BCIT SCHOOL OF ENERGY

Mechatronics and Robotics Engineering Technology

4491: Mechatronics Project

Final Project Report:
Jamie Boyd, Matthew Wonneberg

Produced By:	
Jamie Boyd	A00218257
Matthew Wonneberg	A01030942
Date:	May 18, 2022
For:	Isaiah Regacho

ACKNOWLEDGMENTS

We wish to acknowledge the instructors without whose imparted wisdom this project could not have succeeded. Greg Scutt taught us the msp430 processor programming used in this project with Andrew Friesen instructing us in the lab. Some of the msp430 code written was based on templates provided by Greg for ROBT 4451, e.g., initialization code for the Universal Asynchronous/Receiver Transmitter (UART) system. Code for increasing processor speed using the Universal Clock system was provided by Greg based on Texas Instruments example code. The kinematics equations for arm movements and the sinusoidal velocity profile we used in generating movements was covered by Isaiah Regaccho in ROBT 3341, who also taught us about DC motor control in ROBT 3351. The feedback control we used in correcting arm position during a movement was covered by Greg and Isaiah, and by Ali Hafez in ELEX 4336. The basics of 3D modeling was introduced to us by Sirine Maalej in MECH 1104. The NAND gate latch used in the emergency stop circuit was taught to us by Colin Shaw way back in ELEX 1215. Finally, we would like to thank our fellow mechatronics students for their support and encouragement.

ABSTRACT

Our team at Modular Robotics designed and built a SCARA (Selective Compliance and Assembly Robot Arm) for use by 1st year mechatronics and robotics students. To that end, our SCARA is compact, inexpensive, and replicable. It uses the same machinery and control techniques taught in the mechatronics program and is easy to control and program.

The SCARA has two arm segments, both 15.24 cm long, and a supporting tower (20 cm tall) that are all 3D printed from an original design. The angles of the arms are positioned by 118 rpm HD premium planetary gear DC motors with 1:71 gear ratios, powered by 12-volt 3-amp supply through a TB6612FNG dual motor driver. Position is acquired by quadrature encoders with resolutions of 0.1 degrees. A rail mounted linear stage actuator with leadscrew driven by a 12-volt stepper motor and A4988 stepper motor driver is attached to the end of the distal arm segment, providing movement in the Z direction for an attached tool. The motor drivers and encoders interface to an msp430f5529 microcontroller launchpad development platform. The launchpad and motor drivers are contained in a single plastic box. For safety, limit switches on the arms cut power to the motors, preventing the arms from extending past their range of travel. A physical emergency stop button is also provided. The final cost of the robot is $430 for acquiring all the parts and is over the proposed budget of $396 before shipping.

Firmware on the microcontroller contains routines for computing smooth, coordinated arm movements to given endpoints in the X, Y plane (joint interpolated movements), and moving the tool endpoint in a straight line or a circle. A sinusoidal velocity profile is used to limit jerk and increase accuracy. For a coordinated move, intermediate joint angle positions are pre-calculated at specific increments in time for a move while a feedback loop with proportional/Integral/ control is used to minimize errors in joint angle positions during a move. The firmware also contains commands for moving the stepper motor for the Z axis.

The firmware on the microcontroller takes commands for movements over the serial interface. Python code for a SCARA robot class is provided, making it easy to program arm movements. The free and cross-platform nature of Python means any student's computer can run the code to control the robot. A Python-driven graphical user interface with menus and buttons is also provided, making it even easier to control the robot when desired.

The SCARA meets the goals of being compact, inexpensive, and replicable while also being easy to use, program, and modify. Instructors that incorporate this robot into their program as a base to be used in labs or projects will be able to provide first year students hands-on experience with a robot that directly relates what they are learning about to a real-world application in a robot.

Contents

FIGURES

TABLES

EQUATIONS

1 INTRODUCTION

This document is a report for the Modular SCARA (Selective Compliance and Assembly Robot Arm) that has been designed and created as a requirement for the 4491 Mechatronics Project course. Our goal is to implement what our team has learned over the course of this diploma program into a project designed to showcase what we have learned as well as to push ourselves to gain new knowledge where we have less experience. This robot allows students in their first year to learn how to program movement of a robot, and to understand the hardware/circuit components that go into building a robot. The robot shows the students what is possible to create by the end of the program using parts and techniques that the students have learned or will learn.

In what follows:

In this report, we will provide a general system overview of the robot as well as the requirements that we designed within. We will then explore the details and purpose of each subsystem of the robot as well as their requirements, design, and implementation. Finally, our team will show how we verified the design to stay within the requirements as well as how we measured its performance and what tests we conducted to verify its operation.

2 SYSTEM OVERVIEW

The Modular SCARA is an educational robot for the BCIT Mechatronics and Robotics program. The SCARA uses the Texas Instruments MSP-EXP430F5529LP (MSP430) microcontroller to take commands from the user and data from the robot to compute the desired movement or response. The SCARA uses familiar limit switches and quadrature encoders to determine its position. The microcontroller outputs a Pulse Width Modulated (PWM) signal for each motor into a dual H-Bridge driver board to control the speed and direction of each motor. The SCARA is programmed to move coordinated in three different ways, joint interpolated movement, linear movement, circular movement. The arm at the base of the robot (L1) is directly driven through the bottom motor (M1) as they are directly coupled together. The second arm (L2) that is attached to L1 is driven through a belt and pulley system that is coupled to the top mounted motor (M2).

