy, repeatability, and size of the robot's workspace. Additionally, if an arm is purchased and needs to be utilized for a task requiring different constraints than originally intended like a higher payload, greater precision, or greater length, expensive modifications may be required or an entirely new robot may be purchased to meet the new requirements. Coupled together, this prevents many independent researchers, small scholastic institutions, hobbyists, and others alike from having access to the wide range of possibilities that robotic arms offer.

The engineering goal was to design, build, and test an adaptable, 6-axis, low-cost 3D printable robotic arm with a sizable payload, workspace, repeatability, and accuracy. Additionally, it would have been compatible with ROS and Python so that it can be used for any complex research tasks. The robotic arm would have a maximum reach of 800mm at the current design but could be easily modified. These modifications include the lengths of the links but could also include separate grippers, motor mounts, timing pulleys, and more. At this standard length, the robot would have a repeatability of under 2mm, a payload of 1kg at the standard maximum length, and an accuracy below 1cm. Additionally, the cost of all the components would not exceed $800.
This paper presents a method of engineering an inexpensive, general-purpose robot. Additionally, it provides insight into the challenges faced when designing an inexpensive robotic arm and can be used by others for the consideration of different factors when designing their robotic arm.

Materials

Material (Electronics)	Price	Quantity	Material (Mechanical)	Price	Quantity
Arduino UNO	$17	1	Polylactic acid (PLA) filament 1.75mm	$19	5
NVIDIA Jetson Nano	$149	1	Metric Screw Kit - 2M,3M,4M,5M	$27	1
Nema 23 Stepper 4.2A 3NM	$33.33	3	16mmOD 5mm ID deep groove bearing	$13	
TB6600 Stepper Driver	$8.33	4	22mmOD 8mmID deep groove bearing	$18	
Nema 17 Stepper 2A 59NCm	$14	2	8mm shaft flange coupler	$15	
Nema 11 Stepper 1A 16NCm	$12	1	660mm 5M 12mmW timing belt	$8	2
DM5423 Stepper Driver	$29	3	645mm 5M 15mm W Timing belt	$11	1
Electromagnet 12V 25N		1	M8*50 bolt w/ hex screws	$8	2
TCA9548A	$1.83	1	8mm shaft collar	$9	1
Nema 23 Stepper 2.8A 1.2NM	$25	3	8mm*250mm steel rod		
JQC-3FF-S-Z Relay	$2	1	2M pitch timing belt (adjustable) with 2M pitch pulleys (kit)	$17	1
AS5600 AbsoluteEncoder	$3.66	6	5M 6mm width customizable timing belt	$23	1
1K Ω resistor			5mm bore shaft flange couple	$4	2
2K Ω resistor			Cutting board/mounting board of choice		
220 Ω resistor			Wood screws		
Jumper wire			10K Ω resistor		
20-16 AWG wire			5K Ω resistor		
Old Cable					

Engineering Goal

The inception of the engineering goal was formed through the consideration of different design limitations and implementation of the purpose of the project. I first considered the different parameters that the goals should be set around. The parameters included payload, workspace, adaptability, maneuverability (6-axis), repeatability, ROS integration, and accuracy, but the central focus was to reduce the cost for the greatest general functionality that could be implemented in a wide variety of research applications, including pick-and-place tasks, robotics research, education, inspection, etc. To engage in most of these activities, the robot needed to be built based on general requirements for successful robotic arms set in the past. To determine the desired reach, I looked to Universal Robots' UR3's workspace, a highly regarded 6-axis robotic arm for general research tasks, and similar robotic arms. With a workspace ranging from 500mm to 1300mm for the average "general purpose robot" and an average payload at max length of over 1kg, it was determined that a length of the robot would be between 500mm and 1300mm, and it would have a payload of more than 1kg at the maximum length to be useful for most implementations. A repeatability benchmark was set at around 2mm (less is better), because at this scale +-1mm is more precision than is required for many tasks like sorting piles, inspecting parts, and most lab automation. An accuracy benchmark of +-1 cm was set to adjust an XYZ position based on environmental feedback. Additionally, to achieve the bar of around 0.1mm for most industrial robots at this length, a 15-bit absolute encoder is needed. Calculations were done by taking the desired length of around 800mm and finding the deviation based on the sensor quantization. A 12-bit absolute encoder has 2^12 = 4096 counts per revolution (CPR) and 360/4096 is 0.0879 degrees of accuracy and then 800m * sin (0.0879) is 1.227mm of deviation and each angle count for the base encoder. However, to achieve close to 0.1mm repeatability, a 15-bit encoder is needed: 800*sin (360/(2^15)). This is only for the base rotation. Because the smaller joints have a greater repeatability due to a smaller length, the accuracy of the 12-bit encoders works just fine and most likely results in a smaller precision. The 12-bit encoders are much cheaper than the 15-bit and a full switch to them would cost at least $100 more for the induction versions and up to around $600 for the shaft coupling versions. Therefore, the standard robot was designed with 12-bit encoders. Unlike most industrial robotic arms, another main goal of the project was to incorporate a design element of low-cost adaptability into the design so that absolute encoders could be easily upgraded for even higher precision, lengths could be changed depending on payload requirements, and new parts could be added for task-specific needs. To give users of the robot an easy and highly flexible platform to utilize the robot for a countless number of different tasks, the robot's hard-wiring and software was designed to operate with ROS through a ROSSerial protocol to the Arduino microcontroller. ROS or Robot Operating System is an open-source framework, not an operating system, that contains software libraries and tools that aid in the goal of making robots better and more accessible. through a ROSSerial protocol to the Arduino microcontroller. This also allows for easy replacement or substitution of the main Microprocessor (Raspberry PI, Jetson Nano, etc.), and adding or swapping sensors to the robot. The price was placed at $800 due to the estimation of the essential electronics of around $430 with the price increasing much more for higher torque motors, better encoders, stepper drivers, and power supplies for a setup that would be able to hold around the goal of 1kg and be a max length between 500mm and 1300mm. The length of 800mm was chosen to maximize the reach while still producing enough torque at that length to lift the goal payload. The initial torque estimation calculations are shown in Figure 1.

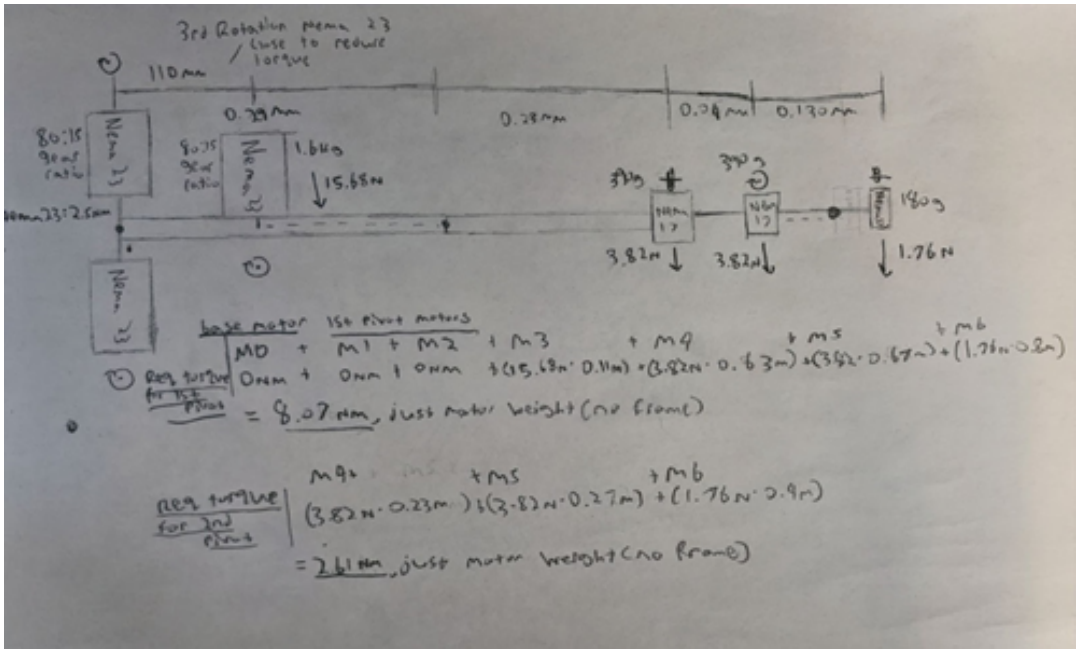

Figure 1. Initial torque considerations using parts under the price goal.

Since torque is T=F*L, the total torque load for the weight of the motors can be calculated by T = (gM1*L1) + (gM2*L2) + … for each different pivot. In doing so, we can observe that the torque due only to the motors is 8.07Nm plus 7.84Nm (1kg*9.8m/s^2*0.8m) is 15.91Nm without the frame, while the total pivoting max torque is 26.66Nm ((80/15) *2.5Nm*2), leaving 26.66Nm - 15.91Nm = 10.75Nm to be allocated to frame weight. The angular acceleration could be calculated through T = I*A where I is (m1r^2 + m2r^2…). Substituting we get I = (1.6kg*(0.4m) ^2) +(0.39kg*(0.63m) ^2) +(0.39kg*(0.67m) ^2) +(0.18kg*(0.8m) ^2), therefore I is 0.701kg/m^2. Because the torque does not significantly decrease until 60rpm and the speed will knowingly be less than that, we can use rearrange A = T/I, so the theoretical maximum angular acceleration for no frame weight is 10.75Nm/0.71kg/m^2 is 15.14 rads/sec^2, and for 0.74kg of clearance for frame weight, the angular acceleration is 10.75nm-(0.35m*(3.13kg-0.74kg)*9.8m/s^2)/0.71 = 3.59rad/s^2.This also means it can travel 1 radian in 3.59 = f(a) where double antiderivative is (3.59t^2)/2 = f(x) substituting t = sqrt((½)*2*3.59) which is 0.58s*2s =1.06s to reach 1 radian of rotation with a 1kg payload including acceleration to mid-point until slowdown. Additionally, at CM at 0.35m, the maximum mass of the casing and other on-board devices would be 10.75Nm/(0.35m*9.8m/s^2) = 3.13kg. Therefore, a lightweight frame would need to be designed to be both versatile and adaptable, yet lightweight. 3D printed plastic was perfect for this. This means that to have safe margins and a rigid body without exceeding $800, a length of 800mm was finalized and so was the payload goal of 1kg.

Initial Design

With a clearly established engineering goal, the next step was to begin drafting a design for the robot. To begin, we must consider the different design factors to produce an efficient structure that accomplishes the desired goals.

