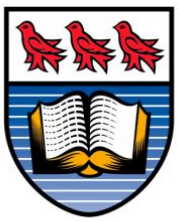
UVic Robotics Club:
Robot Arm Control System

Final Report

ECE 499

Group 9

August 5th, 2019



University of Victoria

Group ID: 9

Faculty supervisor: Dr. Panajotis Agathoklis

Team Information: UVic Robotics Club: Robot Arm Control System

S. No.	Name	V Number
1	Calvin Tierney	V00816283
2	Alex Rudston-Brown	V00797891
3	Alex Kolodinsky	V00831707
4	Amr Talkhan	V00801429
5	Connor Hiliard	V00806634
6	Kyle Cathers	V00851761

Acknowledgments

Our team would like to take the opportunity to thank and appreciate everyone who contributed to the completion of this project. Firstly, the institution, the University of Victoria, for providing us with the knowledge and resources to be able to take on this project. A special thank you to our project supervisor, Dr. Panajotis Agathoklis for being readily available to address our concerns and for his wonderful guidance and encouragement over the course of this project. We would also like to thank the UVic Robotics club for providing us with the equipment that was paramount to the success of this project, in addition to helping us troubleshoot technical issues that arose during this project. In addition, many thanks to Dr. Ilamparithi for his excellent effort in putting together this course and the chair of ECE for funding the project. Overall, the above names provided us with the resources to succeed and learn a great deal throughout the course of this project.

Executive Summary

The goal of this project was to facilitate the use of automatic control on the robotic arm developed by the UVic Robotics Club. Before this project, the arm only implemented manual control using a joystick and struggled with fine control for small movements. The end goal for the Robotics Club is to make the arm perform autonomous object detection, autonomous movement, and the ability to pick up an object without the need for human interaction. This project is a step toward this end goal, by developing the autonomous movement and control of the robotic arm. Thus far, the team has taken several steps to achieve this goal. Research on inverse kinematics was done in order to determine the joint parameters of the arm and to relate them to the position of the end-effector, hence using it to calculate various angles within our setup. PID control was integrated with the arm, the goal of which was to make the arm movements as fine and as smooth as possible. Additionally, the system was integrated on the arduino mega due to the high number of I/Os available on it and its general reliability. Overall, the implementation was successful and allowed us to develop an autonomous robotics solution.

Table of Contents

List of Figures

List of Tables

Glossary

PWM - Pulse Width Modulated signal

Degrees of Freedom - Specified degrees of movement for the robotic arm

Forward Kinematics - Used to determine the position of the end-effector of a robotic arm using specified joint values.

Inverse Kinematics - Used to recover joint parameters from other relevant data.

LiDAR - Light Detection and Ranging

End-effector - The device at the end of the robot, designed to interact with the environment.

I. Introduction

Throughout the world, robotics and AI technologies are becoming more prevalent each year in both commercial and personal settings [1]. Industries that rely on mass production were some of the first to fully realize the capabilities of robotics and how efficiency, quality control and production speed can all be increased rapidly with the new technology. Recently, humanoid robots have made great advances and the potential for growth is still in its beginning stages [1].

Of all the robotic devices currently in operation, a commonly recognized piece of equipment is the robotic arm. Most often possessing a long arm made of different lengths and ending in a “gripper” apparatus, this device has many uses throughout the world and new iterations are invented each year. The UVic Robotics Club is currently designing and building a rover-style robot equipped with a robotic arm currently capable of 3 degrees of motion. Our project will focus on expanding the movement and controls of the robotic arm in order to give the rover increased range and flexibility when picking up items.

There is a growing need for automated technology in a variety of sectors throughout the world [1]. We chose this project in the hope that it will provide a cost effective autonomous robotic arm that can increase productivity in multiple industries around the world. There are many potential uses for our design, from automotive and electronic industries to remote explosives disarmament and hazardous waste disposal.

II. Project Goal

The goal of the project is to develop and test a robot arm control system capable of locating an item based on its unique coordinates, moving autonomously to that object and returning it to a home location.

III. Design Objectives

The current robotic arm for the UVic Robotics Club utilizes manual control to move and pick up objects. The arm has a wide range of motion with minimal fine control, creating difficulties in accuracy and picking up smaller objects. By moving the arm slower, the voltage signal often cuts out, resulting in a “jerking” motion. The manual control process is tedious, as maneuvering the joystick controller takes time to pick up a larger item, and cannot properly pick up a smaller item. Our design objective will allow the robotic arm to move from a set “home” location to another set of specified coordinates, all without the need for a manual controller.

Our work will include designing inverse kinematic functions for the arm, in order to take a point in space and move the arm to that specified spot. The goal of the inverse kinematics is to determine the joint parameters of the arm and hence determine a desired position of the arm’s end-effector. Kinematics analysis is essential to the design of any robotics based solution as it provides the designer information on the position of each aspect of the mechanical system.

We are also implementing a PID control system for arm movement. Our desired parameters are listed in Table 1, and these performance measures were decided on after a discussion with our project supervisor [2]. Our objective with the controls system was to make the arm movement as quick as possible, without sacrificing accuracy. Hence, we have a small rise time coupled with a small chance of overshoot. For any overshoot experienced, we need to allow for slightly more settling time. Our settling time parameter can have a larger acceptable range, as any overshoot experienced is often so minimal it can be discounted [2].

Table 1 - Desired Performance of the Control System

Parameter	**Desired Control System Performance**
Rise Time	< 0.5 s
Overshoot	< 5%
Settling Time	< 2s
Steady State error	Minimized (<2%)

The chosen power source must be able to supply power to two linear actuator motors, as well as two DC stepper motors. These motors will be manipulated based on logic from an Arduino Mega, which will also need to be powered from an external source. This will require the use of a

12V power supply to provide current on the ampere scale to the motors, and on the milliampere scale to an Arduino Mega. The final design uses a 12V 40A rated power supply to provide adequate amperage for all of the motors. It was found that the DC motors and linear actuators required the most current, with any current level under 20A not sufficient to drive all of the components.

The physical arm design includes parts and assemblies ranging from large supports, to small intricate gearboxes. The required design must be easily available to machine and must be able to handle repeated stress from use of the arm. Due to these requirements, a Laser Cutter machine was used to provide plywood parts with high tolerances. This provided accurate and reliable parts that were suitable for both gearboxes and supports.

IV. Literature Survey

Forward Kinematics

Kinematics deals with the study of motion but without considering the moments or the forces acting on the object [8]. Forward kinematics deals with calculating the position and orientation of a robotic arm using the joint variables, such as length of the link and angles between the joints.

The Denavit-Hartenberg Method is the most Common Method of describing robot kinematics [8]. Using Denavit-Hartenberg notation, you would need parameters to describe how a link frame (i) relates to another frame (i-1).

Link frames are attached as follows: joint axis are identified and imaginary lines are drawn along each axis. A point of intersection is determined between them, and hence a link frame origin is assigned. Axis are then assigned to the joints. The following equations are then derived to calculate the position and orientation of the arms [9].

$$ {}^{i-1}_{i}T = R_x(\alpha_{i-1})\,D_x(\alpha_{i-1})\,R_z(\theta_i)\,Q_i(d_i) $$

=

$cos\theta_i$	$-sin\theta_i$	0	a_{i-1}
$sin\theta_i cos\alpha_{i-1}$	$cos\theta_i cos\alpha_{i-1}$	$-sin\alpha_{i-1}$	$-sin\alpha_{i-1} d_i$
$sin\theta_i sin\alpha_{i-1}$	$cos\theta_i sin\alpha_{i-1}$	$-cos\alpha_{i-1}$	$-cos\alpha_{i-1} d_i$
0	0	0	1

(2)

Equation (2) is used to determine the general transformation matrices Where Rx and Rz represent rotation, Dx and Qi represent translation. The table above is a 4x4 matrix representing the product of the variables involved.

The end-effector forward kinematics are then determined by multiplying all the involved general transformation matrices:

$$\overset{base}{_{end\ effector}}T = \quad {}^{0}_{1}T \quad {}^{1}_{2}T \ldots \quad {}^{n-1}_{n}T \tag{3}$$

The purpose of this section is to provide the reader with a general understanding of forward kinematics, as it will assist them in understanding Inverse Kinematics, which is the main focus of this report.

Forward kinematics was considered for our approach, however due to a lack of resources and proofs of concept that exist on it, Inverse Kinematics was chosen.

Inverse Kinematics

Inverse kinematics deals with the calculation of all possible joint angles given the position and orientation of some points on the robot arm. The application for inverse kinematics is to allow for computation of required joint angles to achieve a desired position and orientation. For the purposes of this project, the desired position will be a location in applicable 3-D space that the end point of the arm can reach.

There are two main approaches for solving the kinematic equations associated with inverse kinematics; Algebraic and Geometric [9]. The Algebraic approach manipulates equations representing the link parameters for the arm to find the kinematic equations and solve for the joint angles of the links. In a Geometric approach, the system is made into multiple plane-geometry problems to solve for the joint angles of the links. Both approaches are applicable to inverse kinematics problems, but the geometric approach aligns better with the project as the robot arm for this project has only two joints for the arm and a limitation of range of motion in the joint. This allows for geometric planes to be matched easily to the current setup and allow for solving of the kinematic equations.

For this project, a geometric approach to inverse kinematics was chosen as our solution to calculating the joint angles required for a desired location in 3-D space.The implementation of this approach will be discussed in detail in section VI.

Motor Control

In order for complete autonomous motion in 3-D space, a feedback mechanism for motor control is required. This control system will drive the linear actuators and stepper motor of the robotic arm until the desired position is reached. The system will drive the motors while receiving feedback from potentiometers on the joints of the arm that gives information about the current position of the arm as it moves toward the desired position. Proportional-Integral-Derivative (PID) control is the most common control algorithm used in industry and has been universally accepted in industrial control systems throughout the world [2].

A PID controller is a feedback mechanism used to continuously calculate the error between a desired set point and measured control signal. PID control is widely usedThe classical expression for a PID controller is:

$$u(t) = ke(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{de}{dt} \quad (1)$$

Where u(t) is the control signal and e(t) is the error between the measured control signal and the desired set point. Each portion of the controller has its own gain, the proportional (P) gain k, the integral (I) gain k_i, and the derivative (D) gain k_d [1].

Individually, the effect of each portion of the controllers is summarized in Table 2 below. Each of the components has an effect on performance factors of the system, mainly: steady state error, rise time, settling time, and overshoot. For this project, a fast settling time with low overshoot and low error is the optimal controller goal.