2.1 SYSTEM REQUIREMENTS

The first-year students only get access to the robots throughout the year for a small amount of time. The idea to introduce a robot that is small enough to fit in a student's personal desk space is the foundation of the structural design of this SCARA. To capture the student's interest in robotics, this SCARA is designed with the fundamental idea that the robot should be made up of as many components that they have learned or will learn about. We chose to use the MSP430 as the microcontroller for the SCARA knowing that they will be using it in the future and that this robot can showcase what is possible to do using it. The H-Bridge motor driver was also chosen because the students will understand its components and may have already learned about its functionality.

Modular SCARA requirements are as follows:

Table 1: System Requirements

Requirement	Detail
Mechanical Drive	Direct drive/Belt driven
Max speed	354 deg/s
Max reach	30.48 cm

User interface	Python GUI + Workspace.py
Motor Type	Brushed DC Motor
Z axis Drive	Stepper Motor
Microcontroller	MSP-EXP430F5529LP

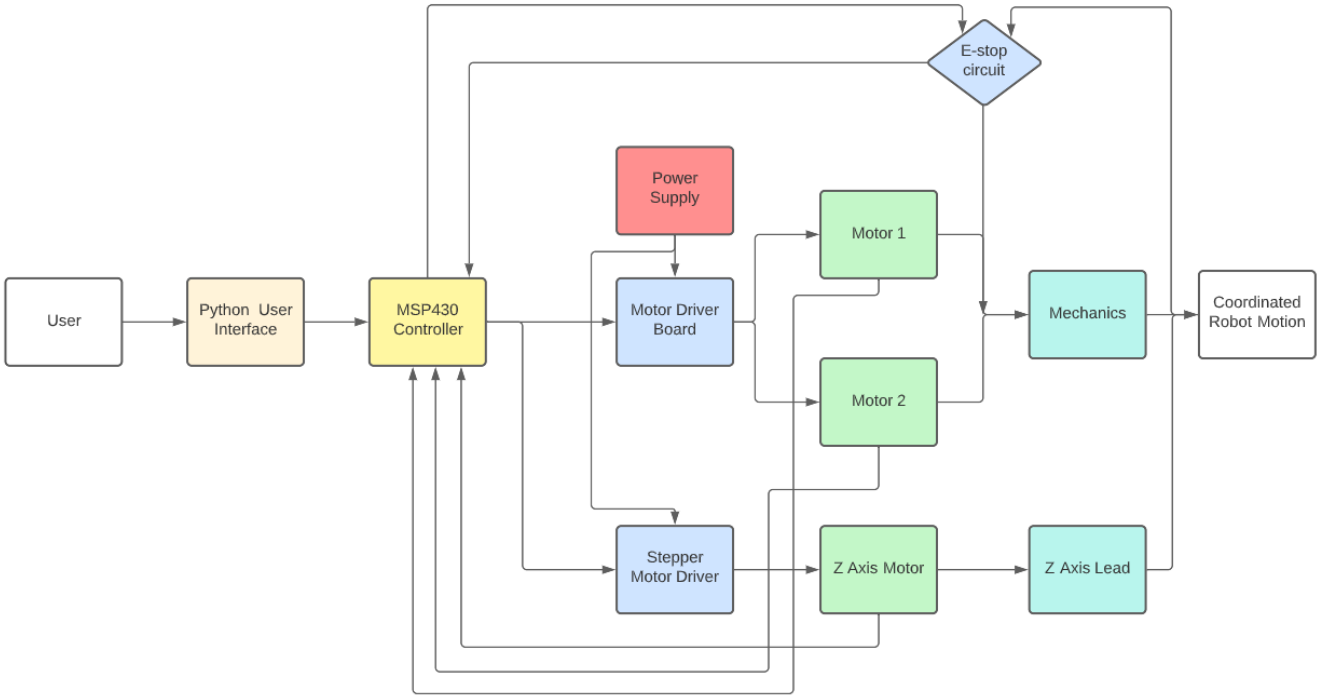

Figure 1: Modular SCARA Overview

3 STRUCTURAL SUBSYSTEM

The Structural Subsystem is made up of the two joints and a base. The parts of the structure were 3D printed using PLA to provide an inexpensive method to produce the SCARA while also allowing for students to design their own structural components and implement them. The 3D printing of the structure gave our team the ability to prototype parts and test them in a time efficient manner. An acrylic platform is used to mount the base and provides the stability needed for the robot to perform its full range of motion while looking more modern than using a wooden platform.