Physical Robot

Since the arm will be 3D printed, it was important to take into consideration a minimization of support material and a maximization of strength for a given weight of plastic. The area moment of inertia for a rectangle is I = (1/3) *b*h^3 so the 3D printed structure would be designed to have a large height especially as the layered lines are printed vertically for convenience, cost minimization, and weight reduction. Timing belts were chosen as the main form of gear reduction. While normal gearboxes are

expensive for high torque stepper motors ($50+) and act with a lot of stress over multiple gears and have backlash, timing belts have low backlashes because they can be tensioned so as not to slip. 8mm bolts were chosen as the pivot/axis of rotation because they allow for easier assembly/maintenance of the robot and 8mm bore ball bearings are one of the most common and cheap types of ball bearings. Standard polylactic acid (PLA) was selected as the main material due to its widespread availability. To get an idea of the effect that each rotation had on the XYZ position of the robot, trigonometric equations were derived as demonstrated in Figure 2.

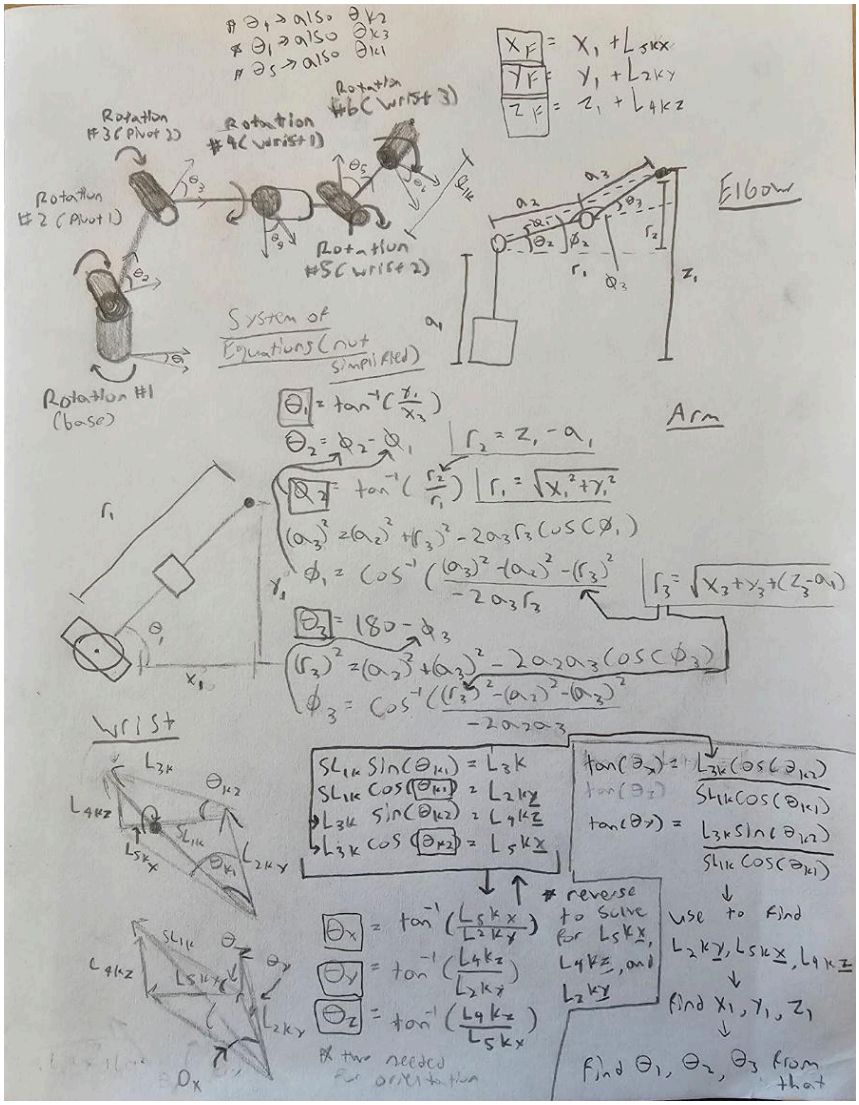

Figure 2: trigonometric equations were derived to understand the relation between joint angle rotations and the end effector XYZ position and orientation.

Based off this Inverse Kinematics (IK) model, the configuration of the robot would be valid to incorporate the following into the design: [Revolute, Pivot, Pivot, Revolute, Pivot, Revolute] where revolute is revolving around the relative Z axis and pivot is rotating about the relative X axis.

Electronics

An Arduino was initially chosen as the microcontroller that would be compatible with the JetsonNano computer board because of its ROSSerial compatibility. 12-bit AS5600 induction encoders that use an I2C bus were chosen as absolute angle encoders that would be mounted to the side of the pivoting bolt (not on the motor incase the timing belt slipped). Because these encoders all

have the same I2C address (0x36), the TCA9548 I2C multiplexer was used. The 3 Nm Nema 23 stepper motors were chosen because they produced the most torque for the effective price and weight (Figure 1) and would be wired to a DM5423 stepper driver that allowed for the stepper to reach its peak current. The 59 Ncm stepper was chosen because it is lightweight and the 16 Ncm stepper was chosen because a bigger stepper would be unnecessarily powerful for the last rotating joint. Both motors would be powered by a TB6600 stepper driver.

Detailed Design

The detailed design can be split into 3 sections: CAD Design, final schematic design, and programming/coding outline.

CAD Design

The CAD design was based on preliminary sketches of motor placement, mechanical timing belts, and ability to be 3D printed. Before the shell was modeled, the timing belt pulleys were designed. To design the timing pulleys, the following calculations were done to 3D model large 80-tooth 5M pitch pulleys that correlated with the 15-tooth 5M pitch aluminum stepper motor pulleys:

1) Pitch Diameter = (Pitch*Teeth)
2) W.A.=360/T
3) ID=P.D-(L1-L2)/2
4) P.D = (5M * 80 teeth)*3.14
5) ID = P.D - (3.8-2.06)/2
7) 0.1mm offset due to 3D printing tolerances

The base was designed with a 127-tooth rotary timing pulley so that only a small, less expensive 1.2 Nm Nema 23 motors could be used. The rotary plate above the base would be attached via an 8mm bolt and thrust bearing created by using the ball bearings from one of the 22mm OD 8mm ID bearings and two washers from the bolt which would then be capped by the hex nut. Additionally, it would ride on 16mm OD 5mm ID ball bearings placed beneath it to reduce friction. All this could be mounted to a scrap wooden board on which weights could be placed. For the first stage pivot mounted on top of the base, it used double bolts that penetrate a flange collar and conveniently has a space for an Allen key to be used. The shell was designed so that belt tensioners could be easily mounted, the 1st pivot rotation could be 120 degrees from a standing position, and the motors could be easily mounted and removed. The second pivoting stage was created by making thick side plates to increase strength along with another staging. These pivoting sections were created modularly so that any one part could be replaced with a different, customized part. Additionally, all parts were designed to be printed vertically (layer lines parallel to base) by considering the area moments. Weight reduction methods include bores in flat rotary pieces and placing motors back to move the CM closer to pivots.

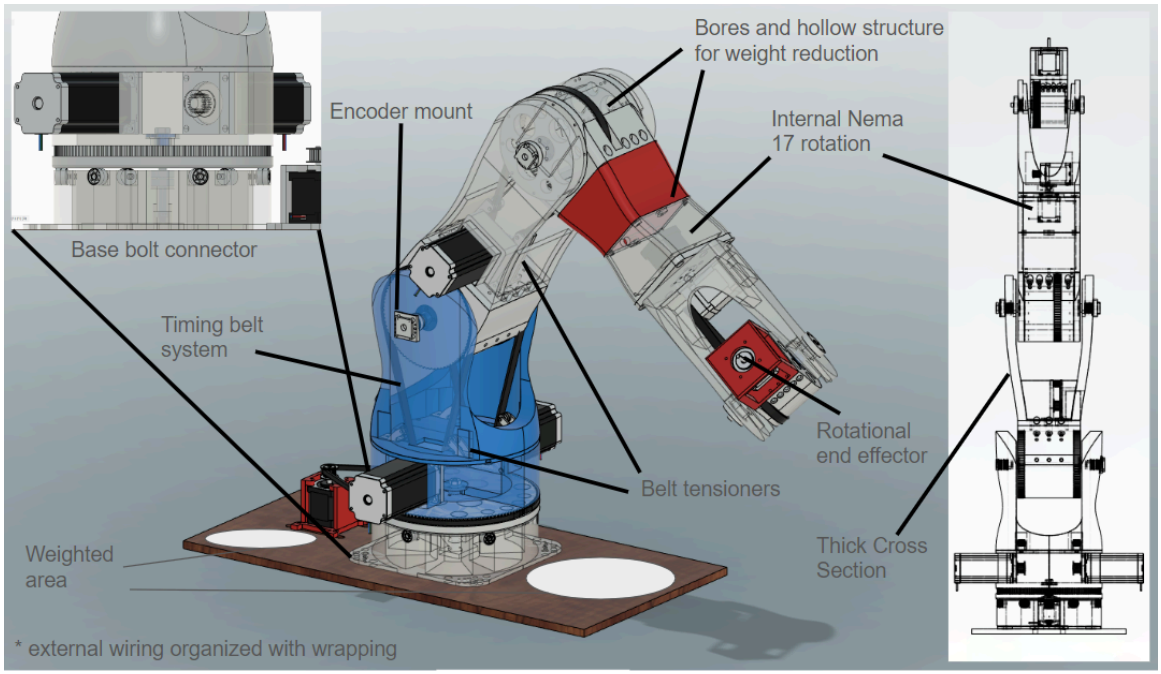

Figure 3: Complete CAD model for simulation and 3D printing.

Structural simulations were conducted to view the weak points in the structure to confirm proper infill gradients on the different arm stages (Figure 4).

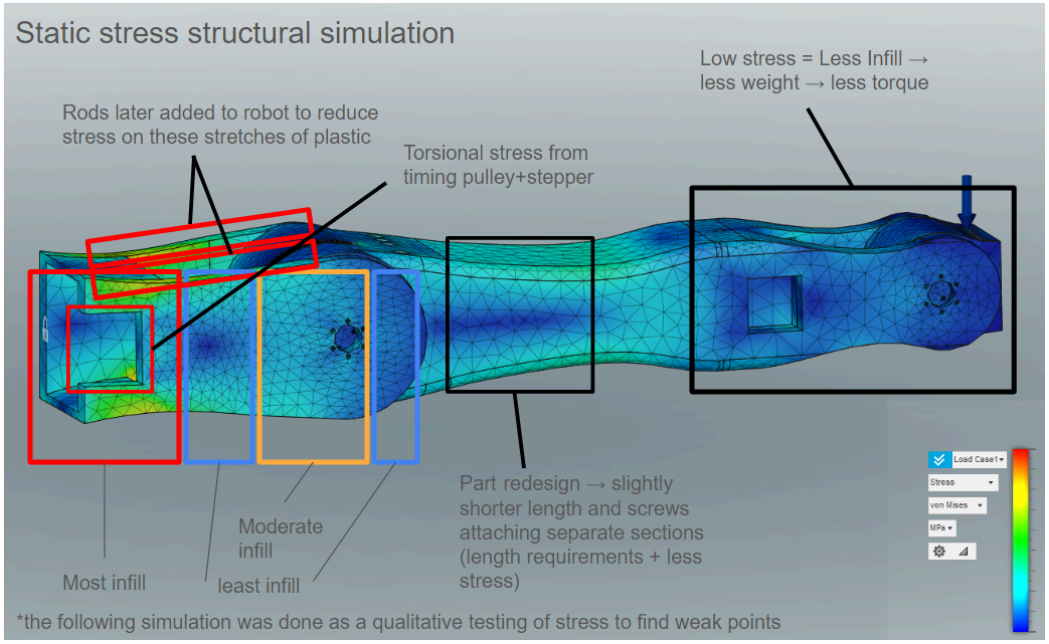

Figure 4: Static stress simulation one of the first iterations of the arm design.