Table 2 - Effect of control parameters on performance areas [1, 2]

Controller	**Steady State Error**	**Rise Time**	**Settling Time**	**Overshoot**
P	Decreases	Decreases	Increases	Increases
I	Eliminates	Decreases	Increases	Increase
D	No Effect	Decreases	Decreases	Decrease

Tuning

There are several acceptable methods of tuning PID controllers for optimal gain parameters. Two widely used methods are the “guess and check” and the Ziegler-Nichols method, where the guess and check method is self explanatory.

In this method, the integral and derivative terms are set to zero and the proportional gain is increased until oscillation is experienced [2]. As proportional is often the most important of the gain parameters, it makes sense to tune it alone first. After P has been set at an acceptable

value, the integral component is increased to minimize any oscillations [2]. The integral term will reduce the steady state error, but the tradeoff is an increase in overshoot [2]. By making a fast and responsive system the priority, some level of overshoot will always occur. Finally, the derivative term is increased until the loop performs acceptably quick to the set point [2]. When increasing the derivative term, overshoot will decrease but the system becomes sensitive to any noise or disturbance [2]. By tuning the parameters in this order, engineers can prioritize certain characteristics of the control system to maximize the intended results.

Similarly to the guess and check method, the Ziegler-Nichols method begins with setting I and D to zero and increasing the proportional gain until oscillation is experienced [2]. At this point, the critical gain and period of oscillation are recorded [2], and used to determine the integral and derivative parameters. Further calculations can be seen in Table 3.

Table 3 - Ziegler-Nichols Tuning, Oscillation Method [2]

Control	P	Ti	Td
P	0.5Kc	-	-
PI	0.45Kc	Pc/1.2	-
PID	0.60Kc	0.5Pc	Pc/8

Based on the desired performance requirements and the performance of the individual controllers, it is determined that a combination of the controllers will be required to match our needs. Although the Ziegler-Nichols method can give a higher degree of precision, we believe that the robot will need to be tuned through the guess and check method. As a physical system can have irregularities and slight imperfections in the building of the design, the guess and check method will allow for greater accuracy when testing and tuning.

V. Team Duties & Project Planning

As a group we were able to keep each individual involved throughout each step of the project. The beginning phases were primarily research tasks to better learn about the project goals. Every group member took part in this, sharing resources, to ensure that there was a mutual understanding on how concepts, such as inverse kinematics, function and how they are applicable to our robotic arm project. Inverse kinematics is a core concept of the project and had to be fully understood by each group member before moving onto the rest of the work.

The next steps were to map out the joint parameters and the position constraints of the arm. These two tasks were completed together by Kyle and Alex RB. Both Kyle and Alex RB. were in charge of wiring and connecting the robot to the testing circuit, as well as ensuring the MATLAB functions created were migrated over to the Arduino interface. Alex K. and Amr were in charge of creating the inverse kinematics functions for each degree of freedom, and then implementing each function in MATLAB to test theoretical values. Calvin and Connor were tasked with

creating and testing a PID control function through the Arduino software, and then tuning the required control parameters to best meet the acceptable values.

While mapping the joint parameters and positional constraints, problems occurred while reading the input values of the potentiometers. These values included a lot of noise, causing the analog values to lack consistency. This problem was solved in two parts, first by including capacitors between the analog signal and ground to filter noise, and second by introducing an averaging method in the software. These two solutions allowed for much more accurate input values from the potentiometers to be used in the feedback mechanisms. Additionally, the range of motion for the two linear actuators had to be constrained using software limits to prevent damage to the system or affect the potentiometer calibrations if an inappropriate value is input by the user. If these ranges were not constrained, the lower linear actuator has the ability to push the gripper into the table or cause the stand to tip over depending on the upper arm position, and the upper arm has the ability to cause the potentiometer to miss rotations and change the associated mapping values from analog input to angle (rendering the conversion inaccurate).

Programming the arms and gripper encoder also encountered some issues initially. The PID functionality for the arms is based on a setpoint that is used as a target position (ultimately user fed), which also includes an “error boundary” built around it. This allows the PID to stop once it has reached the setpoint ± error. While initially testing the arms (after converting to the single Arduino Mega) the motors would not stabilize at the setpoint, instead it would travel past the setpoint and then reverse back again, causing a large and unsteady oscillation. This was fixed by simply turning off the motors once the error boundary was reached, which was not initially set up properly. The encoder used for the opening and closing functionality of the gripper also seemed to have issues when closing the gripper after opening it. The encoder signals its direction based on the phase shift of the two signals, which then allows us in theory to track the current position of the claw. However, after testing the gripper would not decrement its position when closing the claw. A workaround was then put in place which bypasses the encoder entirely and uses print statements on the serial console to slow down the position counter instead.

While developing the inverse kinematics, Alex K and Amr faced minor difficulties with regards to calculating the angles of the arm. The initial plan was to determine the angles to the maximum degree of motion the arm is allowed to perform. This worked well for the first three angles out of four, however the final angle calculation was very difficult to perform due to the complexity of the mathematics involved and the resulting equations would have one variable too many, preventing the determination of a proper general solution. As a result, the gripper was constrained to a straight down position. Constraining the gripper managed to make the inverse kinematics equations simpler and easier to obtain. This way, the final angle was able to be calculated successfully, as the resulting geometry of constraining the gripper was significantly simpler than having the last joint at an incline. As a result, all of the required angles for the joints were able to be mapped.

While coding the PID control system, Calvin and Connor experienced minor difficulties with testing and debugging. The original plan was to build a rudimentary test function on a single DC

motor, and ensure that this motor could be controlled properly before attempting to run the functions on the robot arm. As neither member had previous experience with the Arduino platform, there was a learning curve at the beginning to understand the correct syntax and commands needed. Fortunately, Arduino software has many online resources as well as PID specific libraries and functions that helped with programming.

General troubleshooting was required when testing the standalone DC motor, then through a variety of functions Calvin and Connor were able to get the motor to turn until it reached a desired value, either forwards or backwards. Once the function was working, a trial and error approach was chosen to tune the K values for the PID controls in order to achieve our desired parameters. The functioning PID system was then able to be transferred over to the motors on the robotic arm so it could be tested and re-tuned for each joint.

In table 4, a rudimentary breakdown of each group member's tasks is shown.

Table 4 - Task Allocation Among Group Members

Group Member	**Tasks**
Alex K.	● Inverse Kinematics Research ● Inverse Kinematics Equation Design ● Inverse Kinematics Testing and Troubleshooting ● Inverse Kinematics MATLAB Function Implementation
Alex RB.	● Inverse Kinematics Research ● Hardware implementation, wiring, testing, troubleshooting ● Integrating PID control, feedback, and inverse kinematics
Amr	● Inverse Kinematics Research ● Inverse Kinematics Equations Design ● Inverse Kinematics Testing and Troubleshooting ● Inverse Kinematics MATLAB Function Implementation
Calvin	● Inverse Kinematics Research ● System of Equations Design ● PID Control (Function Design and Implementation) ● PID Testing & Troubleshooting
Connor	● Inverse Kinematics Research ● PID Control (Function Design and Implementation) ● PID Testing & Troubleshooting ● Website Design
Kyle	● Inverse Kinematics Research ● Linear actuator angle mapping ● Arduino programming, testing, troubleshooting ● Assisting with hardware wiring/troubleshooting

VI. Design Methodology & Analysis

Inverse Kinematics - Calculating Movement

Using knowledge built on inverse kinematics during background research, a system of equations was able to be developed to relate the unique robotic arm geometry to any set of cartesian coordinates, X_1, Y_1, and Z_1 within the robot arm's range. The system was analyzed on two cartesian planes, with the XY plane providing a top-view analysis of the robotic arm, and the XZ plane providing a side-view analysis. By doing this, enough relations were able to be found to relate each joint of the robotic arm to a desired end-effector position.

As discussed in Section V, the gripper was constrained to a position perpendicular to the base due to initial difficulties while modelling the kinematics with our number of degrees of freedom. In order to model the entirety of the joints along the robotic arm, including the gripper, two different positions were used: orientation position P(X_0, Y_0, Z_0) and rotation position P(X_1, Y_1, Z_1).

Modelled in Figure 1, both positions P(X_0, Y_0, Z_0) and P(X_1, Y_1, Z_1) can be seen. P(X_1, Y_1, Z_1) being the location of the end effector, and P(X_0, Y_0, Z_0) being the orientation vector responsible for positioning the first 3 joints of the robotic arm. The orientation position was calculated by taking in the cartesian coordinates of the end effector (P(X_1, Y_1, Z_1)), then subtracting a unit vector in the negative $\hat{z}$ direction, multiplied by the length of the gripper arm, a_3 to obtain P(X_0, Y_0, Z_0). This point was used in the previous calculations for the majority of the inverse kinematic equations.

$$P[X_0\,;\,Y_0\,;\,Z_0] = P[X_1\,;\,Y_1\,;\,Z_1] - a_3[\dot{x}\,;\,\dot{y}\,;\,\dot{z}]$$

$$\sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2} = 1;$$

Moving forward, the orientation vector P(X_0, Y_0, Z_0) will be used for calculations in the orientation section, and the end effector position P(X_1, Y_1, Z_1) will be of focus in the rotation section.

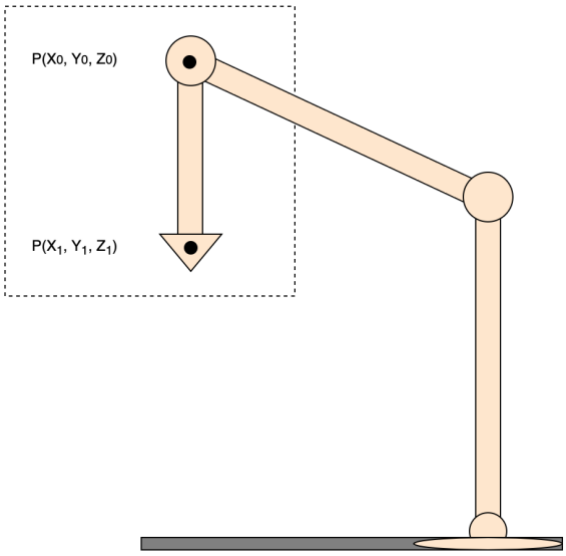


Figure 1 - Joint 0 and Joint 1 of robotic arm

Inverse Kinematics - Orientation

From the XY plane, the robotic arm can be modelled as shown in Figure 2. In this plane, four different parameters may be realized:

- X_0 is the desired position of orientation along the X axis.
- Y_0 is the desired position of orientation on the Y axis.
- r_1 is the projection of the two robotic arm lengths on the XY plane.
- θ_1 is the angle that the stepper motor creates at the base of the robotic arm.