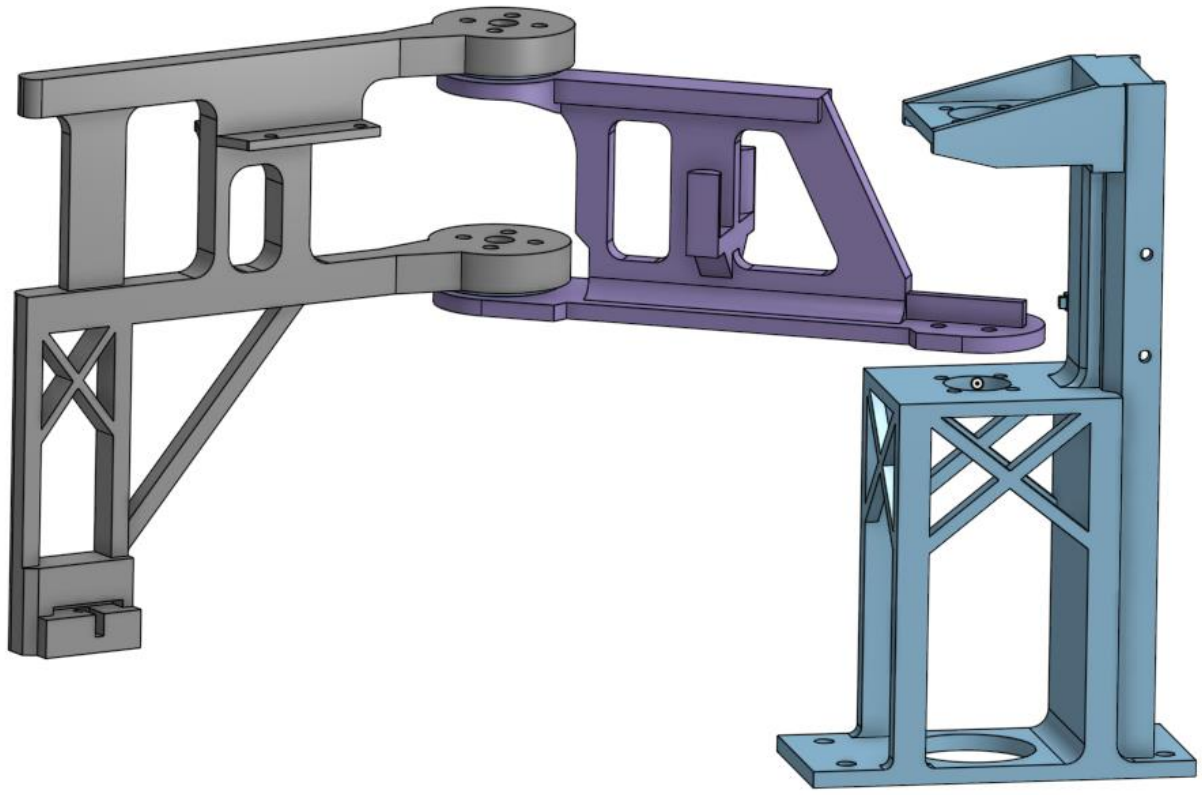

Figure 2 SCARA Structure

The base of the robot provides the mounting points for both arm motors and it is where the robot arm extends from. Limit switches are also mounted to the base to cut power to the robot if it extends beyond its safe limits.

Joint one (L1) shown in purple in Figure 2, is mounted directly to the bottom motor's shaft (M1) and extends from the robot base out into the work envelope. The limit switches that are mounted to the base only act on this arm. This arm length is 15.24 cm measured from the center point of the motor shaft hole to the center point of the hole for the arm coupling shaft at the opposite end of L1.

Joint two (L2) shown in grey in Figure 2, is mounted to the end of L1 through bearings which allow for independent movement of this joint. L2 is also 15.24 cm long measured from the center point of the L1 coupling shaft hole to the center of the Z-Axis mounting point. L2 has limit switch mounting points that are positioned so that if L2 violates its distance limit with L1, power to the motors will be shut off in an E-stop condition.

The Z-Axis stepper motor and lead mechanism is attached to the mounting point on the end of L2 close to the ground. A mounting fin for the tool is attached to the lead mechanism that allows for switching tools easily and adjusting the tool's angle.

3.1 Structural Subsystem Requirements

The design requirements for the structure of the SCARA were as follows:

- Non wooden platform
- 3D printed parts
- 15.24 cm Length arms (from center of rotation to center of rotation)
- Strength to withstand torque from motors twisting the structure
- Mounting points for limit switches and platform

3.2 Structural Subsystem Design

Designing of the structure was done using free online CAD software provided by onshape.com. This software was used because it allowed our team to collaborate on the CAD design and because it is a great alternative to SolidWorks as some team members did not have a valid licence. It provides full simulation of assembled parts and does not require a home computer because it is browser based.

3.2.1 Robot Base

The base of the SCARA is where most of the stress on the structure occurs because it is where the motors mount and where high torque must be withstood to minimize vibration and preserve accuracy. The counter torque created by the excitation of the motors produces a significant amount of torque applied to the base, which is why bracing was a necessary design feature of this part.