Gyroid infill patterns were chosen due to a combination of torsional load from the motor and load from the weight of the arm meaning the stress was acting in multiple directions.

Final Schematic Design

The schematic design is an iterative process that needs to be tested as a larger scale of electronic connection is achieved: therefore, each subsection of the electronics was tested thoroughly. The stepper drivers were controlled via a PWM signal that outputs a certain number of revolutions pulses per revolution (PPR). The PPR on the stepper drivers was set relatively center at around 6400 PPR to give the Arduino a great amount of control on the speed while keeping the micro steps of the motor smooth and not induce any vibrations. The stepper drivers would be connected to their respective motors, however the dual-stepper drive that operates the first pivoting point would need to be rotating in opposite directions as they are on opposite sides. Even though this could be fixed in code by setting the direction opposite to the other motor, it was better to reverse the coils on one of the stepper drivers while programing the motors with the same direction (DIR) signal (+5v/0v) to reduce the catastrophic chance of accidentally programming the motors to rotate opposite in respect to the robot which would rip apart the 1st rotational tube and damage other unexpecting parts of the robot. Since the encoder wire will not return any signal if their wires are unplugged, the PWM signals would be paused to the stepper drivers and hold the motors at a standstill preventing any damage. All stepper drivers were powered via 31V from an adjustable power supply but can take any voltage from 18V-42V and will consume around 31V * 2.9A = 89.9W powered while not moving and around 31V * 3.5A = 108.5W while moving. I2C communication is meant to be used for small distances (50cm - 2m) depending on how fast the clock cycle is running. While initially attempting to wire the I2C, it didn't work for the encoders attached at longer distances, and when it did, the EMF from the stepper motors produced significant interference. This meant the internal pull-up resistors of the microcontroller were too weak and that a steady 5V to external pull-ups was needed to connect to the I2C to drive the signal up properly at a high clock speed so ROSSerial could also stay in sync. The following calculations could be used to find the I2C proper pull-up resistance value range:

1)Rp (MAX) = Tr/0.8473*Cbus, Rp (MAX) = 300*10^-9s/(0.8473*40pF/ft*~4.5ft)

2)Rp (MIN) = Vcc - Vol(max)/Iol, Rp (MIN) = (5v - 0.4v(max))/0.003a (current through I2C Arduino)

This defined Rp (MAX) as around 1.9k ohms and Rp (MIN) as around 1.5k ohms for RP (min). This gave an idea of how the more resistance should be in place for the pull-up resistors and after some tuning, it turns out that the resistors worked at values of 2K and 1K, depending on their corresponding wire length. The TCA9548A can easily shift the from each encoder I2C by shifting a single bit left in the data type, while the AS5600 can send and receive data on the 0x36 7-bit address registrar and compile the angle by writing the 1st 7-bits of the angle via one request on 0x0F or 0x0D, and the next 7-bits of the angle on the next request on 0x0C or 0x0E compiling into a 12-bit angle request. Magnetic presence can also be detected for debugging purposes through requesting from 0x0B. The 5V supply was enough to power all the I2C communication and the ground was connected to a breakout breadboard for the purpose of wire management. The completed schematic for this circuit can be shown in Figure 5.

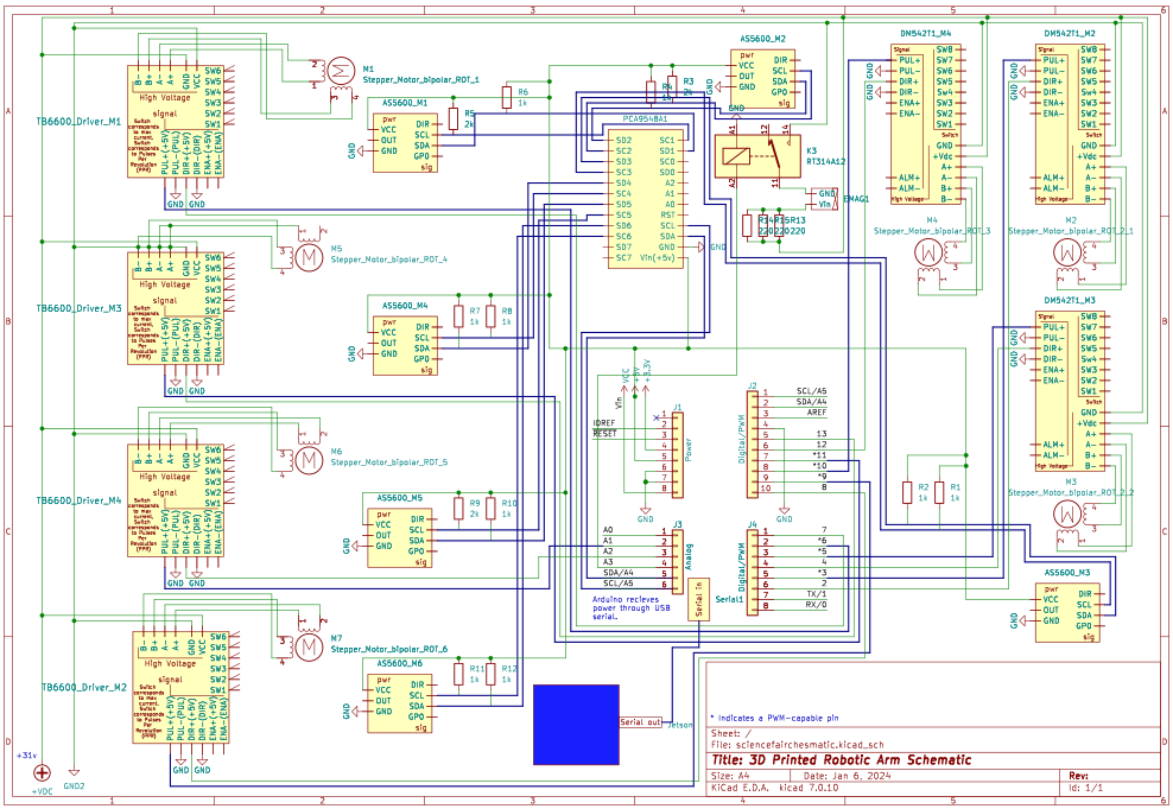

Figure 5: Complete schematic for the wiring of the robotic arm

Coding Outline

Since ROSSerial was to be used to communicate between a low-level microcontroller (Arduino) and a higher-level functioning computer like the Jetson Nano on ROS, the code was split into two main sections: scripts for the Arduino and scripts for ROS on ubuntu-based Jetson Nano. On the main computer, the higher level programs like those running pre-programed tool paths send end effector XYZ position and orientation to ROS which is then converted to individual joint angles using an Inverse Kinematics solver. These individual joint angles are sent to the Arduino through the ROSSerial protocol . On the arduino, the updated individual joint angles are constantly being attempted to be reached through a custom program that reads the encoder position and updates the stepper motor speed accordingly. Each angle would be updated a millisecond or so from ROS to vary the arm speed as delta theta/T would be the velocity of each angle. This can be thought of as similar to a derivative of a function where is the current angle, with a set angle being time T away, if broken up into thousands of points would give the individual angle positions to reach the desired position with a specific velocity or double prime (acceleration). This PID control could be done through ROS and Arduino could receive the raw data. To do so, the Arduino needed to be able to take a set of positions slightly different from the previous and update them real time on the motors (Figure 6).

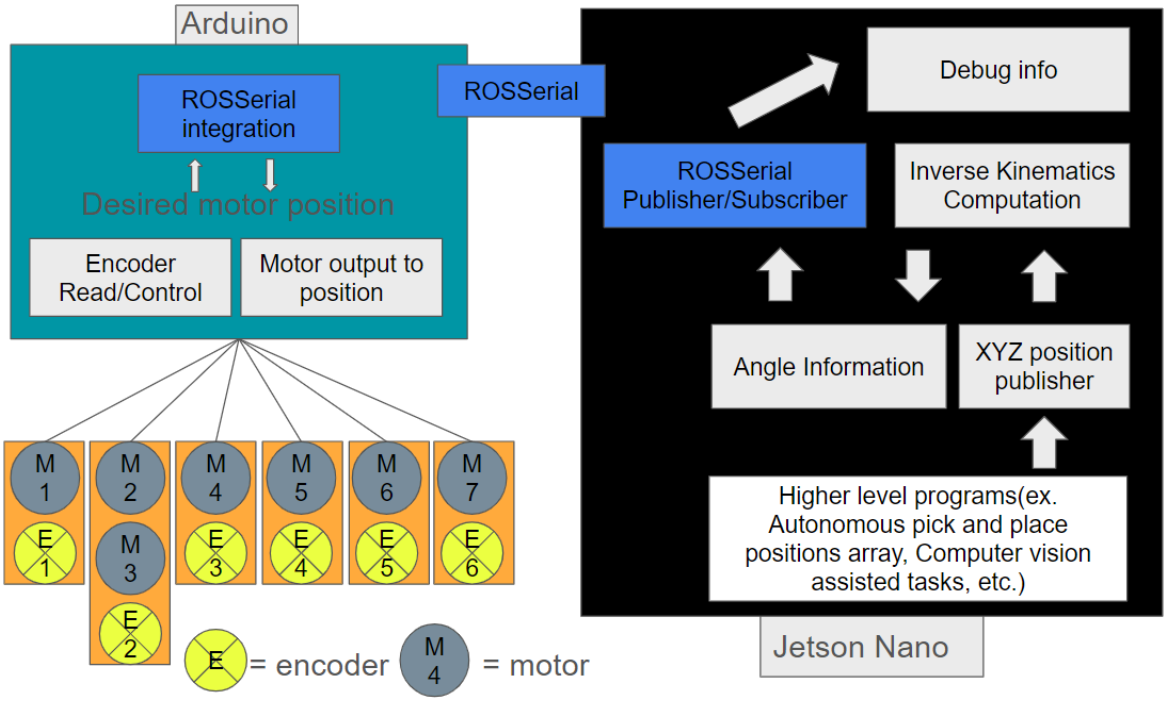

Figure 6: The outline of communication between different vial functions, controllers, and electronics

To accomplish this, code was written to accomplish the following tasks in order:

1) Initialize ROSSerial and I2C Wire library (AS5600 communication)

2) Measure the current encoder angles

3) All at once, move the motors to their respective desired angles through PWM signaling while continuing to check for an updated angle from ROSSerial

The following can demonstrate this process in greater detail:

1) Definition of the ROS library, Wire library, and ROS message types.