By using the geometry of Figure 2, the first two inverse kinematic equations can be solved by realizing that both parameters r_1 and θ_1 may be written in terms of the known variables X_0 and Y_0. Using equation 1, the desired rotation required from the stepper arm may be solved. Equation 2 then solves for variable r_1. It is important to re-state that parameter r_1 does not actually correspond to any physical lengths of the robotic arm, however used to as an important parameter of analysis in the XZ plane.

$$\theta_1 = tan^{-1}(\frac{Y_0}{X_0})$$

Equation 1: Desired stepper motor angle.

$$r_1 = \sqrt{X_0^2 + Y_0^2}$$

Equation 2: Projection r_1 solved using pythagoras theorem.

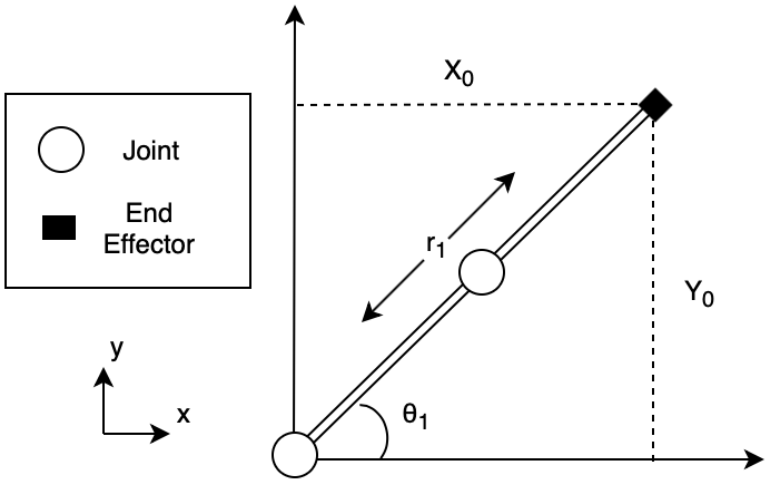


Figure 2 - Modelling the Robotic Arm in the XY plane (top-down view).

When the robotic arm is modelled in the XZ plane, the desired angles of the two robot arm joints are observed. These were labelled as parameters θ_2 and θ_3.

- θ_2 corresponds to the angle created by the lower joint on the robotic arm
- θ_3 corresponds to the angle created between the projection of the lower arm and physical position of the upper arm.

Using the geometry of the arm, two special triangles were created by using an elbow up configuration. From Figure 3, equations for θ_2 and θ_3 were formed in terms of the angles created by the special triangles.

$$\theta_2 = \varphi_2 + \varphi_1$$

Equation 3: The angle of the lower robotic arm joint is equal to the sum of angles created by triangle 1 and 2.

$$\theta_3 = 180° - \varphi_3$$

Equation 4: Solving for θ_3 using trigonometry.

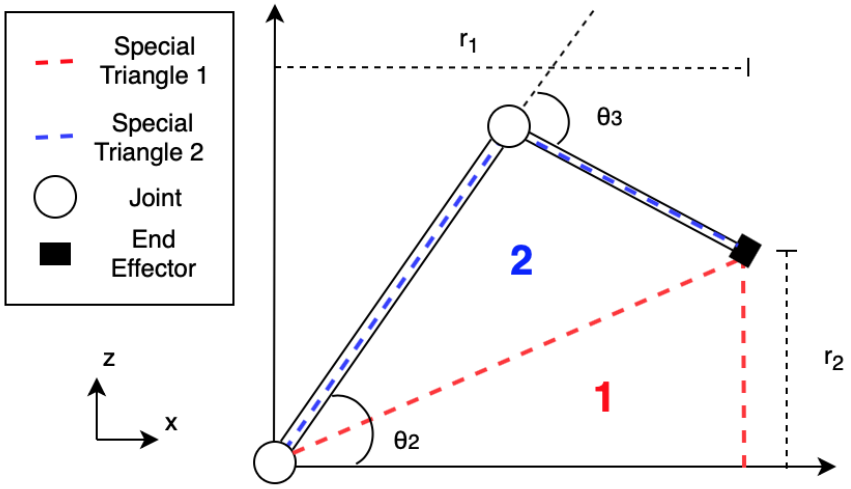


Figure 3 - Modelling the Robotic Arm in the XZ plane (side-on view).

Breaking the system into triangles 1 and 2 proved to be necessary in order to solve for the desired joint angles. By doing so, further equations were able to be written to solve for Φ_1, Φ_2, and Φ_3, as well as other parameters that aid in following analysis steps. Angle Φ_2, was solved for by using the geometry of triangle 1 in Figure 4 and realizing that parameter r_2 is equivalent to parameter Z_0 acquired from the given cartesian coordinates. Since r_2 is known to be Z_0 from equation 5, and r_1 was previously solved for through modeling the system in the XY plane, r_3 may also be solved through Pythagoras Theorem.

$$r_2 = Z_0$$

Equation 5: Z position relation

$$.\varphi_2 = tan^{-1}(\frac{r_2}{r_1})$$

Equation 6: Solving for Φ_2 using trigonometry.

$$r_3 = \sqrt{r_1^2 + r_2^2}$$

Equation 7: Solving for r_3.

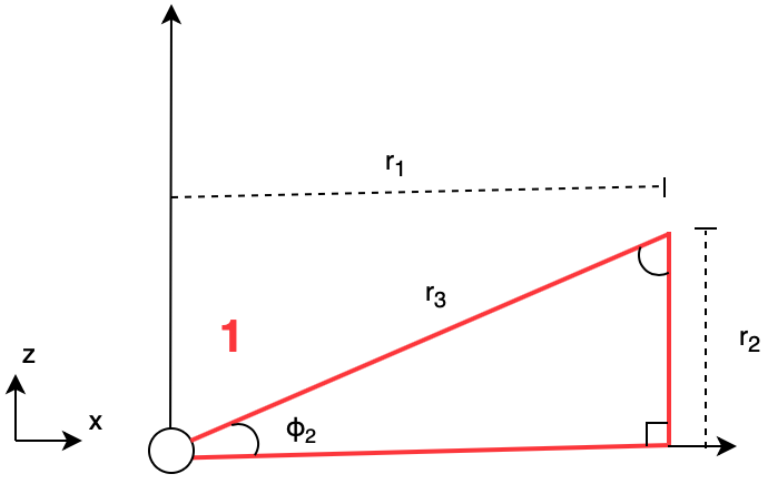


Figure 4 - Special Triangle 1, XZ plane.

The second special triangle shown in Figure 5 is a non-right angle triangle formed from robot arm lengths a_1, a_2 and the hypotenuse of triangle 1, r_3. By using the law of cosines combined with previous calculations to determine r_3, two equations for Φ_1 and Φ_3 were able to be obtained. Finally, these values, Φ_1, Φ_2, Φ_3, were able to be used with equations 3 and 4 to determine the final joint angle values θ_2 and θ_3.

$$\varphi_1 = cos^{-1}\left(\frac{a_2^2 - a_1^2 - r_3^2}{-2a_1r_3}\right)$$

Equation 8: Cosine law, solving for φ_1.

$$\varphi_3 = cos^{-1}\left(\frac{r_3^2 - a_1^2 - a_2^2}{-2a_1a_2}\right)$$

Equation 9: Cosine law, solving for φ_3.

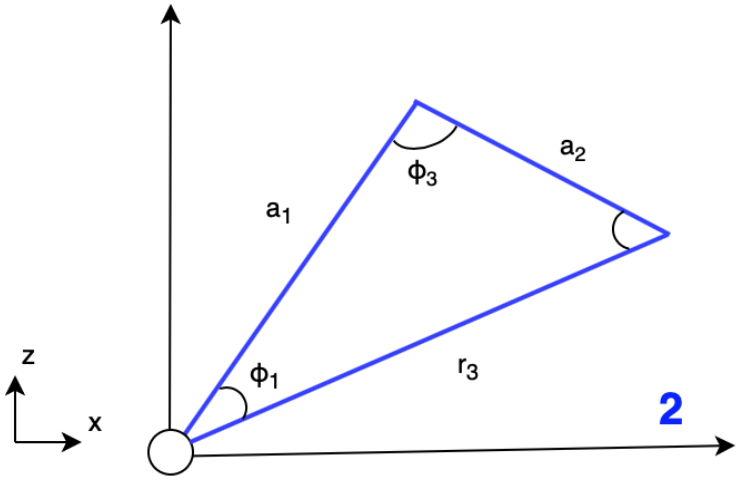


Figure 5 - Special Triangle 2, XZ Plane

Ultimately, these nine equations obtained through inverse kinematics produce a set of three angles, θ_1, θ_2, θ_3, to correspond to any point in cartesian space for the orientation vector $P(X_0, Y_0, Z_0)$, within the robot's constraints. Once these equations were obtained, they were placed in sequential order based on known values, such that one unknown may be solved from all previously known values. Table 5 at the end of the inverse kinematics section shows a full list of the inverse kinematic equations ordered sequentially based on the order that they are computed in.

Inverse Kinematics - Position (Gripper Arm Angle):

Since the gripper arm will always be constrained perpendicular to the base, the angle of the joint connection the gripper to the robot arm length a_2 will vary depending on other parameters of the robot. In order to calculate this angle, θ_4, further geometric analysis was conducted.

To determine the kinematics for the gripper, a 3rd special triangle was created (See Figure 6 below). The distance L_2 is determined by subtracting r_1 (this value was found using the orientation vector) and the distance L_1. The angle ϕ_4 is then calculated by obtaining the arcsine of L_2 divided by a_2. The final angle θ_4 is then determined by observing that the angle created is equal to 180° - ϕ_4.