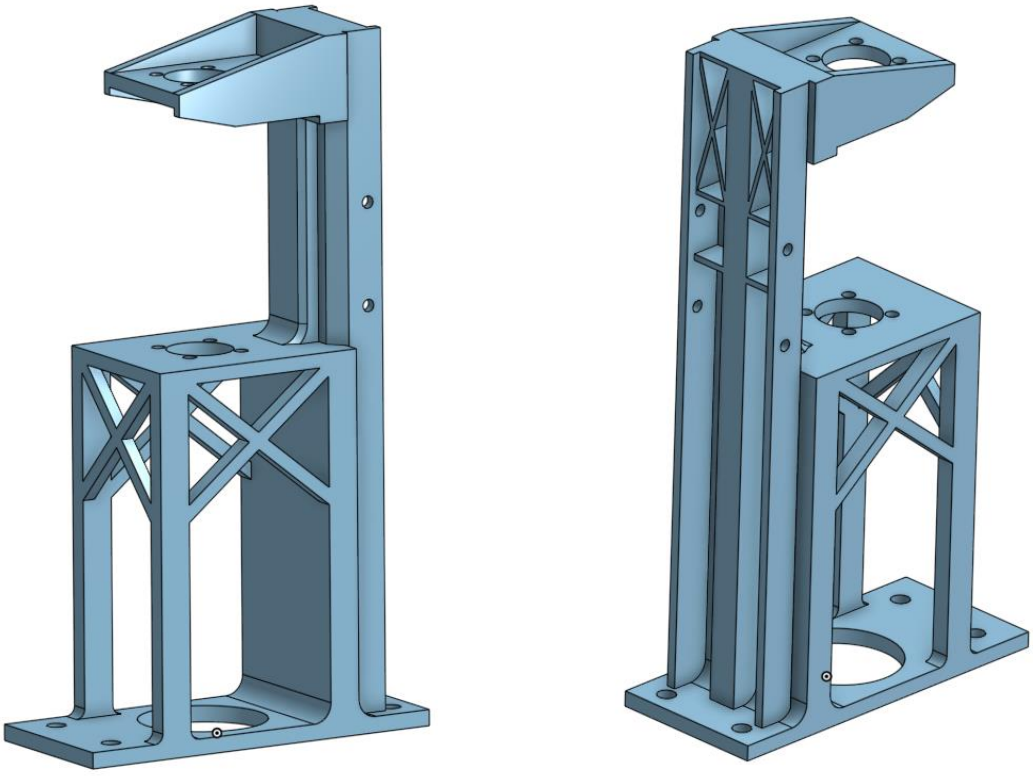

Figure 3: Robot Base (front and back view)

The base of the robot features cross bracing to maintain its rigidity when the motors move due to their high counter torque that acts in the opposite direction that the shaft spins. A slight chamfer is integrated into the spine of the robot shown in the front facing view of Figure 3 to allow for a larger range of motion of L1. Limit switch mounting points are positioned to either side of the spine and mounted as far back as possible to not interfere with the range of L1. These mounting points were extruded along the entire span of the spine to increase the rigidity. At the top of the base where motor M2 mounts struts are connected from the platform to the spine to prevent drooping due to the added weight of the motor.

3.2.2 Robot Joint One

This joint L1 of the robot arm mounts directly onto the M1 motor shaft and allows the arm to bend so that the end point of the robot can reach positions in between the robot base and the outer work envelope limit. The end of L1 is also where L2 is connected to, which is why it needs to be strong enough to support the extra weight of another arm and hardware placed at the end of it.

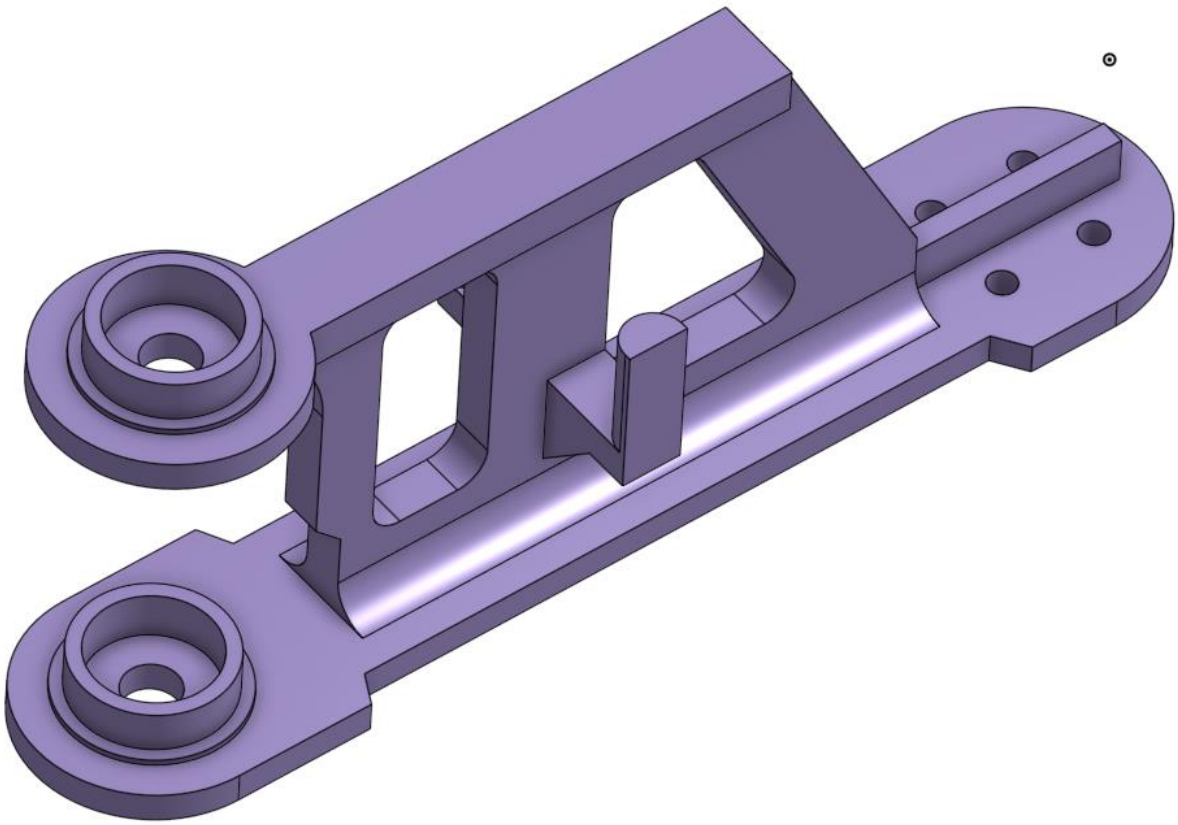

Figure 4: Robot Joint One (L1)