2) Defining PWM and DIR output IO pins, SCL and SDA pins, Zero-ed encoder angles, Arduino array of angle goals.

3) Definition of ROSSerial upon chatter from Jetson Nano.

4) Call Back function (Spin Node Handle) that updates angle goals to the most recent ROS imputed angles.

5) TCA9548A that allows you to change which I2C encoder is being accessed.

6) Void Setup that initializes the ROSSerial node publishers and subscribers, the I2C communication, and baud rate.

7) Loop function that checks for incoming ROS information and turns all motors.

8) Defines an I2C function that checks the angle of one encoder.

9) Defines Update angles function that checks each individual encoder by using the I2C function and TCA9548A.

10) Turn all motors function that does the following:

- initializes all pins set for outputting signals.
- sets the true angle goal to be the zeroed angle plus the desired angle change.
- establishes a while loop that keeps updating all angles until they reach the angle goal.

- checks the value of each encoder along and will send a pulse to all motors that need to be moved along with their correct direction.
- updates the time delay between pulses to control the speed of the motors so they could be close to if not synced.

While the trigonometric inverse kinematic(IK) equations may suffice for the estimation of certain specific angles and aid in better understanding the factors that should be accounted for when designing the robotic arm, they are hard to implement code wise and are restricted in their usage and reliability as multi-solutions answers occur (elbow up/elbow down) and become increasingly complex when solving for limited angles, velocities, and null-space exploration. MoveIt, a popular ROS-based toolbox for programming, analyzing and interacting with robotic manipulators was used for IK solving through the creation of a URDF file and solver. The MoveIt plugin for the ROS 3D visualization application, RViz, was also used, which also allows you to publish and subscribe to topics that give position, orientation, and rotation values. To utilize MoveIt, ROS was installed on the Jetson Nano and a catkin workspace was created with the sourcing code being directly put into the setup.bash file for convenience. To solve IK, the 3D CAD model of the robotic arm, which was originally used to design and build the arm, was also used directly to create a Unified Robotics Description Format(URDF) file that could be used by ROS/MoveIt for the creation of the IK computer model. The 3D CAD model was simplified into a model with components for each rotational piece with their own individual simulated joints that allowed a conversion plugin (Kitamura, et al, 2021) to create a .xacro file (file needed to be in mm). The ROS command “rosrun xacro xacro.py file_name.xacro > file_name.urdf” allowed for the URDF conversion and the MoveIt Setup Assistant created the kinematic model of the robot with the specified KDL kinematics solver, joint limitations through collision, RRT robot motion planning, and predefined joint states. From there, roslaunch can be used to launch the demo.launch file that opens an RViz model of the robot. From the RViz model, you can move the XYZ/orientation position around in the GUI or publish to the XYZ position (topic “/rviz_moveit_motion_planning_display/robot_interaction_interactive_marker_topic/feedback”) of the end effector from MoveIt.

To utilize the MoveIt application and its integration in RViz, 4 main nodes (or scripts) were added to the program: ROSSerial, Inverse_Kinematics_Node, Position_Publisher, and info_bac_subscriber (shown in figure 7).

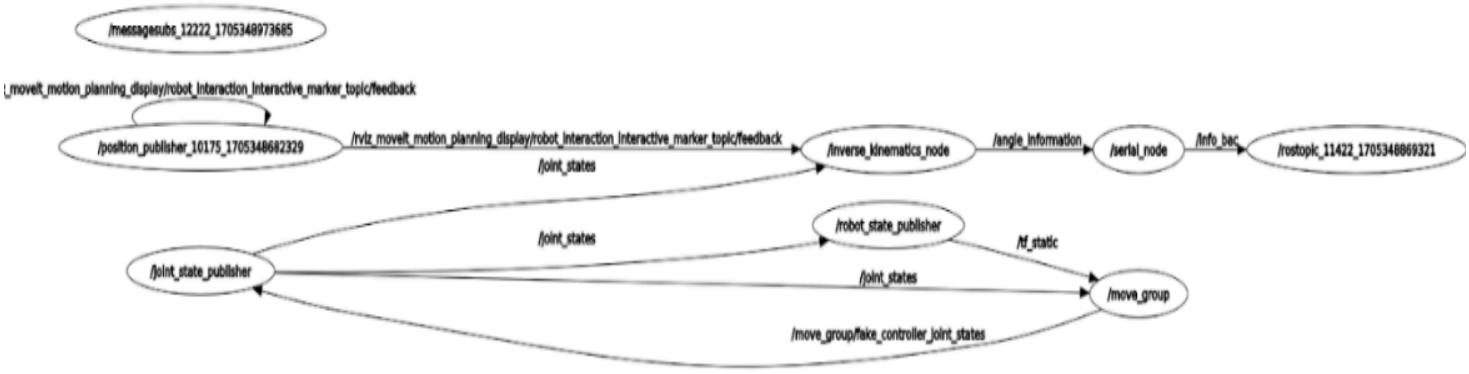

Figure 7: All the ROS nodes that communicate to find and publish the desired angle positions.

The Inverse_Kinematics_Node uses the moveit_messages service “GetPositionIK” to execute an inverse kinematics computation for a specific XYZ position. The XYZ position is taken by subscribing to the topic that gives the XYZ pos in RViz and is then pushed

through the message “GetPositionIKRequest” with the most recent joint state position (which makes IK computation more efficient and accurate as the point being solved for is close to the last point) along with the name and `avoid collisions` set to true. That message is then pushed to the computeIK service the MoveIt provides and returns the solution in which the motor joint state positions can be extracted from. From there, the joint angles would be multiplied by 57.295 to convert them to degrees and then publish them in a joint_state message to the angle_information topic where the ROSSerial node will subscribe and receive the Joint_states. On the Arduino side, the joint_states would be published back through the info_bac topic (also used joint_state message) which then allows the info_bac_subscriber node to print the angles the Arduino gets for debugging purposes.

Construction, Coding, and Debugging

Like the detailed design, the robot construction was split into 3 major sections: Mechanical parts, electronics and wiring, and programming.

Mechanical Parts

To begin the construction of the robot, the 3D printed parts came first. Using the Cura slicer, I was able to fine tune the infill similar to the stress shown in Figure 4 to minimize the amount of material used. To reach a comparable infill percentage, small, applied force tests were performed to test for part deformatting, cracking, and/or snapping in half. Surprisingly, for the first pivoting stage which hoists the majority of the load, the infill was able to reach a range of 10% to 12% with a 4 line wall thickness at 0.2mm quality with the assistance of a small steel and carbon rod and tape inserted directly into the PLA gyroid infill when the 2-day print snapped in half, because no time or resources were left, but this method works surprisingly great. To give clarity to this issue, it would be advised for future assemblies to print at a slightly higher infill (12%-20%) or add guides for the addition of small inexpensive structural rod supports. Medium-size parts like the second stage pivot on the arm have an infill of around 15%. For other parts that had little effect on the robot’s performance based on their weight, higher infills were chosen, base - ~%20, smaller task specific parts (camera/electromagnet mounts) ~%60, etc. In total, all the parts took about a month of printing (albeit, on and off) with the longest print lasting more than 2 days and over 600 grams of PLA. The total amount of PLA used when considering the broken, mis-printed, mis-tolerance, or small unrelated prints is about 6 kg of PLA. However, you could be in a safe margin to print with some error occurring of around 4.5-5 kg of PLA. Once all the parts were printed, the frame was assembled. Using the M2, M3, M4, and M5 screws for different sized parts and modular components, the raw lengths were completed. The 16mm ball bearings were attached to the base. Additionally, the base was connected to the initial rotary plate by inserting 2, 8mm ID 22mm OD ball bearings into the bore in which the 8mm * a 50mm hex bolt was tightened into the shaft flange collar and attached with a hex bolt to the other end with the thrust bearing preventing friction (Figure 3).

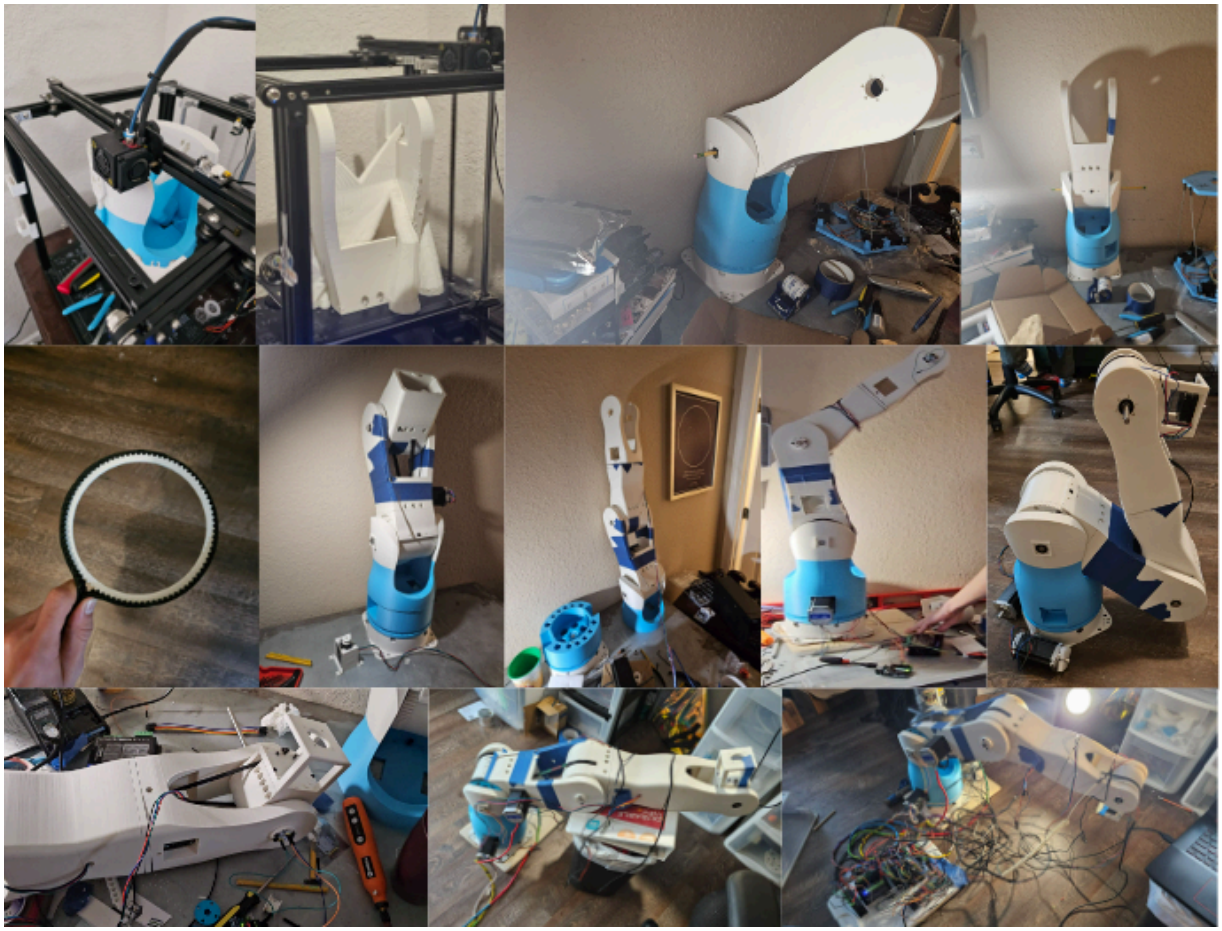

Figure 8: Building process of 3D printing, assembling, and wiring.