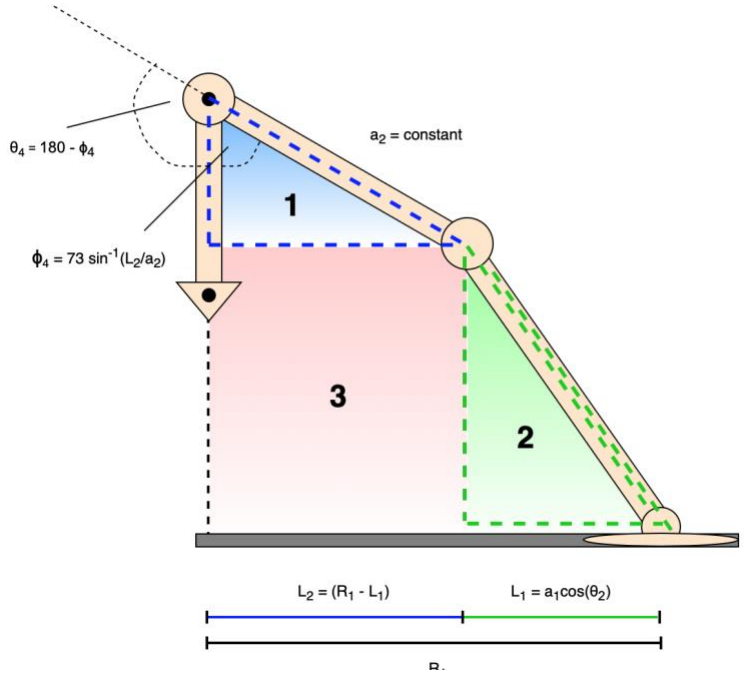


Figure 6 - Arm with 3rd Triangle

Inverse Kinematics - Implementing Equations:

The table below shows the list of equations obtained from the kinematics that were then integrated and implemented into MATLAB. They were tested with multiple values of the end-effector to determine if the angle values agreed with the values determined by hand. The code can be found in the appendix.

Table 5 - Inverse Kinematic Equations In Sequential Order

Complete List of Equations	
#1	$\theta_1 = tan^{-1}(\frac{Y_0}{X_0})$
#2	$r_1 = \sqrt{X_0^2 + Y_0^2}$
#3	$r_2 = Z_0$
#4	$\varphi_2 = tan^{-1}(\frac{r_2}{r_1})$
#5	$r_3 = \sqrt{r_1^2 + r_2^2}$
#6	$\varphi_1 = cos^{-1}(\frac{a_2^2 - a_1^2 - r_3^2}{-2a_1 r_3})$
#7	$\theta_2 = \varphi_2 + \varphi_1$
#8	$\varphi_3 = cos^{-1}(\frac{r_3^2 - a_1^2 - a_2^2}{-2a_1 a_2})$
#9	$\theta_3 = 180° - \varphi_3$
#10	$L2 = r_1 - L1$
#11	$\varphi_4 = sin^{-1}(\frac{L2}{a2})$
#12	$\theta_4 = 180° - \varphi_4$

PID Control - Testing & Tuning

A key part of our project was to design and code a PID control system that would assist with the arm movement at each joint. To begin, we determined that a PID test system would be designed and tested on an individual DC motor. We set up a test bench with a 12 V supply connecting a Pololu G2 High-Power Motor Driver 18v25 [4] with an Arduino Nano [5] controlling a DC motor [6].

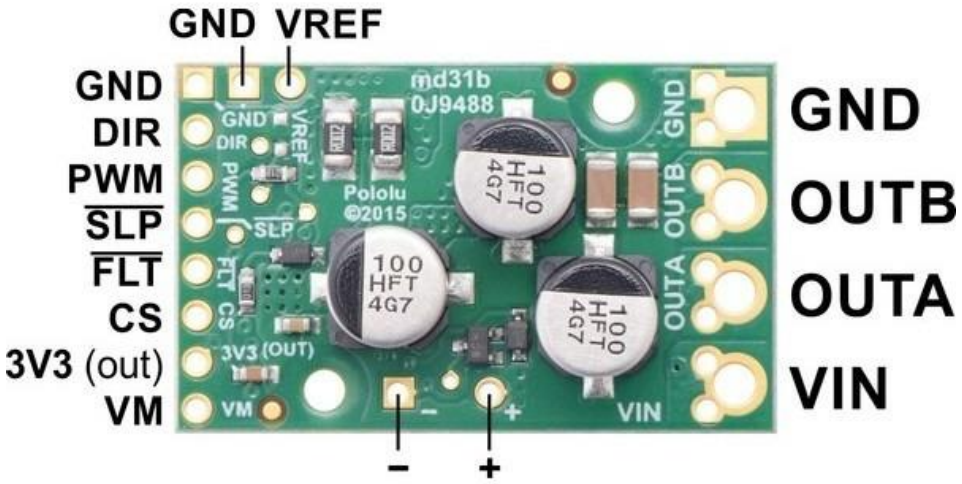


Figure 7 - Pololu G2 High-Power Motor Driver 18v25 [4]

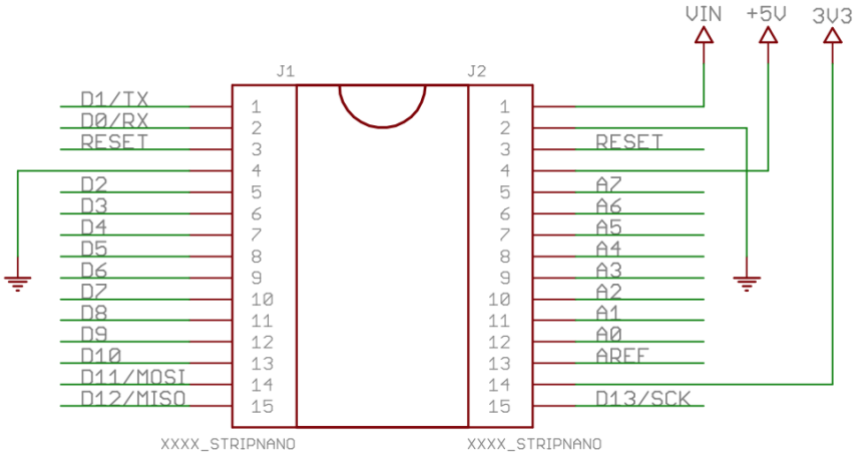


Figure 8 - Arduino Nano Pin Layout [5]

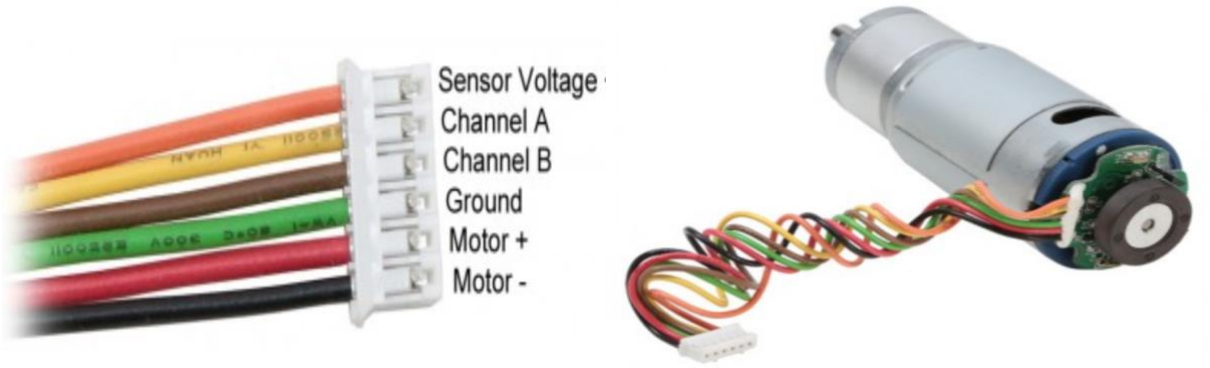


Figure 9 - DC Encoder Motor Connections [6]

The Arduino utilizes a PWM signal to send commands to the motor. To begin we alternated HIGH and LOW signals, as the motor driver is enabled with direction control.

The test circuit was used to determine how to run a DC motor in both forward or reverse rotation up to a setpoint. Using the Arduino syntax and HIGH/LOW commands, we began to test the motor. Fortunately, the Arduino platform has a dedicated PID library that can be used to automatically compute the required calculations for the PID function when given K values and a specified setpoint [7].

The final arm design maps joint angles to specific potentiometer values, and will move until these values become the current position. When testing our individual DC motor, we decided to use a manually chosen setpoint to mimic the operating conditions of the robot. We created two functions, one would have the motor run in the forward direction until it reached the setpoint and the other running in reverse to accomplish the same goal. By experimenting with different setpoints and PID parameters, we were able to see the different effects the K values had on error, overshoot and settling time. As our single DC motor was much less complex than the final robot, we determined that any specific tuning of the test motor was unnecessary as it would have to be redone completely on the arm. Through the test bench setup, we were able to write a baseline program that could then be further implemented on the arm.

The complete design would receive the analog signal from the potentiometer corresponding to the joint it is moving. The signal would then be converted to a digital value between 0 and 1023 and input that value into the PID function (also specified for that joint). The potentiometer reading will then be compared with the destination position and begin movement. All joints will be actively moving to their destinations at the same time based on the PID assessment.

VII. Final Design Details

Our final robot arm is shown in Figure 10. This design encompasses the robot arm and relevant components, as well as the wiring and circuitry for running autonomously. The robotic arm is divided up into two major sections; the ‘Arm’, which encompasses the two linear actuators and the base stepper motor, and the ‘Gripper’, which encompasses the pitch stepper motor, the rotational stepper motor, and the DC motor controlling the claw.

During the testing, the Arm section was used to complete the large majority of the movement while the Gripper was used to position the claw around the object to be picked up. The Gripper has been constrained to always point straight down, since this was the optimal way to pick up the object.

Figure 10 - Final Robot Arm

The final design required 6 motors; 3 stepper motors, 1 DC motor, and 2 linear actuators. Each of the motors require a motor driver and for implementing the control systems, each motor requires feedback on the current position. For the feedback 5 potentiometers, a rotary encoder, and a limit switch were used. The total pin requirements for this complete system equates to 16 digital pins and 5 analog pins, which is why an Arduino Mega was used. The system was breadboarded for the final testing and is shown below in figure 11.