L1 is 15.24 cm long from the center of rotation to the center of the holes for the coupling shaft. It is designed with a spine that spans along the distance of the arm to maintain its rigidity. The spine was also designed with the requirement that it had to be thin enough to allow space for a belt that spans along the length of the arm. The spine is also necessary for the arm because a relatively large amount of weight will be creating a torque down due to the metal coupling shaft, hubs, and screws. The extruded circles at the end of the arm are where the bearings fit around to be used for L2 to mount to. The filleted part along the bottom of the arm is designed to add rigidity to the spine while also keeping the spine as thin as possible to allow for a greater range of L2. The indents along the bottom of

the arm are there to slightly increase the range of L1 from contacting the base at its full range of motion. The extrusions that come out of the spine on either side of the arm are tensioners used to hold the tension of the belt and reduce jerkiness in a move. Cut-outs in the middle of the arm are simply there to reduce the amount of PLA that is used during the printing process.

3.2.3 Robot Joint Two

This second joint of the arm is the mounting point of the Z-Axis lead where the tool is mounted to. This joint features an empty space that is designed to extend its range by allowing it to get very close with L1 without coming into contact with the belt.

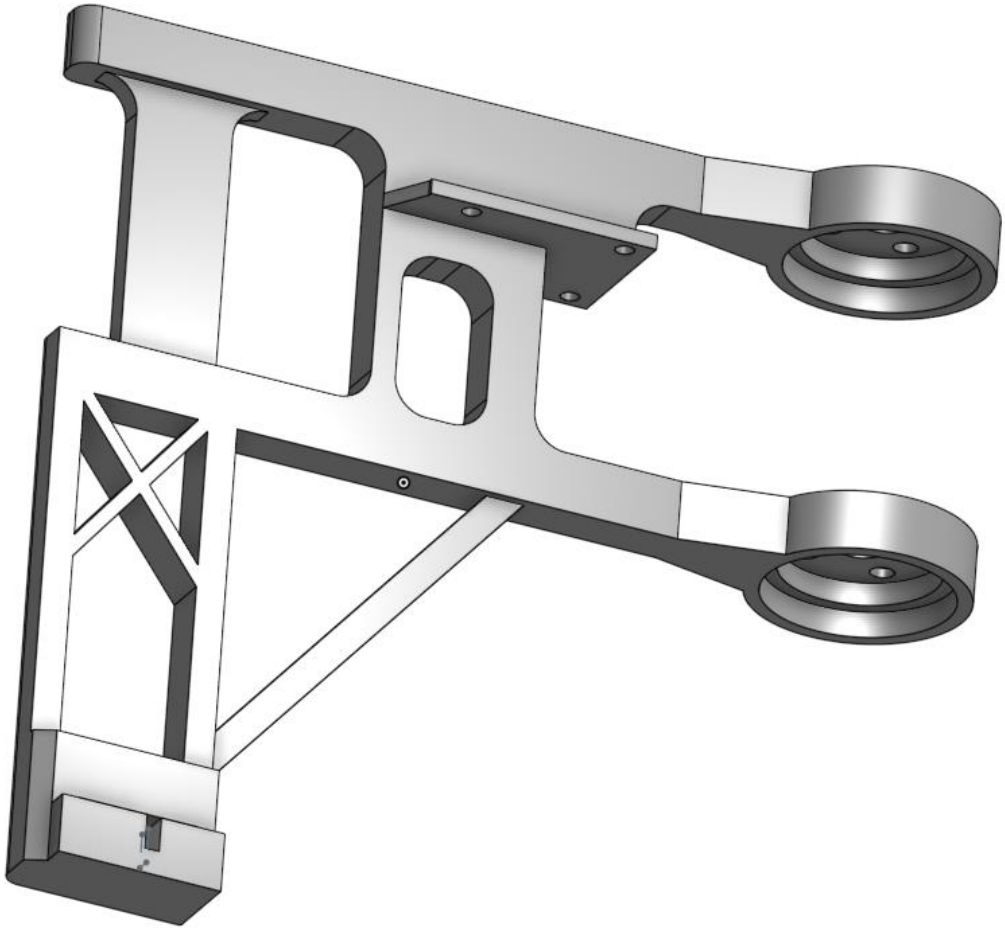

Figure 5: Robot Joint Two (L2)

L2 is 15.24cm long from the center of the coupling shaft holes to the center of the tool mounting point. Four holes are located around the coupling shaft hole to mount the hubs which connectL2 with the coupling shaft. The empty space on the right side of the arm in Figure 5 is there to allow room for the belt to operate while L2 comes close to L1. The extruded portion in the bottom of Figure 5 is to provide the Z-Axis lead a mounting point close to the ground because it is where it is most stable. At the very bottom of the arm, a slot is designed for the Z-Axis mechanism so that it can slide into place and stay secured. Limit switch mounting ridges that are positioned up high near the middle of the arm so that if one comes into contact with L1, power to the motors will be shut off and damage will be prevented.