At the end of the bolt, the induction magnet (for the encoder) was glued centered onto the end. On the first rotary plate, both stepper motors were loosely attached so that the timing belt could be tensioned by tightening them downwards. For each subsequent rotational pivoting joint, support 8*22 bearings were added to the shell (held in by press fit and 3D printed guides), magnets were attached to the bolts (a steel rod for 5th rotation), timing belts were connected, shaft collars were tightened, stepper motors were mounted, and encoder mounts were attached. For the direct drive rotational motors, a 3D print of an AS5600 mount ("AS5600 Magnetic Position Encoder", Curious Scientist) was used to mount the encoder onto the back of the stepper motor where the magnet was glued directly to the shaft. After fully assembling the mechanical parts of the robotic arm, the wiring was completed.

Electronics and Wiring

To retain proper wire management for the robotic arm, spirally cut straws were used to ground I2C wires, stepper wires, and power cables. Additionally, the stationary electronics (stepper driver, TCA9548A, Arduino, and Jetson nano) were mounted on a separate plastic base. To reduce the price of wirings, scrap USB cables were used to extend the lengths of I2C cables but also doubled as wire with a low capacitance as there was plenty of insulation and the diameter was thin. Because some wires were longer than others due to the range of heights on the robotic arm, I2C pull up resistances had to be fine-tuned to get the best signals. The 5v pin was connected to a breadboard where the voltage was distributed and circuitry was connected based on the electronics schematic. All the stepper drivers were wired in parallel and thicker gauge wire was used for the stepper motors. Additionally, a relay-electromagnet pair was added to the system so that the robot would be capable of demonstrating pick and place tasks.

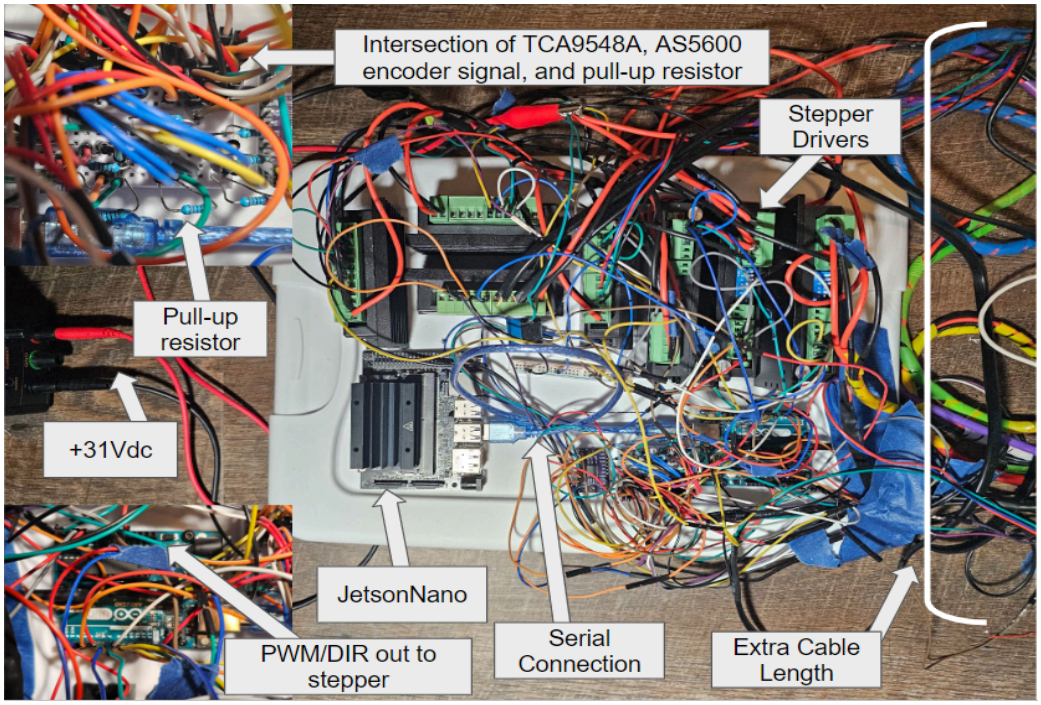

Figure 9: Complete wiring of the robotic arm.

Programming

Once the wiring was finished, coding needed to be implemented to drive the electronics that control the mechanical systems on the robot. Before working on the final, large program, individual parts were tested. The stepper motors, I2C encoders, and TCA3458A were all individually run and thoroughly tested before being tested in a larger grouping of code. Next, intermediate blocks were written to ensure 1 joint could rotate to its proper position, then all 6 at different times. After making sure to know what specific problem might be occurring with certain errors, using the outline from the Detailed design phase, The code for the Arduino was written and compiled together (all code can be viewed on GitHub (Lucci, 2024)). For the ROS programming, the outline, once again from the detailed design was compiled and ran successfully on its own, just like the Arduino code. However, a major problem occurred when attempting to bridge the gap of communication over ROSSerial: DRAM. The dynamic memory of the Arduino was too little to use both ROSSerial and the wire communication library from the I2C. This was determined when the first rotational value of the Joint States messages was being published properly, but the rest of the values (angles 1-6) were returned as 32-bit integers that were almost zero. To mitigate this program, a more compact message, Float64MutliArray, was attempted to be implemented. However, like the Joint States message, there were still 4 64bit Floats being returned at almost zero due to the same DRAM issues with the Arduino. Eventually, it was decided to send individual floats over the ROSSerial communication so that they could be properly processed without being affected by the low DRAM of the Arduino. Once communication between ROS and ROSSerial, the code on the Arduino needed to update fast enough so multiple Rospy spins were called during the Update angle function.

Once the communication between ROS and Arduino was successfully established, the procedure to establish the payload, adaptability, repeatability, and accuracy of the standard robotic arm. The first three tests would be adaptability, repeatability, and accuracy in case the arm broke during the payload and speed testing. For the repeatability testing, the arm would be set to a random position, the motors powered off, moved back to a standstill with the joints at a different position than the last, and then turned back on to see how close or repeatable it was to the target position. This procedure would be repeated 5 times for 6 random positions of significantly different orientation measuring the XYZ position deviation to reduce experimental error and get accurate measurements

that encompass all of the robots movement. The Z distance could be observed by using known thin rigid objects to find the distance between the end effector and the table. The X and Y values were measured using calipers. The total repeatability distance would be found by sqrt(x^2 + y^2 + z^2). The accuracy would be tested by setting the robot to the standing position, moving it to a desired position in RViz (translated to real robotic arm), measuring its position from the start, and then measuring how much distance the movement along one axis compares in the real-world vs the computer application. This would be repeated for 6 different lengths on 3 different positions to reduce experimental uncertainty and get a good insight to the accuracy of the robotic arm. The adaptability was tested by using different modified parts with different lengths to demonstrate the adaptability. The payload testing would go to a point of failure of motors or structure for 8 different lengths up to 800mm.

Data/Results/Observation

Testing round 1 [prior to final]

Repeatability

When data was collected for repeatability, multiple issues arose. Firstly, if too much pressure was applied to the base, the base rotation would change causing an outlier in the repeatability data. After some observation, it was noted that the base was tilting forward because the shaft/bolt connecting the rotary mechanism was flexing. Because the induction magnet whose position is read by the encoder was tilting with the bolt, even though the flexing was not substantial (<1 degree), it was enough to alter the orientation of the magnetic field enough to change the encoder position reading (the encoder was mounted separately and not flexing) and move the base. Even though a majority of the time this problem self-corrected or did not occur, future designs will implement a guard rail like that on a roller coaster track to prevent future flexing causing unreliability.

The next issue occurred when after the 3rd repeatability test was performed and maintenance was being done, the robot fell while standing and cracked at the weakest point (middle). This called for tighter belt-tensioning and a closer inspection to the mechanical components. In doing so, another crucial issue was discovered: the 4th direct drive stepper motor had a loose flange shaft collar and bolts connecting it which caused it to shift orientation every time the position was reset. Additionally, it was discovered that a couple shaft flanges connecting to the bolt were also slowly shifting due to improper shaft collar tension. Once these issues were resolved, the repeatability became much more promising. Before the arm's mechanical issues were solved, the repeatability was over 2mm, however when mechanical issues were (for the most part solved), the average repeatability of the last 3 positions came in at the goal of less than 2mm. In the data, it can be observed that strings of 2 or 3 trials, in the same axis the results were very close to each other until another trial was performed. This could mean a couple of things: shifting encoders, small changes in bolt positioning, or minute shifts in the robot's baseplate. For future trials, to ensure an even greater repeatability, the base was mounted down more firmly heavier weights and more duct tape straps, encoders were mounted more firmly with a greater number of screws, and the bolt-shaft flange connection received another worm screw tightener. Additionally, over a long-time span (+20 minutes) of leaving the robot on, heating problems began to occur with the motors so the robots' motors would need to be periodically powered off. Too much heat from the motors could deform the plastic and loosen the timing belts, which result in worse performance. Heat sinks will soon be added to the stepper motors to increase cooling.

Accuracy tests were performed by connecting ROSSerial to the robot and actively moving and measuring the position of the robot in real-life based on the RViz model. The debugging process included revising angles (resulted in increments of 3 cm instead of 1 cm), tuning the sync between ROS and Arduino over the serial, and revising the code to be conservative with the low Arduino

DRAM. Once the issues were fixed and the robot was properly zero-ed (stand position), The accuracy was able to be tested. For the first test, increments of 1 cm were almost exact as well as a whole increment of 10 cm. The exact position was difficult to measure but will be done so as this is ongoing research. Two more positions need to be tested before accurate conclusions can be drawn on the accuracy, but so far, the results look promising. When attempting to trace a straight line, the robot was able to do so, but in a slow and shaky manner. This was due to the low position updating time of around 0.1s instead of the preferred 0.001s or less. The low updating time was due to the updating timing from the nh.spinOnce being interfered by delays to send pulses to stepper motors and delays in transmission with I2C encoders.