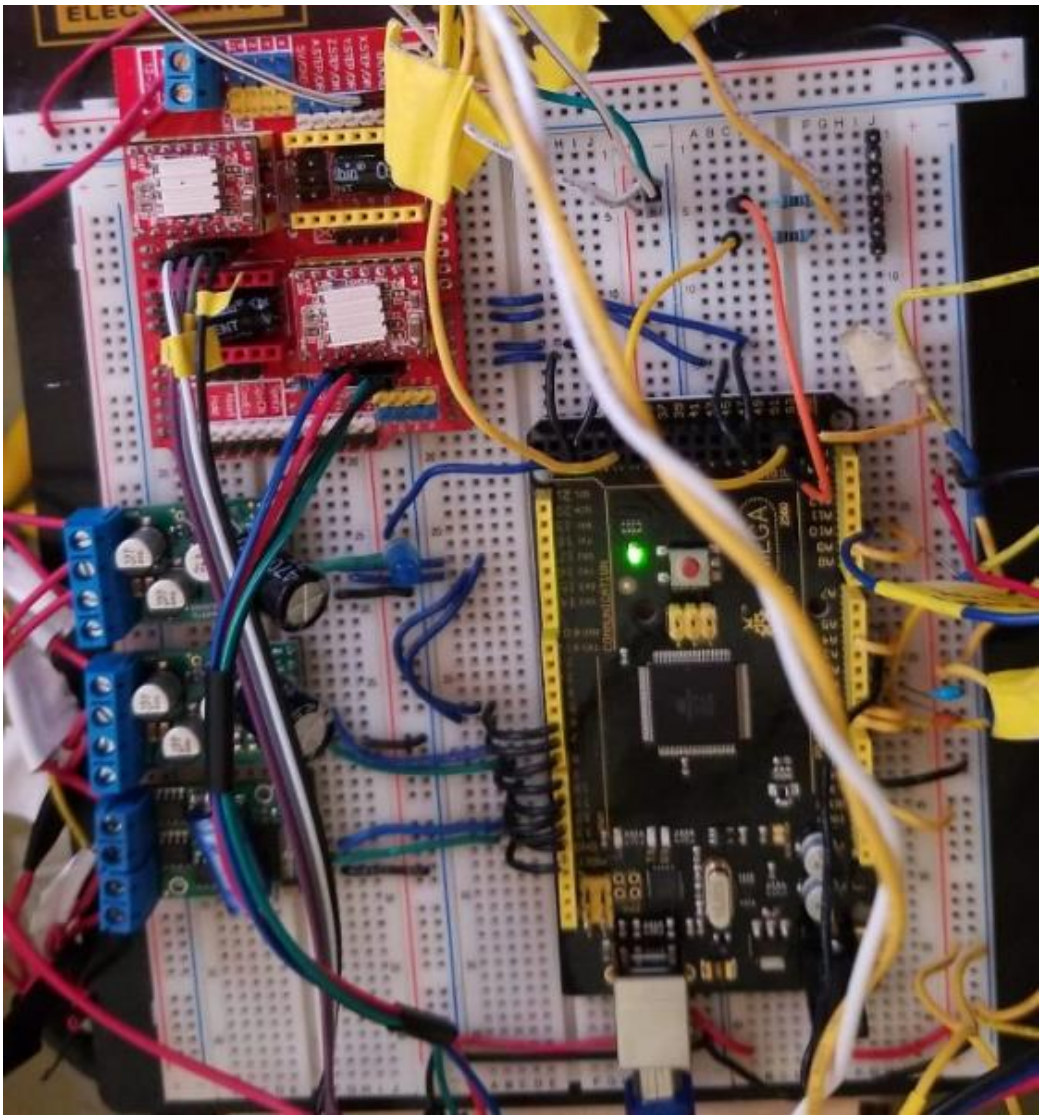


Figure 11 - Final Circuit Board

For the final design, 6 motor drivers were used. For the linear actuators and the DC motor used for the claw, the Pololu G2 High powered motor driver described in section IV was used. This driver is rated to easily handle the 12V and 10A load that comes with the high powered linear actuators. For the Pitch and Rotational stepper motors, and A4988 stepper motor driver was used. This driver is capable of handling supply voltages from 8-35V, while limiting the output

current based on the supply voltage as to not damage the stepper motors. The wiring for the A4988 driver is shown below in figure 12.

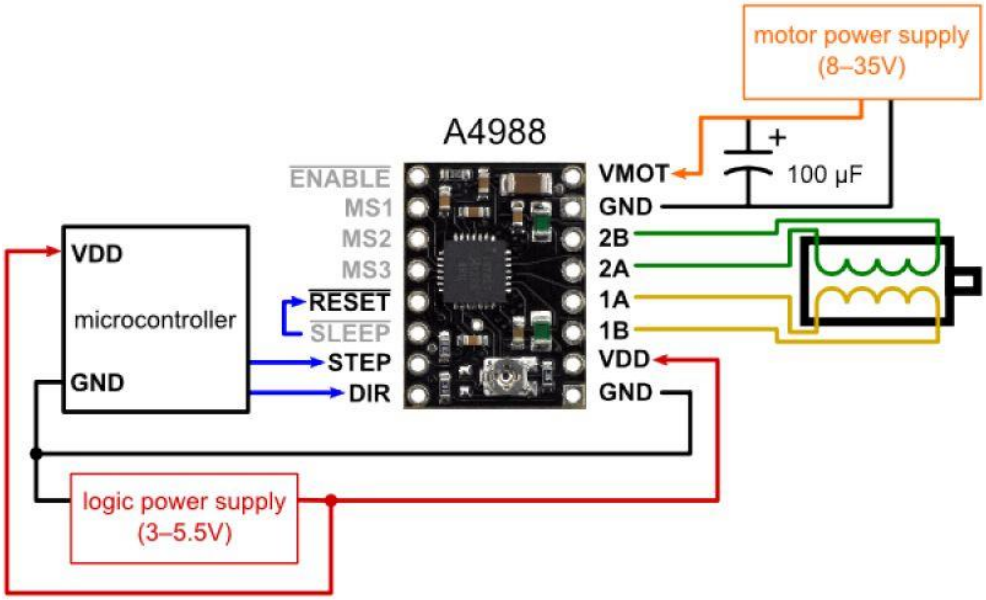


Figure 12 - A4988 stepper driver (https://solarbotics.com/product/51090/)

For the base stepper, a microstep driver was required. This driver can handle inputs from 9-42V, output a current of 0.5-4A, and can be set to different microstep values depending on the situation. The driver and connections is shown below in figure 13.

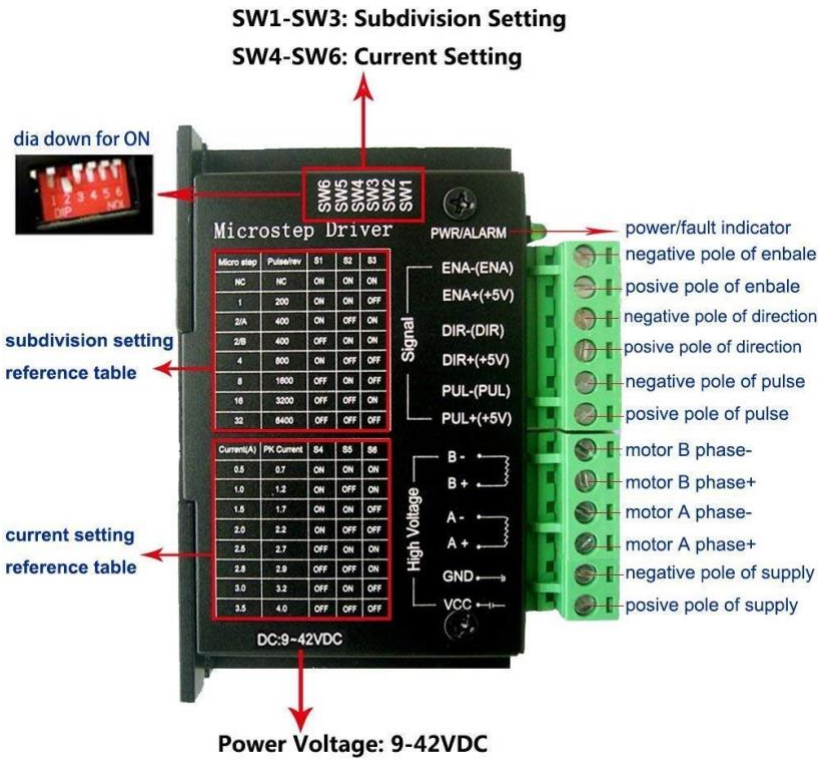


Figure 13 - Base stepper motor driver (https://www.amazon.ca/gp/product/B07B9ZQF5D/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1)

The wiring connections for the entire system is shown in figure 14 below.

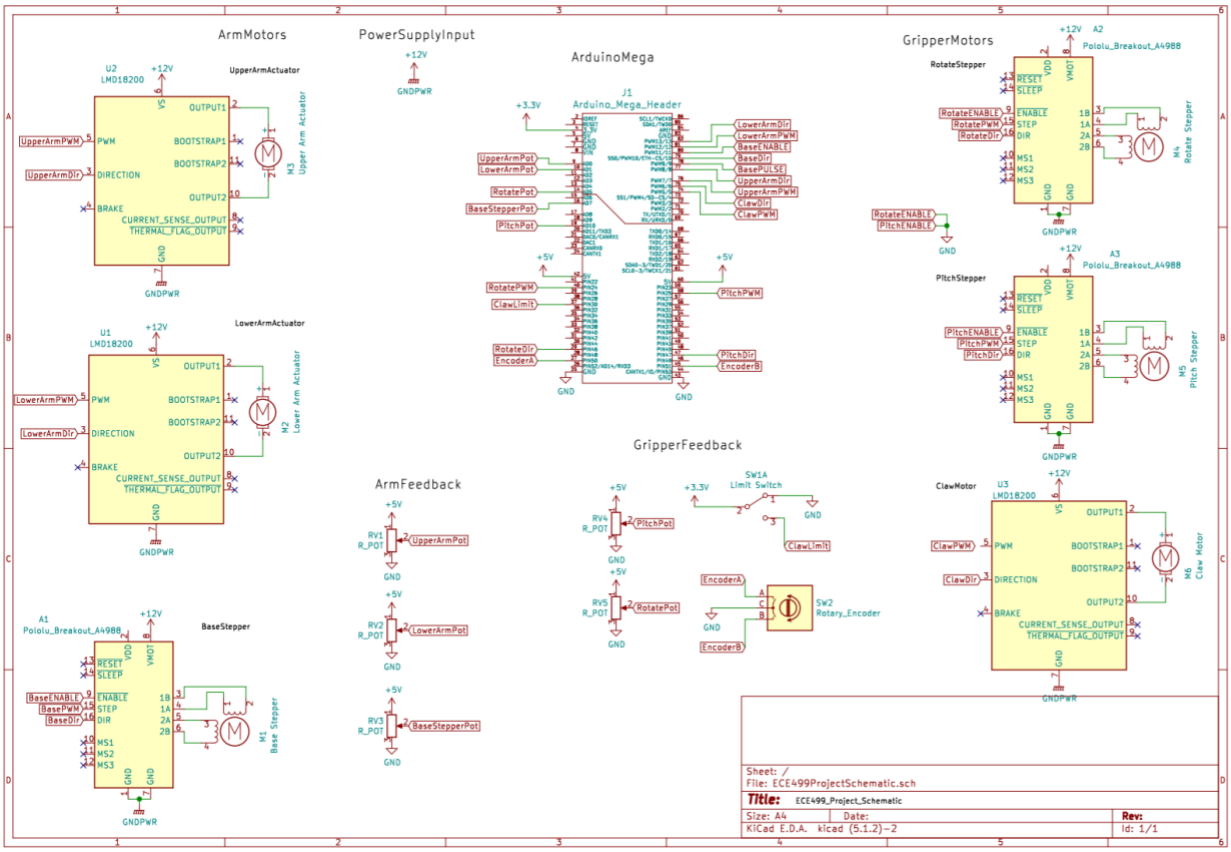


Figure 14 - Final Circuit Board Wiring Schematic

The rotary encoder was used to determine the current position of the DC motor controlling the gripper. Since the rotary encoder is only relative control, it is necessary to set a start point to begin the feedback, which is where the limit switch came in. The claw motor was run until the limit switch was triggered, and the start position of the rotary encoder was set.