3.3 Structural Implementation

The base, L1, and L2 along with the tool mount were 3D printed using an Ender 5 Pro 3D printer. The settings for printing these components are listed below.

Table 2: 3D Printer Settings

Baseplate Temperature	70℃
Nozzle Temperature	210℃
PLA Diameter	1.75mm
PLA Recommended Nozzle Temperature	190℃ - 230℃
PLA Recommended Baseplate Temperature	0℃ - 60℃

The reason for using a higher baseplate temperature than the recommended value for the PLA, is that less warping of the components raft was observed over the first few prints.

4 Mechanical Subsystem

The mechanical subsystem comprises all the components that allow for the transfer of movement from the motors to the structure. This involves the coupling hubs for each motor (M1 and M2), the belt and pulley system between M2 and L2, and the coupling shaft and hubs and bearings to connect L1 with L2.

4.1 Mechanical Requirements

Parts used to transfer movement meant that they had to be robust and as lightweight as possible to reduce the radial torque on the motor shaft caused by gravity. To reduce the play in the arms when they are assembled, finding hubs that would couple using the most surface was a necessary design requirement.

4.2 Mechanical Implementation

Through searching online for parts that would meet our criteria, we found that most parts could be ordered on servocity.com.

Table 3: Mechanical part supplier list

Parts Needed	Part supplier
2x 6mm D shaft hub	Servocity.com
3x ¼ inch shaft hub	Servocity.com
1x 3 inch long by ¼ inch shaft	Servocity.com
2x 3mm pitch pulley	Servocity.com

3mm pitch belt	Servocity.com
2x 25x36x7 shielded ball bearing	Bearingscanada.com
2x Micro Limit Switch 2 pack	Servocity.com
M4 x 0.7 bolts and nuts	Home Hardware
M3 x 0.5 bolts and nuts	Home Hardware

5 User Interface

The user interfaces with the MSP430 through a host computer that sends commands and receives results back over a serial interface. Commands include initializing positions and requesting moves of different kinds. Results sent back from the SCARA are angle positions of the two arm segments, and the position of the Z-axis. A Python SCARA class provides functions to move the robot, and a data structure that stores the state of the SCARA. These can be accessed from the Python shell. The SCARA class can be accessed from a control panel that also runs in Python, allowing visual control of the SCARA with no programming needed. Note that, while a Python interface is provided, interacting with the SCARA is not limited to Python, and any environment that can send and receive binary serial data (Igor Pro, MATLAB, or compiled C code, e.g.) could be used to create an interface for the SCARA.

5.1 User Interface Requirements

The user interface was needed to have several components so that the user could get as much access to controlling the robot as possible. This meant that a graphical user interface was needed to send commands to the robot for testing out new components or demonstrating its ability. A workspace that outlined code needed to create an array of motor position values was also needed if students were to program the robot. We also wanted to allow for fast communication between the user interface and the MSP430 so that data could be sent back and forth quickly if the user wanted to graph the robots position for a move or any other feature that could be added.

5.2 User Interface Design

The MSP430 launchpad has a USB/serial controller that connects to a USB port and presents as a standard serial port to the host computer. Thus, any operating system that supports serial communication, e.g., Microsoft Windows, Macintosh, or Linux, can be used with the SCARA. Note that the development environment, Code Composer Studio, does not need to be running on the host computer to interface with the microcontroller. Combined with the use of the open source and cross platform Python language, it is ensured that the SCARA has a low barrier to access for students.

Commands sent to the SCARA consist of a single byte corresponding to the number of the command, followed by any needed data, such as target positions, in serialized binary format. Conversion factors for degrees/count for L1 and L2 and mm/step for the Z-axis are stored in the SCARA class so only raw values need to be sent to the msp430.

Binary command interpreter code running on the msp430 initializes the serial port and constantly reads the port from an interrupt function, storing commands in a buffer. Commands are processed in the order they are received. This processing consists of recognizing the command from the transmitted index, parsing the data in the input buffer as appropriate for the command, and calling the corresponding movement functions. Commands that have results to send back to the host computer place the data in a results buffer, where an interrupt function handles the data transfer back to the host PC.

5.3 User Interface Implementation

The provided GUI allows users to input data that is relative to the type of movement that they want to perform. When a move command button is pressed, it only transmits the relative data of that move to the MSP430.