The payload was tested by increments. The arm was successfully able to lift around 0.8lbs or 0.36kg at 800mm when the belts were not tightened. With manual intervention by squeezing the timing belts, the payload was able to work at 1.9lbs or 0.86kg at 800mm, have a functional limit at around 3.22lbs or 1.45kg, and stalled and began cracking after 1.45kg. As the research is still ongoing, smaller incremental data is being gathered for weights at 800mm but also for weights at 700mm, 600mm, etc. However, because the limiting factor was timing belt tensioning, I believe that with a new belt tensioner (currently underway as research is ongoing), the arm will be able to easily surpass the 1kg goal by itself.

The robot has successfully demonstrated its adaptability in sections of the design. The middle section length was able to be replaced by another part after it broke, and it was also able to be modified to a shorter length for smaller workspace requirements. Additionally, when parts were designed or printed incorrectly, they were able to be swapped out for better parts: timing pulleys and rotational tubes. Finally, the modular like design demonstrated how almost any part could be remixed for specific tasks.

		Repeatability						**Accuracy**	
	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5			**Position #1**	*1cm increments along X
Position #1						ROS Length (X)	X distance	Y distance	Z distance
X	5.01mm	4.04mm	3.17mm	1.32mm	1.36mm	0cm	0cm	0mm	17.13mm (from stand)
Y	1.84mm	1.99mm	2.07mm	4.07mm	2.51mm				
Z	0.08mm	0.16mm	2.18mm	1.28mm	1.84mm	1cm	0.961 cm	line established	25.12mm (from stand)
Total Distance	5.34mm	4.51mm	4.37mm	4.47mm	3.39mm				
Position #2						2cm	1.058cm	1.51mm	27.76mm (from stand)
X	1.26mm	1.15mm	1.60mm	1.97mm	0.18mm				
Y	0.49mm	1.29mm	2.83mm	3.02mm	4.02mm	3cm	0.968cm	1.53mm	28.25mm (from stand)
Z	1.30mm	2.98mm	3.30mm	1.30mm	1.30mm				
Total Distance	1.88mm	3.44mm	4.63mm	3.83mm	4.23mm	4cm	0.974mm	1.71mm	27.10mm (from stand)
Position #3									
X	0.65mm	5.49mm	5.56mm	1.51mm	6.44mm				
Y	2.05mm	1.82mm	2.68mm	3.75mm	1.09mm	5cm	0.942mm	1.40mm	26.95mm (from stand)
Z	0mm	1.20mm	1.20mm	0.32mm	0.48mm				
Total Distance	2.15mm	5.91mm	6.29mm	4.06mm	6.55mm				
----------	arm snapped, middle section rebuilt + belts & screws properly tensioned					6cm	0.907mm	0.85mm	26.29mm (from stand)
Position #4									
X	0.20mm	0.51mm	0.41mm	0.50mm	0.15mm	10cm	4.395mm	2.11mm	18.40mm (from stand)
Y	0.40mm	0.84mm	0.61mm	1.02mm	1.36mm				
Z	0.08mm	0.24mm	0.56mm	0.48mm	0mm		total (added):10.205cm		Absolute XYZ pos not measured yet
Total Distance	0.45mm	1.01mm	0.92mm	1.23mm	1.37mm				
Position #5							total (merasured): 10.338cm		
X	0.39mm	0.69mm	0.74mm	0.66mm	0.66mm		measuring accuracy: 0.133cm		
Y	1.48mm	0mm	0.16mm	1.28mm	1.36mm		Percent error (vs ROS positioning): 2.72% error		
Z	0.28mm	0.46mm	0.57mm	0.81mm	1.27mm				
Total Distance	1.56mm	0.83mm	0.95mm	1.65mm	1.97mm				
Position #6	*ros position running in background for test 6								
X	0.26mm	0.95mm	0.57mm	0.35mm	0.51mm				
Y	0.48mm	0.31mm	0.94mm	1.15mm	0.96mm				
Z	0.08mm	0.16mm	0.24mm	0.16mm	0mm				
Total Distance	0.55mm	1.03mm	1.13mm	1.21mm	1.04mm				
*X,Y, and Z are absolute values are deviations from the center point					Average repeatability after belts tensioned =			1.13mm	
0mm = too small to measure									

Figure 10: Data collected on repeatability positions and 1 accuracy positioning.

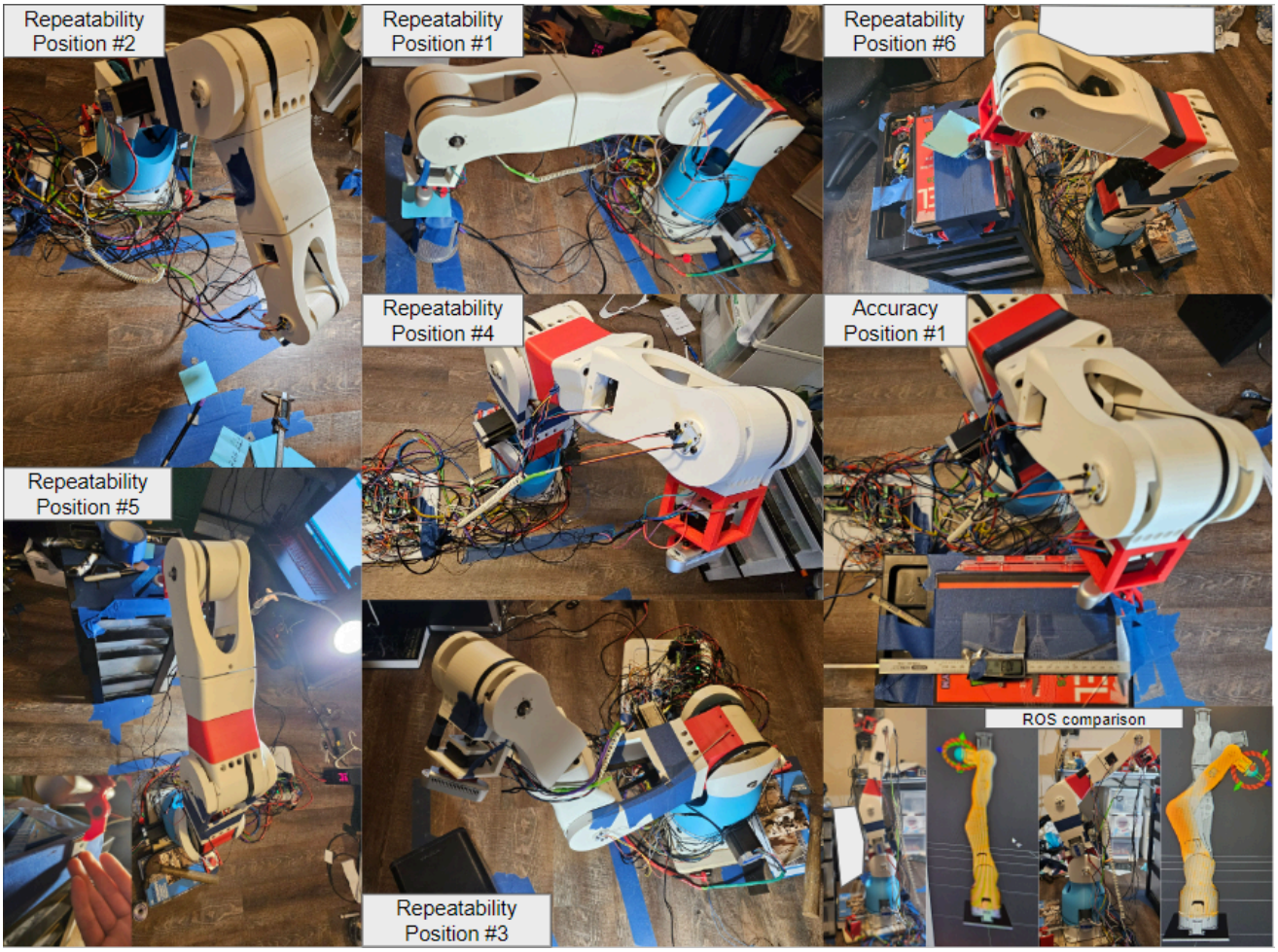

Figure 11: Different positions tested for repeatability and one position tested for accuracy.

Conclusion Round 1 [Prior to Final]

From the current testing, the conclusion can be drawn about the repeatability successfully reaching its goal of under 2mm, the price under $800, ROS integration, and adaptability. The accuracy, although currently under 1cm for smaller increments with the current testing so far, more data needs to be collected regarding the total position from the origin and more positions to confirm the accuracy is under 1cm. The payload, although currently limited by the belt tensioners to around 0.36kg and with forceful tensioning of around 1.45kg, is predicted to achieve over the goal of 1kg with new belt tensioners that will be added promptly. As for the speed, data still needs to be collected but so far has been relatively slow for initial testing but has the theoretical capability of reaching higher speeds.

Testing Round 2

The continued testing consisted of mainly validating the accuracy, adaptability, payload, and practical application. For practical application testing, the line trace and pick-and-place test were chosen because they encompass general robotics movements and demonstrate the possibility for the robot to complete a broad range of tasks.

The full testing for the accuracy went as follows:

1) The robot was moved to its initial position where it is zeroed.
2) The robot's position was zeroed by measuring the initial x, y, and z offsets and then those positions were chosen as the positions that would be considered zero in the RVis visualization or way of inputting xyz signal into the robot.
3) The accuracy was measured by moving the arm by increments of 1 cm, 2cm, 5cm, or 10cm and measuring the real-world translation along the specific defined axis. The deviation between these two positions is what is considered the accuracy. This was repeated for >16 different position increments and 3 different robot configurations to reduce experimental uncertainty. Each configuration traced a different axis to ensure inclusive results.

The full testing for the payload went as follows:

1) Measure a specific length of 800mm, 600mm, 400mm, and 200mm where the maximum payload will be tested on.
2) For each length, progressively overload the end effector with weights until the robot's performance is beginning to be compromised by a significant factor and record that weight. Repeat this for all lengths.
3) Additionally, more specific payload versus performance was conducted at the distance of 800mm to get a better idea of the maximum payload capacity.

The full testing for practical application goes as follows:

Line trace:

1) Measure the location of the relative X, Y, or Z axis in reality compared to the location in the RVis manipulator.
2) Increment the robots position along this axis to find how deviant the "line trace" will be.
3) Create an image to visualize how deviant the line is.

Pick-and-place:

1) Set a small bin with an M4 screw in it on a stable surface and a separate, empty small bin that the screw can be placed in for the pick-and-place transer by the robot.
2) Use the modular system of the robotic arm to easily attach the electromagnet to the robots end effector and connect it to the microcontroller.
3) Using the ROS Serial angle calculating and publishing scripts and the RVis interactive display respectively. Maneuver the robot and to pick up the screw from the initial bin and transfer it to the empty one.
4) Repeat this multiple times in order to catch any flaws or errors with the system that can be worked out.

Adaptability:

1) The adaptability was tested by creating modular parts and attempting to implement them. The adaptability can also be demonstrated through the ability of the robotic arm to be iterated on without obstruction of a design that can't be easily modified.