The rest of the feedback was performed by potentiometers, which give absolute position control, meaning the potentiometer value is mapped to an angle corresponding to the joint it is attached to. Based on the potentiometer value and the mapped angle, the corresponding motor could be run until the required angle matched the measured angle. The potentiometers were connected the assigned joint by a pair of gears, one fixed to the moving joint and the other fixed to the potentiometer. This allowed the potentiometer to change values with the joint as the joint was moving.

Some of the gears were 3D printed, while others were lasercut. Eventually, all the gears will be 3D printed as it allows for better tolerances between the gear teeth and therefore more accurate potentiometer readings.The 3D printed gears attached to the Pitch stepper motor is shown in figure 15.

Figure 15 - Pitch stepper motor potentiometer gear assembly

The system is powered by a 12V continuous DC power supply, capable of drawing up to 40A. Table 6 shows the approximate current draw for each of the motors used in the final design, adding up to just under 30 A of current. For the final design, the number of motors moving at a given time was limited as the motors could sometimes interfere with one another. Typically, the Arm and Gripper motors were run separately; the Arm first to move into position, followed by the Gripper to position the claw to pick up the object. Therefore, the typical current draw was limited to 20 A for the arm and approximately 5 A for the gripper.

Table 6 - Approximate current draw of the system

Motor	**Approximate Current Draw (A)**
Upper Linear actuator	10
Lower linear actuator	10
Base Stepper motor	2.5
Pitch stepper motor	1.5
Rotational stepper motor	1.5
Claw DC motor	2
Total	**27.5**

Due to the large size of the Gripper, the weight of the robot arm was extremely shifted forward, causing a large amount of excess movement in the Arm which negatively affected the feedback mechanisms of the potentiometers. To overcome this, a delrin spacer was placed between the base of the arm and the stand. This spacer was successful in stopping the excess movement but created a large amount of friction between the base of the arm and the stand which was too much for the base stepper motor to overcome and rotate. For this reason, the final design was implemented without any base rotation, constraining the arm to move in one plane.

Additionally, the movement of the Arm was constrained in software by setting minimum and maximum angle values for the linear actuators as to not operate in areas that could damage the Arm, or change the mapped values of the potentiometers by causing the gears to miss turns. It was found during testing that movement that exceeded the maximum Upper actuator angle and the minimum lower actuator angle could cause the robot arm to become unbalanced and potentially fall into the table, while also changing the values of the calibrated potentiometers forcing them to be recalibrated. For future tests, the stand of the robotic arm will need to be bolted down to ensure complete support of the system.

Table 7 - Software set minimum and maximum angles

Motor	**Max angle (Degrees)**	**Minimum angle (Degrees)**
Upper actuator	110	49
Lower actuator	93	50

VIII. Testing & Validation

To begin, we tested each joint individually to confirm operation. Each potentiometer outputs a specific voltage that corresponds to a unique angle, as seen in the tables below. The voltages were read as analog values from each sensor.

Table 8 - Lower Arm Joint Mapping

θ_1 (Degrees between base & lower arm)	**Sensor Value (Analog 0-1023 bit)**	**Voltage (V)**
93 (max value)	205	0.939
80	263	1.201
70	315	1.433
60	360	1.633
50	401	1.819
40	433	1.960
30	467	2.115
20	500	2.267

Table 9 - Upper Arm Joint Mapping

θ_2 (Degrees between lower arm & upper arm)	**Sensor Value (Analog 0-1023 bit)**	**Voltage (V)**
49	900	4.055
60	848	3.830
70	812	3.670
80	750	3.394
90	705	3.192
100	664	3.003
110	625	2.830
123 (max value)	602	2.700

We set a specific setpoint for the arms individually, and ran the program to test our code and physical response. Once we had the arm move to the correct setpoint, we tuned and tested the K values in our PID control functions to account for any experienced overshoot and error.

An issue we experienced was finding an acceptable error value. Without an error range the arm would continuously try to correct itself to the exact setpoint and be unable to finish the movement. By setting a physical error range around the setpoint, e.g. setpoint ± 10 (based on an analog reading from the potentiometer), we had the robot arm move rapidly to the setpoint and settle quickly. We started with the lower arm, then the upper arm, and once each worked we tested the arms simultaneously to a variety of setpoints.

We tested the pitch and gripper separately from the arms, as the inputs were connected to two different Arduino Nano. We set the pitch to constrain the gripper movement to a constant 90 degree angle, or to always be pointing straight down regardless of the arm position. This required constant adjustment of theta 3, as the angle had to change depending on the two other arms. We also locked the rotation of the gripper into one place, in order to keep the gripper position constant when picking up an object. As the gripper initialized, the motor in charge of rotation would lock the claws in place. If any disturbance was felt, the motor would automatically readjust the claw position to the original locked setpoint.

To test the claw, we first initialized it to close fully each time a new setpoint was entered regardless of where it was positioned. Once the start was reached, it then would extend until the setpoint was reached and stop. By experimenting with different setpoint values, we determined the limits of the claw and what potentiometer analog signal corresponded to the correct width to pick up our object.

We were able to obtain an Arduino Mega board, allowing us to merge each Nano's pin connections onto a single board. Once the Mega was set up, we merged the separate codes into a single file, and began testing each section individually to confirm operation had not changed from the Nano's.

Once all the parts were working individually, it was brought together to test in one fluid motion. For testing purposes we put a wood block down and provided the arm with all the inputs needed to pick up the block. The act of picking up the block was done in sequences, moving to the home position, then going to the block and closing the grip on it, then taking it to a destination position. While testing the whole arm we found that if we moved too many parts at once there would be more errors, this was likely due to limitations on how fast the arduino can process information. To account for the error we chose to move the arm in a set order, first initializing the gripper, then moving the arms, then adjusting the pitch of the gripper, and finally closing or opening the hand. Once it was all moving and able to pick up and put down the block, we knew it was fully functioning.

IX. Discussion & Recommendations

Results

Overall, our general design objectives were reasonably met within our expected results. Our initial design planned to have a rotating base, however the base motor failed and as a result we weren't able to include the first kinematics angle for rotation (theta1). However, all other kinematics angles were successfully calculated using the end-effector position and hence we were able to implement inverse kinematics on the arm. With added time and resources, we may have been able to reach a more complete state where all our objectives were met.

Recommendations

As the base on the robot is currently unavailable for movement, the accessible range of the robot's reach is limited. The spacer that allows the base to move and turn proved undersized for our activities. When the arm extends out, the base tilts slightly. This is not noticeable when the arm is controlled by a joystick, but when it depends on specific angles to move autonomously it makes the potentiometer values inaccurate. In order to minimize any angle error, the spacer should be replaced by one with a larger circumference.

Our robot arm is able to locate and pick up an object based on a set of specific coordinates. Currently, these coordinates are limited to a set location that is entered into the program by the operator. The next step would be to map out a grid of all points the robot can physically access, and place objects anywhere in the grid for the arm to locate. This would allow for a better presentation that showcases the full range of movement that the robot has access to.

To improve performance and increase the autonomous capabilities, an object detection system should be implemented in the future. There are multiple object detection options available, with varying accuracy and cost. LiDAR is a system of active sensors that send out waves of energy to calculate how far away objects are, and can pinpoint the exact distance of an item [12]. Cameras are another alternative, often using a system of cameras to triangulate distances and potential objects [12]. Overall, an ultrasonic wave system such as LiDAR is quite costly [12] when compared to a simple camera set up, and for the object detection purposes of the robot could prove to be overkill. One solution would be to implement a cheaper camera system first, and if this proved unreliable to then make the transition to a LiDAR based system.

Future Work

Our robot arm design is a part of the Uvic Robotics Club's "Rover" project, which will have the arm mounted onto a vehicular base. The rover will have the option to be both manually and autonomously controlled and will be able to avoid obstacles and navigate a set course in a robotics competition. Once the arm is attached to the rover, the vehicle will be able to move to

items and pick them up based on their coordinates. The rover will have a “return home” function similar to the arm and will then move back to its original starting point to drop off the item.

With further research in inverse kinematics, full use of the gripper arm can be implemented. The gripper arm would be able to move freely to any angle combination most efficient to pick up a specific object. This would include more research into the inverse kinematics for articulated manipulators and transition matrices to implement a fully inclusive version of inverse kinematics. That being said, having the gripper constrained perpendicular to the base will still be a popular method to pick up objects, and this mode can be coded through software to be used when the application fits.

As discussed, computer vision would be an invaluable addition to the robotic arm. This would allow for the elimination of inputting cartesian coordinates that dictate the movement of the arm. Instead, an attached system would be able to feed the required location to the Inverse Kinematics program, which would then be converted into the required angles needed to achieve that position. Due to the timeline of the project, the addition of computer vision falls under future work and is highly recommended to be implemented into a later design revision. A starting point for this work would include research into the openCV library to begin programming for computer vision.

X. Conclusion

In conclusion we found this project a great success. As a team we were able to take a robotic arm limited to human controls and give it the ability to reach a destination coordinate on its own. Some of the objectives were not completed due to a lack of time for shipping and manufacturing of replacement parts. Regardless, the robotic arm was still able to achieve near full functionality, effectively demonstrating the purpose of our project.

Working alongside the UVic robotics clubs provided us with some limitations and value. They were very knowledgeable about the arm and always willing to provide assistance. The downside however, was we were working on their arm, which meant our work schedule was based around their timeline and if they needed the arm for competitions or presentations we were unable to work on it.