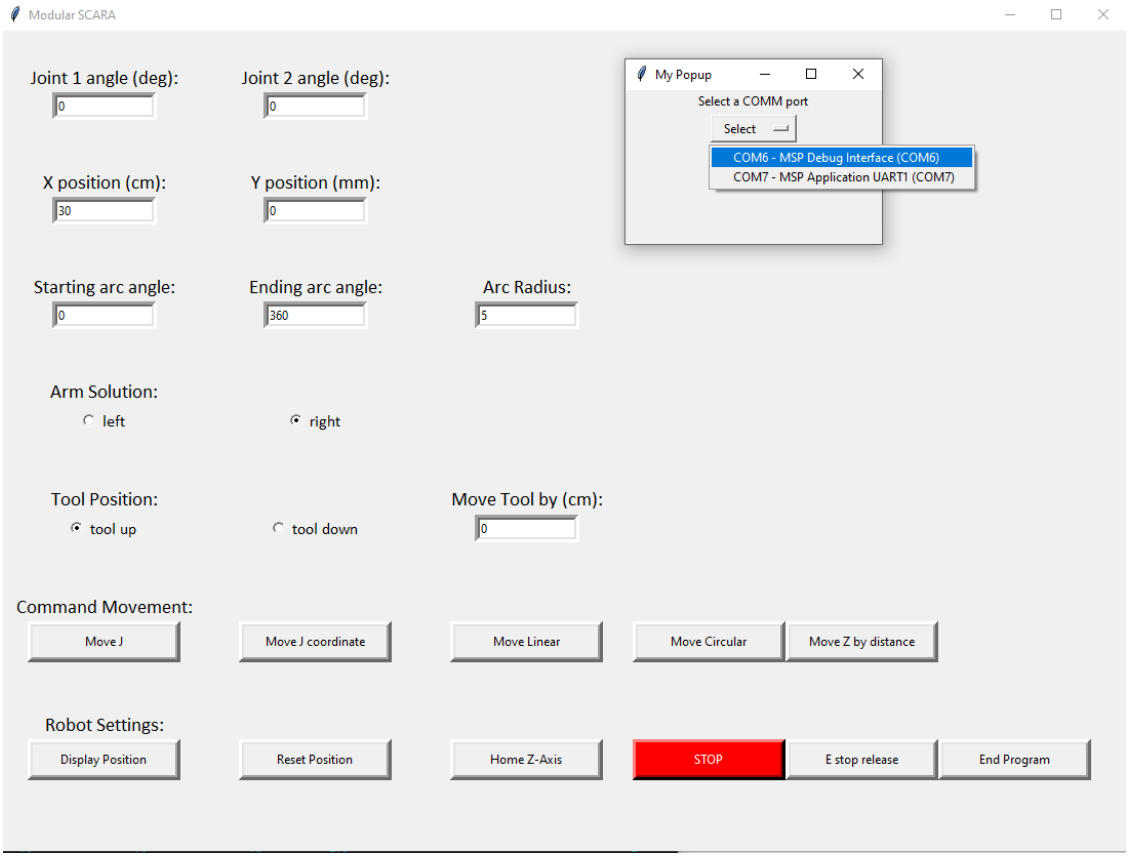

Figure 6: SCARA Graphic User Interface

```
#----------------------User created Move J ------------------------------------
def moveJProgram():
    screen.grab_release() # disables the GUI command screen
    global indexCMD
    indexCMD = 8
    # code to send both joint angles to the msp430
    screen.grab_set() # enables the GUI command screen


#--------------------- User created Move L -------------------------------------
def moveLProgram():
    screen.grab_release()
    global A_MAX_LINEAR
    #Xa, user provided
    Xa = 0
    #Xb, user provided
    #Ya, user provided
    Ya = 0
    #Yb, user provided
    deltaX =0
    deltaY =0
    deltaD = 0
    tInc = 0

    deltaD = math.sqrt(pow(deltaX, 2) + pow(deltaY, 2))
    timeForMove = math.sqrt((abs(deltaD)*2*PI)/A_MAX_LINEAR)
    w = (2*(PI))/timeForMove
    arrayLength = (timeForMove/T_UPDATE)+1

    for tInc in range(arrayLength):
        # now you must determine all of the x,y coordinates for every
        # index of the arrayLength throughout the line based on the time of the move.

        #The displacement along the line from 0 to D with respect to time is d(t).
        d = ((A_MAX_LINEAR * (tInc*T_UPDATE))/w -  (A_MAX_LINEAR)*(math.sin(w*(tInc*T_UPDATE)))/pow(w,2))

        #from the displacement along the line, and the known start and end coordinates of the line,
        # use this formula to compute the corrisponding (x, y) coordinate of the move
        x = Xa + (d*deltaX)/deltaD
        y = Ya + (d*deltaY)/deltaD

        # to do:
        # check that the x, y coordinates are ouside of the robot's base boundary.
        # compute the joint angles using inverse kinematics.
        # convert the joint angles from degrees to pulses of
        # the motor using the constant 9.48866 Pulses of the motor per degree of rotation


    for tInc in range(arrayLength):
        # in send over two array values in x,y coordinates if the array was computed successfully
        indexCMD = 9

    screen.grab_set()
```

Figure 7: Python provided outline for programming movement

The functions shown above in Python are fully commented to guide the student programming the movement what must be done to program the robot. Instructors using this robot have the option to omit comments or pieces of code to require more from a student to program movement of the robot.

6 ROBOT MOVEMENT FIRMWARE

The movement firmware consists of position acquisition of the joint motors, velocity profiling for coordinated motion, robot arm kinematics, and velocity signal transmission. Knowing the robots current position allows the program to calculate accurate and coordinated movement. This also provides the program with the ability to compensate for movement that is too fast or too slow. Controlling each motors velocity is key to achieving the movement that is desired. For this robot, we decided on using a PWM (Pulse Width Modulated Signal) to control the motor's velocity.