Results Round 2

Accuracy:

The accuracy of the end effector was an average of 1.33cm achieving only 3mm off from the goal. Possible causes for accuracy include off center induction magnets glued to shafts and a non-zero zeroed standing position. The graphs represent the amount of distance for each axis deviated from a specific ROS location when moving the robot some distance (Figure 12, 13, 14) on an X, Y, or Z axis in ROS (Figure 15).

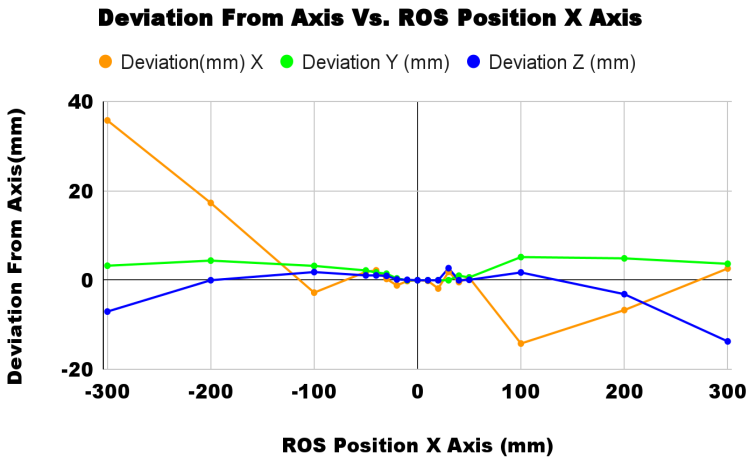

Figure 12: The deviation from the physical, measured axis versus the incremented, virtual ROS position along the Y axis.

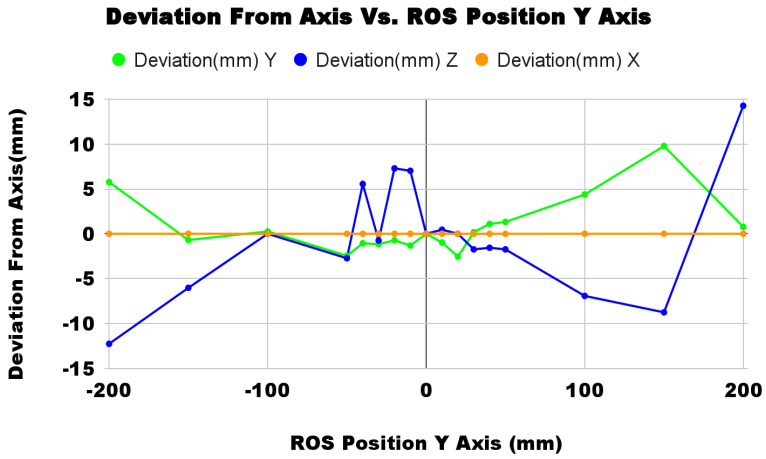

Figure 13: The deviation from the physical, measured axis versus the incremented, virtual ROS position along the Y axis.

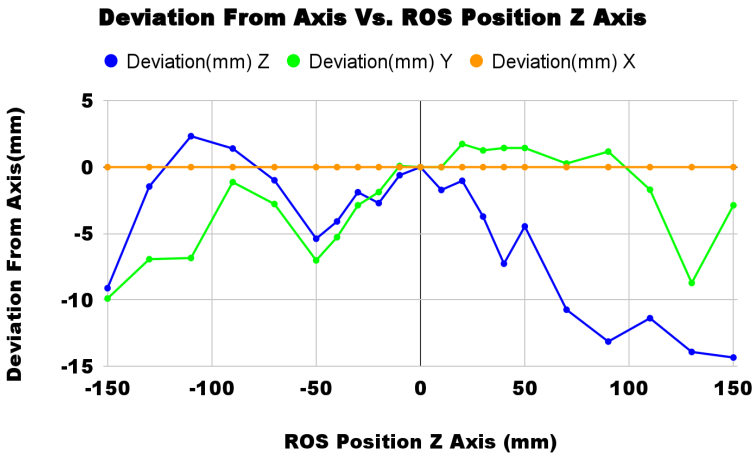

Figure 14: The deviation from the physical, measured axis versus the incremented, virtual ROS position along the Z axis.

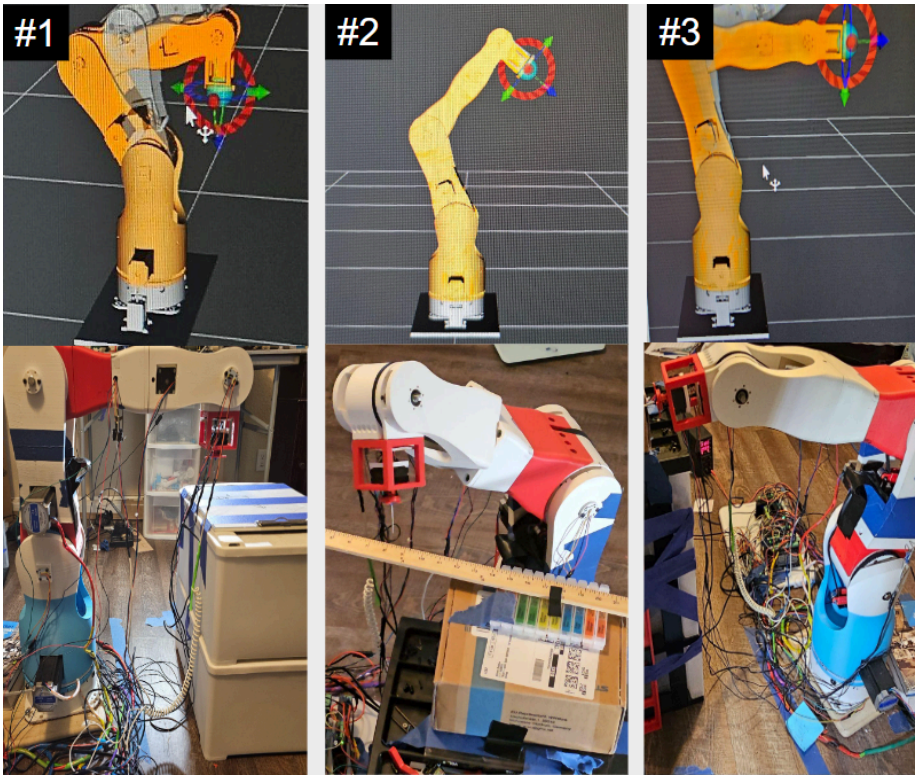

Figure 15: The different accuracy position trials tested on their respective X, Y, and Z axes.

Payload:

At 800mm the robot was able to manipulate 0.64kg effectively while being able to lift or hold heavier weights of up to 1.45kg at the same length with a large performance compromise. Since $F = T/\Delta x$, the graph (Figure 16) demonstrates this relationship. The increase in payload capacity by more than 2x from 0.4m to 0.2m can be explained by the change of CG of the robot due to a different position configuration existing. This graph (Figure 16) can be used by users of the robotic arm to make appropriate adjustments/length adaptations and know exactly what length they need to modify the robotic arm for their desired payload.

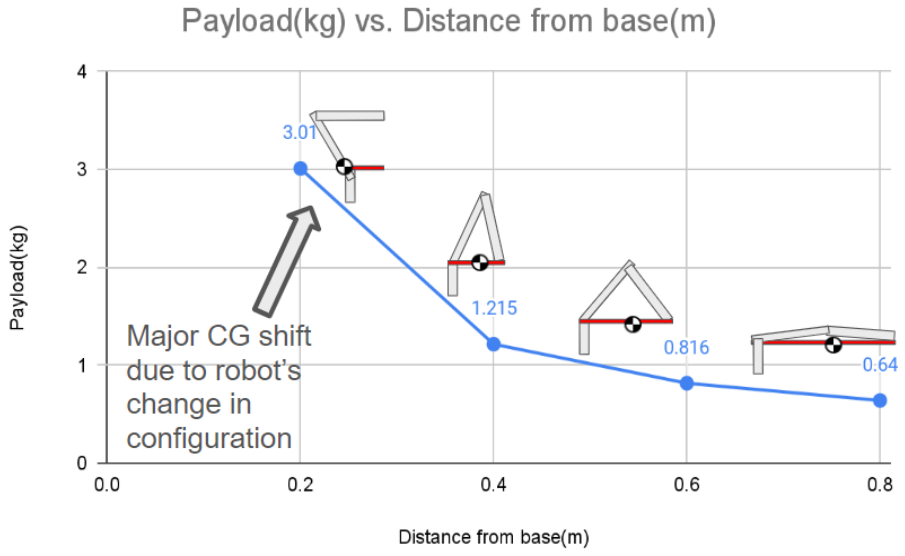

Figure 16: The payload (kg) vs. distance from base (m), displays robot configuration visualizations for each data point.

Practical application:

The robot was successfully able to trace a straight 300mm line (Figure 17) before deviating by a significant amount and running into the physical limits of the robot's maximum reach.

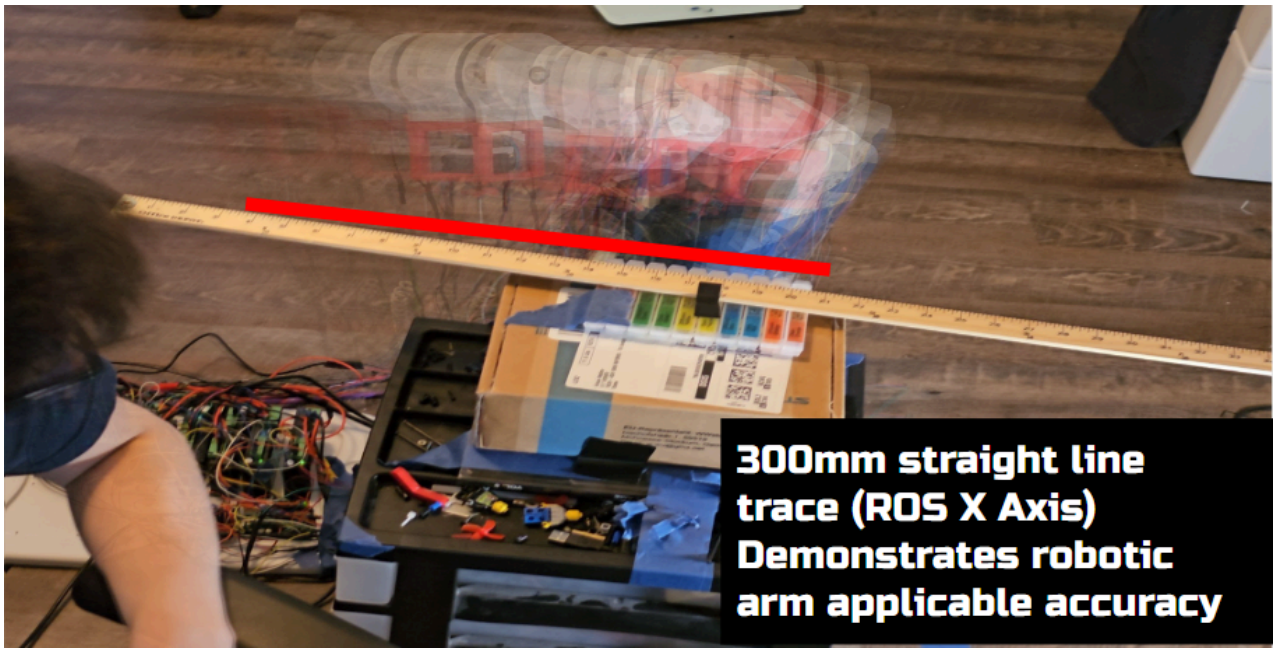

Figure 17: The 300mm straight line trace through the use of live position controlling in ROS.