Overall, as a team, we are very proud of what we were able to accomplish and are excited to take this knowledge forward into our future projects and workplaces.

XI. References

[1] D. Honeywell, "PID Control", Cds.caltech.edu, 2019. [Online]. Available: https://www.cds.caltech.edu/~murray/courses/cds101/fa04/caltech/am04_ch8-3nov04.pdf. [Accessed: 20- Jun- 2019].

[2] "PID Theory Explained - National Instruments", Ni.com, 2019. [Online]. Available: http://www.ni.com/en-ca/innovations/white-papers/06/pid-theory-explained.html. [Accessed: 20- Jun- 2019].

[3] "These countries have the most robot workers", World Economic Forum, 2019. [Online]. Available: https://www.weforum.org/agenda/2019/05/infographic-the-countries-with-the-highest-density-of-robot-workers/. [Accessed: 05- Aug- 2019]

[4] Personal Conversations, “Dr. Pan Agothoklis”, May-August 2019.

[5] S. Skogestad, "Probably the best simple PID tuning rules in the world", Pdfs.semanticscholar.org, 2019. [Online]. Available: https://pdfs.semanticscholar.org/05b2/07a58365e1b19485ca82cac53bb8e9ea1025.pdf. [Accessed: 05- Aug- 2019]

[6] Pololu G2 High-Power Motor Driver 18v25", Pololu.com, 2019. [Online]. Available: https://www.pololu.com/product/2994. [Accessed: 05- Aug- 2019]

[7] E. Vita, "Arduino Nano-Rev3.2", Arduino.cc, 2019. [Online]. Available: https://www.arduino.cc/en/uploads/Main/Arduino_Nano-Rev3.2-SCH.pdf. [Accessed: 05-Aug- 2019]

[8] 313 RPM HD Premium Planetary Gear Motor w/Encoder", Servocity.com, 2019. [Online]. Available: https://www.servocity.com/313-rpm-hd-premium-planetary-gear-motor-w-encoder?fbclid=IwAR1r8KtTSA1urY_hubvSbjcRtHp3nNI3CKtOhtQ00FMSKSz4fI9fsWWOPIg. [Accessed: 05- Aug- 2019]

[9] *Arduino Playground - PIDLibrary*. [Online]. Available: https://playground.arduino.cc/Code/PIDLibrary/. [Accessed: 05-Aug-2019]. (myPID function library)

[10] S. Cubero, *Industrial robotics: theory, modelling and control*. Mammendorf: Pro-Literatur-Verl., 2007.

[11] J. J. Craig, *Introduction to robotics*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2005.

[12] "LiDAR v. Cameras - The Autonomous Vision Race", *motivo*, 2019. [Online]. Available: https://www.motivo.com/single-post/2018/02/07/LiDar-v-Cameras---The-Autonomous-Vehicle-Race. [Accessed: 05- Aug- 2019].

[13] A. Driver, "A4988 Stepper Motor Driver :: Solarbotics", *Solarbotics.com*, 2019. [Online]. Available: https://solarbotics.com/product/51090/. [Accessed: 05- Aug- 2019].

[14] *Amazon.ca*, 2019. [Online]. Available: https://www.amazon.ca/gp/product/B07B9ZQF5D/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1. [Accessed: 05- Aug- 2019].

XII. Appendix A: Arduino Code

```
#include <Stepper.h>
#include <PID_v1.h>
#define ENCODER_DO_NOT_USE_INTERRUPTS
#include "A4988.h"
#include <math.h>

// arm defines
#define UPPER_PWM 7
#define UPPER_DIR 8

#define LOWER_PWM 12
#define LOWER_DIR 13

#define STEPPER_ENABLE 11
#define STEPPER_DIR 10
#define STEPPER_PULSE 9

// gripper defines
//Pitch is up/down motion of gripper
#define PITCH_PWM 25
#define PITCH_DIR 47

//Set RPM and step number for Pitch Stepper
#define RPM 25
#define STEPS 400

//Spin is the rotational motion of the gripper
#define SPIN_PWM 24
#define SPIN_DIR 46

//Claw is the opening/closing of the fingers
#define CLAW_PWM 5
#define CLAW_DIR 6

//Define encoder digits
#define ENCODER_A 50
#define ENCODER_B 51

#define LIMITSWITCH 30
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ global variables
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// input coordinates
int x_coordinate;
int y_coordinate;
int z_coordinate;

// Upper arm variables
int upperDirection = 0;
double upper_max_lim = 900;          // angle = 49deg
double upper_min_lim = 620;          // angle = 115deg
int theta3;                 // upper arm
double Setpoint_upper;       // target upper arm position
double Input_upper;          // upper arm pot reading
double Output_upper;         // PID speed control
int error_upper = 10;        // error boundary
int arm_count = 0;
double average_upper;

// Lower arm variables
int lowerDirection = 0;
double lower_max_lim = 400;          // angle = 20deg
double lower_min_lim = 200; // angle = 90deg
int theta2;                 // lower arm
double Setpoint_lower;       // target lower arm position
double Input_lower;          // lower arm pot reading
double Output_lower;         // PID speed control
int error_lower = 10;        // error boundary
double average_lower;

// claw variables
int error_spin = 30;         // position error boundary for spin
int currentStateCLK = 0;     // encoder variable
int stateB;                 // encoder variable
int average_dir = 0;         // encoder counter
int previousStateCLK = 0;            // encoder variable
int Setpoint_claw;           // claw goal position
int claw_max_pos = 400;      // claw max limit
int claw_min_pos = 0;        // claw max limit
int clawState = 1;           // 0 = off, 1 = on
int clawDirection = 0;       // direction
int count2 = 0;              // average counter for claw function
int claw_position = 0;       // current position of claw
```

```
int current_claw_pos = 0;
int Limit = 0;              // used for claw initialization
int difference;

// pitch variables
double average_pitch;       // average of pitch position readings
double Setpoint_pitch;      // pitch goal position
double Input_pitch;         // pitch position reading
int count = 0;              // average counter for pitch function
int theta4;                 // pitch angle
int error_pitch = 15;       // position error boundary for spin

// spin variables
double Setpoint_spin;
double Input_spin;

//Completion flags
bool upperarmDone = 0;
bool lowerarmDone = 0;
bool pitchDone = 0;
bool clawDone = 0;
bool Event = 0;

// serial read variables
int input1;                 // x or theta1
int input2;                 // y or theta2
int input3;                 // z or theta3
int input4;                 // gripper coordinate
int input_count = 0;
char data;
String command = "";
bool restart = 0;

// inverse kinematics variables
int a1, a2, a3;
int wx, wy, wz;
int r1, r2, r3;
int X0, Y0, Z0;
int phi1, phi2;
int length1, length2;

// stepper setup
A4988 stepper_pitch(STEPS, PITCH_DIR, PITCH_PWM);
A4988 stepper_spin(STEPS, SPIN_DIR, SPIN_PWM);
```

```
// PID functions
//Specify the links and initial tuning parameters
double Kp=2, Ki=0, Kd=0;
PID myPID_upper_rev(&average_upper, &Output_upper, &Setpoint_upper, Kp, Ki, Kd,
REVERSE);
PID myPID_upper_fwd(&average_upper, &Output_upper, &Setpoint_upper, Kp, Ki, Kd,
DIRECT);
PID myPID_lower_rev(&average_lower, &Output_lower, &Setpoint_lower, Kp, Ki, Kd,
REVERSE);
PID myPID_lower_fwd(&average_lower, &Output_lower, &Setpoint_lower, Kp, Ki, Kd, DIRECT);

// Setup code, runs once
void setup() {
  Serial.begin(9600);

  // arm setup
  pinMode(UPPER_PWM, OUTPUT);
  pinMode(UPPER_DIR, OUTPUT);

  pinMode(LOWER_PWM, OUTPUT);
  pinMode(LOWER_DIR, OUTPUT);

  pinMode(STEPPER_ENABLE, OUTPUT);
  digitalWrite(STEPPER_ENABLE, HIGH);
  pinMode(STEPPER_DIR, OUTPUT);
  pinMode(STEPPER_PULSE, OUTPUT);
  digitalWrite(STEPPER_PULSE, LOW);

  //turn the PID on
  myPID_upper_rev.SetMode(AUTOMATIC);
  myPID_upper_fwd.SetMode(AUTOMATIC);
  myPID_lower_rev.SetMode(AUTOMATIC);
  myPID_lower_fwd.SetMode(AUTOMATIC);

  // gripper setup
  pinMode(PITCH_PWM, OUTPUT);
  pinMode(PITCH_DIR, OUTPUT);

  pinMode(CLAW_PWM, OUTPUT);
  pinMode(CLAW_DIR, OUTPUT);

  pinMode(SPIN_DIR, OUTPUT);
  pinMode(SPIN_PWM, OUTPUT);
```

```
  pinMode(ENCODER_A, INPUT_PULLUP);
  pinMode(ENCODER_B, INPUT_PULLUP);

  pinMode(LIMITSWITCH, INPUT);

  stepper_pitch.begin(RPM, 1);
  stepper_spin.begin(RPM, 1);

  Setpoint_spin = 710;          // neutral position for claw spin

  // inputs
  Setpoint_claw = 20;           // claw position
  theta4 = 90;                  // pitch angle
  theta3 = 90;                  // Upper arm angle
  theta2 = 90;                  // Lower arm angle

  angle_upper_to_reading(theta3);
  angle_lower_to_reading(theta2);
  pitchAngleToPot(theta4);

  // Reset claw
  initializeClaw();
  // claw position = 0

  difference = Setpoint_claw - current_claw_pos;
}


// Main code, runs repeatedly
void loop() {
//
//  while(1){
//        Serial.println(analogRead(A10));
//  }

  // Move upper arm down (in case of fire)
//  while(1){
//        digitalWrite(UPPER_DIR, 1);
//        analogWrite(UPPER_PWM, 127);
//        Serial.println(analogRead(A0));
//  }

  // Move upper arm up (in case of fire)
```

```
//  while(1){
//        digitalWrite(UPPER_DIR, 0);
//        analogWrite(UPPER_PWM,127);
//  }

  // Move lower arm down (back)
//  while(1){
//        digitalWrite(LOWER_DIR, 1);
//        analogWrite(LOWER_PWM, 127);
//        Serial.println(analogRead(A1));
//  }


  //spin(); // spin to gripper to neutral position (Setpoint_spin)

  if((upperarmDone && lowerarmDone) == 0){
        //Serial.print(upperarmDone && lowerarmDone);
        arms();          // move lower and upper arms to desired angles (theta3, theta2)

  } else if (pitchDone == 0){
        // move to desired pitch (theta4)
        pitch(); // runs to theta4

  } else if (clawDone == 0){
        // move to desired gripper position (claw_position)
        gripper();

  } else if (upperarmDone == 1 && lowerarmDone == 1 && pitchDone == 1 && clawDone == 1){
        current_claw_pos = claw_position;
        read_inputs();
  }
}

// moves gripper to claw_position
void gripper(){
  // clawDirection => 0 = open claw, 1 = close claw
  // clawPosition => higher = more open, lower = more closed

  if (clawState == 1){  // only read values in motor is running

        difference = Setpoint_claw - current_claw_pos;

        stateB = digitalRead(ENCODER_B);
        currentStateCLK = digitalRead(ENCODER_A);
```

```
//        stateB = (PORTB & 4) >> 2;
//        currentStateCLK = (PORTB & 8) >> 3;

        // If the previous and the current state of the inputCLK are different then a pulse has
occured
        if (currentStateCLK != previousStateCLK){

        // If the inputDT state is different than the inputCLK state then
        if (stateB != currentStateCLK) {         // Encoder is rotating CW (closing)
        average_dir--;
        } else {                    // Encoder is rotating CCW (opening)
        average_dir++;
        }

        previousStateCLK = currentStateCLK;
        }

        if (count2 == 5){
        if (average_dir < 0){   // average CW (closing)
        claw_position--;
        } else {            // average CCW (opening)
        claw_position++;
        }

        // move claw to desired position
        if (difference != 0){
        if (difference > 0){       // open claw more
        analogWrite(CLAW_PWM, 100);
        clawDirection = 0;
        digitalWrite(CLAW_DIR, clawDirection);
        clawState = 1;
        current_claw_pos++;
        } else {
        analogWrite(CLAW_PWM, 100);
        clawDirection = 1;
        digitalWrite(CLAW_DIR, clawDirection);
        clawState = 1;
        current_claw_pos--;
        }
        } else { // turn off claw,
        analogWrite(CLAW_PWM, 0);
        clawState = 0;
        clawDone = 1;
        claw_position = current_claw_pos;
```

```
        }



        Serial.print("Claw_pos: ");
        Serial.print(claw_position);
        Serial.print("\nCurrent_claw_pos: ");
        Serial.print(current_claw_pos);
        Serial.print("\nDifference: ");
        Serial.print(difference);
        Serial.print("\nSetpoint claw: ");
        Serial.print(Setpoint_claw);
        Serial.print("\n");
        }

        count2++;
        if (count2 > 10){   // reset at 11
        count2 = 0;
        average_dir = 0;
        }
  }
}


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ arm functions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void arms(){

  // Read Pot and determine direction
  Input_upper = analogRead(A0);
  Input_lower = analogRead(A1);

//-----------------------------------Limit Check-----------------------------------------//
  if ((Setpoint_upper > upper_min_lim) && (Setpoint_upper < upper_max_lim) &&   // limit check
        (Setpoint_lower > lower_min_lim) && (Setpoint_lower < lower_max_lim) &&
(Input_upper > 65)){

        average_upper = average_upper + Input_upper;
        average_lower = average_lower + Input_lower;

        if (arm_count == 9){
        average_upper = average_upper/10;
        average_lower = average_lower/10;
```

```
//-----------------------------------Upper-------------------------------------------------//

        if(average_upper > Setpoint_upper + error_upper) {          // increase angle
        myPID_upper_rev.Compute();
        upperDirection = 0;
        digitalWrite(UPPER_DIR, upperDirection);
        digitalWrite(UPPER_PWM, Output_upper);

        } else if (average_upper < Setpoint_upper - error_upper) {  // decrease angle
        myPID_upper_fwd.Compute();
        upperDirection = 1;
        digitalWrite(UPPER_DIR, upperDirection);
        digitalWrite(UPPER_PWM, Output_upper);

        } else {
        digitalWrite(UPPER_PWM, LOW);
        upperarmDone = 1;
        }

//-----------------------------------Lower-------------------------------------------------//
        if (average_lower > (Setpoint_lower + error_lower)) {          // increase angle
        myPID_lower_rev.Compute();
        lowerDirection = 1;
        digitalWrite(LOWER_DIR, lowerDirection);
        digitalWrite(LOWER_PWM, Output_lower);

        } else if (average_lower < (Setpoint_lower - error_lower)) {  // decrease angle
        myPID_lower_fwd.Compute();
        lowerDirection = 0;
        digitalWrite(LOWER_DIR, lowerDirection);
        digitalWrite(LOWER_PWM, Output_lower);

        } else {
        digitalWrite(LOWER_PWM, LOW);
        lowerarmDone = 1;
        }
        }

        arm_count++;
        if (arm_count > 9){
        arm_count = 0;
        average_upper = 0;
        average_lower = 0;
```

```
        }
  }
  if((upperarmDone && lowerarmDone) == 1){
        delay(2000);
  }
}

// Rotates pitch motor to desired pitch (Setpoint_pitch)
void pitch(){
  // Read Pot and determine direction
  Input_pitch = analogRead(A10);
  // Take average of the potentiometer reading for increased accuracy
  average_pitch = average_pitch + Input_pitch;

  // Take average of 6 potentiometer values
  if (count == 5){
        average_pitch = average_pitch/6;

        if(average_pitch < (Setpoint_pitch - error_pitch)){          //Increase angle
        stepper_pitch.rotate(10);
        } else if(average_pitch > (Setpoint_pitch + error_pitch)){  // Decrease angle
        stepper_pitch.rotate(-10);
        } else {
        stepper_pitch.stop();
        pitchDone = 1;
        }
  }

  count++;
  if (count > 5){   // reset at 6
        count = 0;
        average_pitch = 0;
  }
}

// angle to reading for upper arm
int angle_upper_to_reading(int angle) {
  Setpoint_upper = (0.0145*pow(angle,2) - 5.8438*angle + 1136.6);   // quadratic
}

// angle to reading for lower arm
int angle_lower_to_reading(int angle) {
  Setpoint_lower = (-0.0161*pow(angle,2) - 2.2429*angle + 550.38);  // quadratic
}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ gripper functions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


// Rotates spin motor to desired angle
void spin(){
  Input_spin = analogRead(A5);

  if (Input_spin < (Setpoint_spin - error_spin)){          // reading too low
          stepper_spin.rotate(-5);   // spin CW (from above)
  } else if (Input_spin > (Setpoint_spin + error_spin)){           // reading too high
          stepper_spin.rotate(5);           // spin CCW (from above)
  } else {
          stepper_spin.stop();
  }
}

// converts given pitch angle to equivalent potentiometer reading
double pitchAngleToPot(int angle){
  Setpoint_pitch = double((4.0227 * angle) + 93.03);
}

//Set the claw position so we know where it is
void initializeClaw(){
  //Read the limitSwitch as high or low

  Limit = digitalRead(LIMITSWITCH);

  //Close the claw until the limit switch is triggered
  //Now we know where the motor is and can use the encoder
  while(Limit != 1){ //Run until the limit switch is triggered
          //Set direction to reverse
          Limit = digitalRead(LIMITSWITCH);
          clawDirection = 1;       // close
          digitalWrite(CLAW_DIR, clawDirection);
          //Write speed between 0 and 255 to motor
          analogWrite(CLAW_PWM, 100);
  }
  stopClaw(); //Make sure claw is stopped

  //Open claw slightly
  clawDirection = 0;
  digitalWrite(CLAW_DIR, clawDirection);
```

```
  analogWrite(CLAW_PWM, 100);
  delay(2000);
  stopClaw();
}

void stopClaw(){
  //Set claw outputs to low
  digitalWrite(CLAW_DIR, LOW);
  digitalWrite(CLAW_PWM, LOW);
}

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ input function
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
void read_inputs() {
  while (Serial.available() > 0){
          data = Serial.read();
          if (isdigit(data) && (data != ' ')){
          command += data;
          }

          if (data == '\n'){
          if (input_count == 0){
          input1 = command.toInt();
          Serial.print("upper angle = ");
          Serial.print(input1);
          Serial.print("\n");

          } else if (input_count == 1){
          input2 = command.toInt();
          Serial.print("lower angle = ");
          Serial.print(input2);
          Serial.print("\n");

          } else if (input_count == 2){
          input3 = command.toInt();
          Serial.print("pitch angle = ");
          Serial.print(input3);
          Serial.print("\n");

          } else {
          input4 = command.toInt();
          Serial.print("gripper setting = ");
          Serial.print(input4);
          Serial.print("\n");
```

```
        theta3 = input1;
        theta2 = input2;
        theta4 = input3;
        Setpoint_claw = input4;

        angle_upper_to_reading(theta3);
        angle_lower_to_reading(theta2);
        pitchAngleToPot(theta4);

        upperarmDone = 0;
        lowerarmDone = 0;
        pitchDone = 0;
        clawDone = 0;
        clawState = 1;

        //restart = 1;
        input_count = 0;
        command = "";

        return;
        }

        command = "";
        input_count++;
        break;
        }
  }
}


void inverse_kinematics(){
  // y_coordinate, x_coordinate, and z_coordinate are input values for the desired positions of
the arm
  // a1, a2 and a3 are lengths of the arms sections
  // theta1, theta2, theta3, theta4 are measuring from current position
  // phi1, phi2, phi3, and r1, r2, r3 are calculated based on inputs

  a1 = 61;      //cm from rotating points
  a2 = 73;      //cm from rotating point to end effector
  a3 = 33;
  wx = 0;
  wy = 0;
  wz = -1;
```

```
  X0 = x_coordinate - a3*wx;
  Y0 = y_coordinate - a3*wy;
  Z0 = z_coordinate - a3*wz;

  // unused
  // theta1 = 180/PI * atan(y_coordinate/x_coordinate);

  r1 = sqrt((X0)^2 + (Y0)^2);

  r2 = Z0;

  phi2 = 180/PI * atan(r2/r1);

  r3 = sqrt((r1)^2 + (r2)^2);

  phi1 = 180/PI * acos(((a2)^2 - (a1)^2 - (r3)^2)/(-2*a1*r3));

  theta2 = phi1 + phi2;

  theta3 = 180/PI * acos(((r3)^2 - (a1)^2 - (a2)^2)/(-2*a1*a2));


  // calculates theta 4 constrained to "straight down"

  length1 = 180/PI * a1 * cos(theta2);
  length2 = r1 - length1;
  theta4 = 180/PI * asin(length2/a2);
}
```