Additionally, the robot was able to successfully pick-and-place a small M4 screw from one bin into another multiple times (Figure 18) through the live control/interaction with RVis.

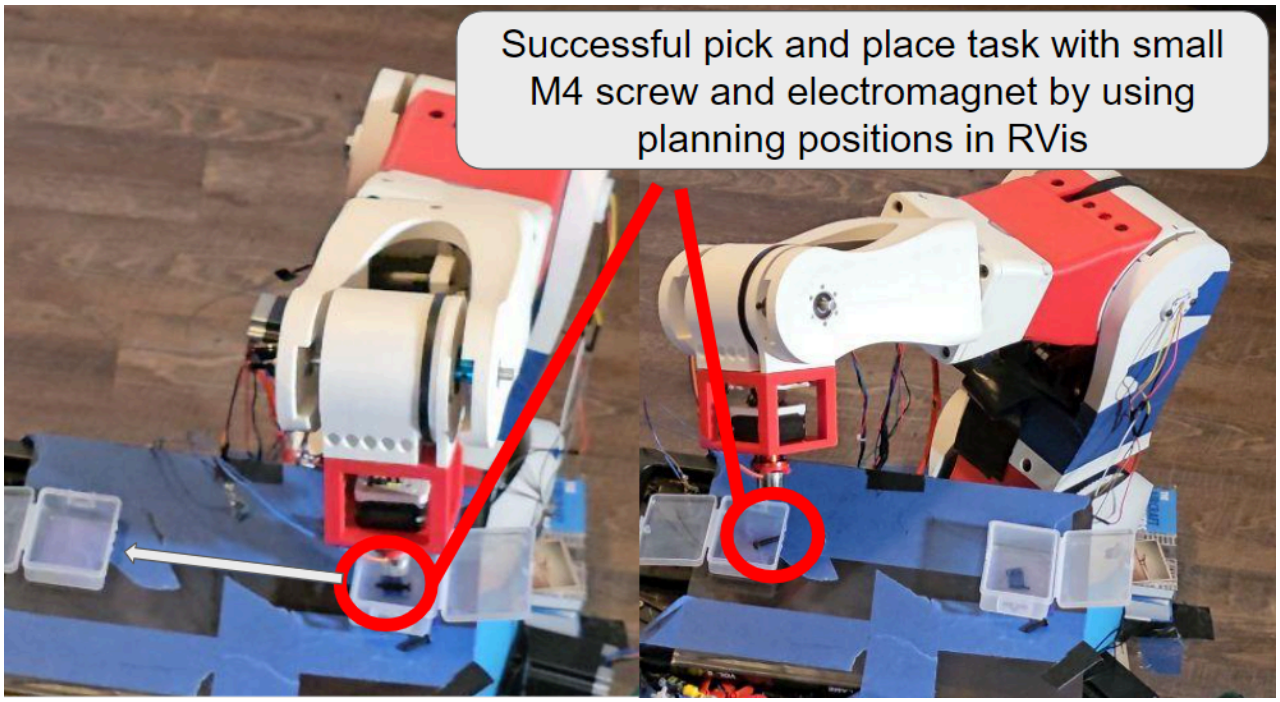

Figure 18: The pick-and-place transfer from bins of a tiny M4 screw using live position controlling in ROS.

Adaptability:

The robot has demonstrated its adaptability in sections of the design (Figure 19). The middle section length was able to be replaced by another part after it broke, and it was also able to be modified to a shorter length for smaller workspace requirements as seen in the figure below. Additionally, when parts were designed or printed incorrectly, they were able to be swapped out for better parts: timing pulleys and rotational tubes. This demonstrates the possibility opened to researchers due to inexpensiveness and customizability of the arm.

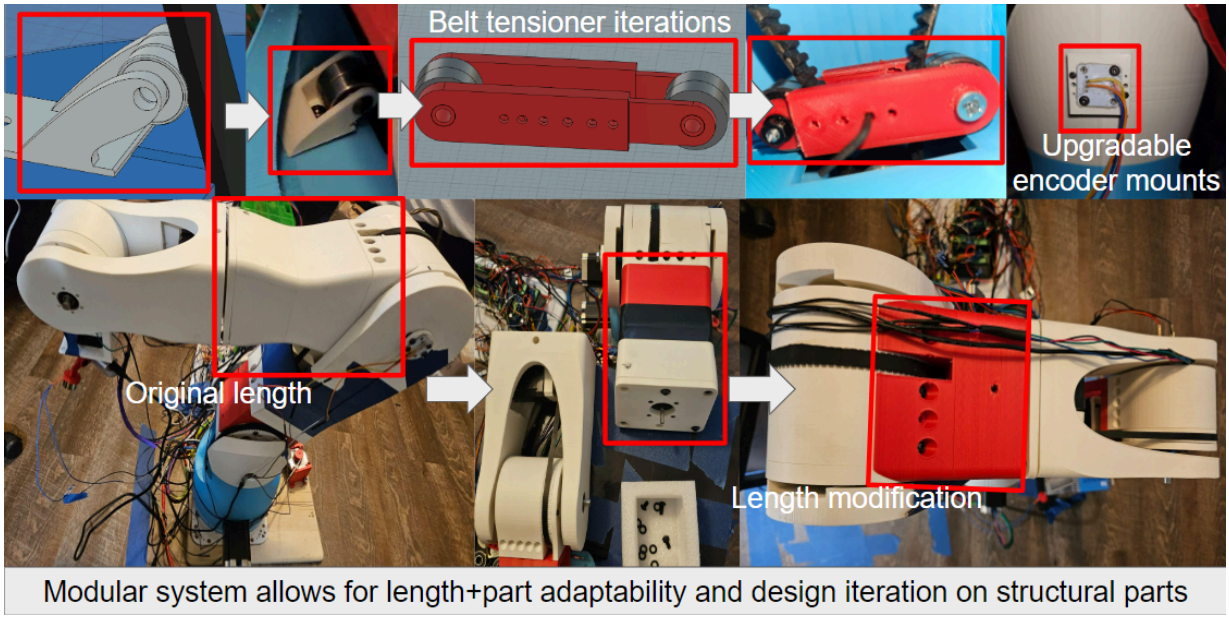

Figure 19: The robot demonstrates its adaptability through its ability to be easily iterated upon and use modular sections.

These tasks demonstrate that the robot has the ability to maintain enough precision, accuracy, controllability to complete most general purpose tasks.

*Data Limitations:

Repeatability and accuracy measurements can deviate by around a 1/10th of a mm (precision of calipers = 0.01mm, paper = 0.06mm (used to measure Z distance from stray base), visual measurement (hard to tell difference within a few 0.01mm between center dot and end effector)).

Final Conclusion

From the current testing, the conclusion can be drawn about the repeatability successfully reaching its goal of under 2mm (1.13mm), the price under $800, ROS integration, and adaptability. The accuracy achieved a range from 0.07cm-3.58cm depending on the robot's configuration with an average of 1.38cm accuracy. Finally, the maximum maneuverable payload was 0.644kg and a maximum holding load of 1.45kg for lifting at 800mm. Due to the general success of the robotic arm's performance, it was made open source for others to build it or guide them through the design of a customizable, inexpensive robotic arm.

Practical Application

Future hobbyists, educators, students, and small institutions can utilize this robot to perform robotics operations, teach principles behind robotic joints and ROS, preform robotics AI research (works with ROS python and can be connected to visions systems), and countless other tasks all for the extremely low price of around $800 without too much compromising of standard industrial robotic parameters.

Future Expansion

To expand this project in the future, I would like to first upgrade the microcontroller to a dual core microcontroller with higher DRAM like the ESP32 so that the angle reading from ROS can be performed on one core while the updating angle is performed on the other. This will allow angles to be updated much faster and move in sync with each other so that position tracing will be much smoother. A greater DRAM would allow you to send full arrays instead of floats that need to be compiled into a separate array on the Arduino. Additionally, a secondary PID tuner on the microcontroller side would be useful to smooth out the tiny adjustments made by the robotic arm; Possibly taking the velocity of each angle joint through ROS as well from a different array and using that for PID control. Additionally, after more thorough testing and modification is performed as needed, I would make the arm open source so that anyone can build it themselves and use it for their intended purposes.

Acknowledgements

This project would not have been possible without my mentor who helped me with technical issues and my mother and grandparents, who fully supported my project in every way.

Bibliography

(1) HTD-5M. (n.d). GPR Industrial.https://www.gprindustrial.com/en/5mm-pitch-5m-timing-belts/141928-1575-5m-26-timing-belt-type-5m-1575mm-pitch-length-26mm-width.html

(2) James, K. (2000, September). Effect of Parasitic Capacitance in Op Amp Circuits. Texas Instruments. https://www.ti.com/lit/an/sloa013a/sloa013a.pdf?ts=1705439688512

(3) AS5600 12-Bit Programmable Contactless Potentiometer. (2018, June 20). OSRAM Group. https://ams.com/documents/20143/36005/AS5600_DS000365_5-00.pdf

(4) MoveIt Setup Assistant. (n.d.). Moveit. https://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html

(5) Kitamura, et al. (2021, September 1). Fusion2URDF. Github. https://github.com/syuntoku14/fusion2urdf

(6) Tully Foote. (2020, March 25). Ubuntu install of ROS Melodic. ROS.org. https://wiki.ros.org/melodic/Installation/Ubuntu

(7) Rajan, Arora. (2015, February). I2C Bus Pullup Resistor Calculation. Texas Instruments. https://www.ti.com/lit/an/slva689/slva689.pdf?ts=1705367071780&ref_url=https%253A%252F%252Fwww.google.com%252F#:~:text=The%20pullup%20resistors%20pull%20the,can%20lead%20to%20signal%20loss

(8) GetPositionIK Service. (2023, September 26). ROS.org. https://docs.ros.org/en/noetic/api/moveit_msgs/html/srv/GetPositionIK.html

(9) AS5600 Magnetic Position Encoder. (March 5). Curious Scientist. https://curiousscientist.tech/blog/as5600-magnetic-position-encoder

(10) Frank Lucci. (2024, January 19). 3DRoboArm. GitHub. https://github.com/piar1243/3DRoboArm/tree/main