6.1 POSITION SIGNAL ACQUISITION DESIGN

Reading the motor's position is essential to designing an accurate robot. Maintaining an accurate position measurement throughout the robot's move is used to ensure that the robot hits its target position without jerkiness. Our team decided to use quadrature encoders mounted to the motor shafts to output encoder signals that we can use to determine the robot's position.

A quadrature encoder from a single motor outputs pulses on two channels A and B. The frequency of these pulses is proportional to how fast the motor is moving. These two channels from an encoder are input into the MSP430. Our program uses pins 2.4 and 2.5 for signal B and A respectively for motor 1. The MSP430 uses pins 2.7 and 2.6 for signal B and A respectively for motor 2. Port 2 is used for all the encoder inputs because they have a higher priority interrupt than other ports that we are using. This means that whenever a signal edge changes from the encoder, the program will stop what it is doing and run the code to update the position.

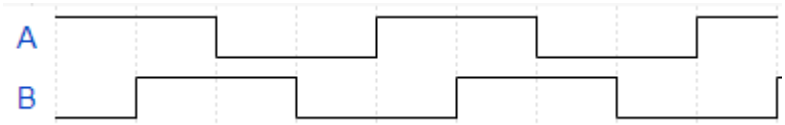

Figure 8: Encoder Pulse Waveform

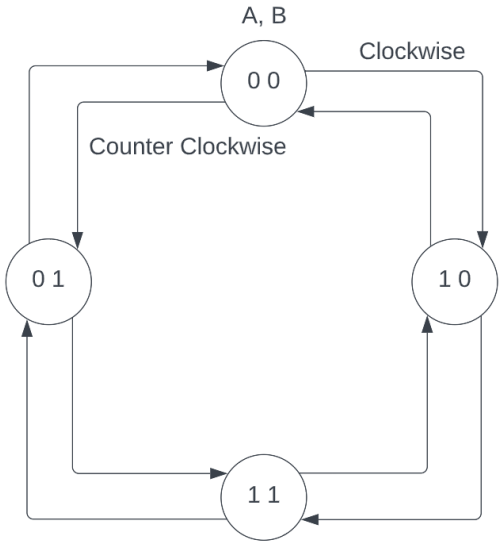

Figure 9: Quadrature Pulse diagram

To determine the motors position and direction from signals A and B shown in Figure 8, we use the XOR operator on the current B signal bit and the previous A signal bit. Figure 9 shows that if this operation results in a logical 1 or true condition, then the motor has moved in a counter-clockwise direction by one pulse. If this operation resulted in a logical 0, we would conclude that the motor has moved clockwise by one pulse. Depending on the direction of the updated pulse, 1 is added or subtracted from the variable that is holding the motor pulses in the program.

6.1.1 Position Signal Implementation

The position signal acquisition is implemented fully in the firmware in module quadEncDec.c.

```
#pragma vector = PORT2_VECTOR // PORT2_VECTOR is defined in msp430.h
 __interrupt void Port2_ISR1 (void) // Port 2 interrupt service routine
{

     if (currentState2 != CURRSTATE2){ // check if motor 2 interrupts went off
         currentState2 = (P2IN & 0xC0); // select the bits in reference to motor 2
         currB2 = (currentState2 & 0x40)>>6; // shift low bit into currB2 variable
         currA2 = (currentState2 & 0x80)>>7; // shift high bit into currA2 variable
         P2IES = ((currentState+currentState2) & 0xF0);// edge select for both motors
             if (currB2 ^ preA2){ // CCW
                 gPosCountL2--; // decrease the position by 1 pulse
                 dirStatus2 =1;
             }
             else{
                 gPosCountL2++;  // CW increase the position by 1 pulse
                 dirStatus2 =0;
             }
             preA2 = currA2; // only need to update previousA because we are not using preB
         P2IFG &= ( ~BIT6 & ~BIT7); // flags are cleared when exiting routine
     }
     else if (currentState != CURRSTATE1){ // check if motor 1 interrupts went off
     currentState = (P2IN &0x30); // select the bits in reference to motor 1
     currA = (currentState & 0x20)>>5; // /32
     currB = (currentState & 0x10)>>4; // /16
     P2IES = ((currentState+currentState2) & 0xF0);
         if (currB ^ preA){ // CCW
             gPosCountL1--; // decrease the position by 1 pulse
             dirStatus =1;
         }
         else{
             gPosCountL1++;  // CW increase the position by 1 pulse
             dirStatus =0;
         }
         preA = currA; // only need to update previousA because we are not using preB
     P2IFG &= ( ~BIT4 & ~BIT5 ); // flags are cleared when exiting routine
     }
     else
         P2IFG &= ~0xF0; // clear all flags if stuck


}
```

Figure 10: Motor position acquisition

6.2 KINEMATICS

Since the movement of the SCARA is only controlled using motor angles, and some of the user commands are relative to the (x, y) plane, it is necessary to create a function that can convert (x, y) coordinates into motor angles. This process of converting (x, y) coordinates into robot joint angles is called inverse kinematics.

There were two options that our team had to choose between when designing the robot that define what formula is used for inverse kinematics. These options were either choosing M2 with a large or small gear ratio. The first option was to use a motor for M2 that would have a small enough gear ratio so that joint L2 rotates by the same amount when the motor M1 moves joint L1. The second option is the one that our team ended up using and it is to choose a gear ratio for M2 that is large enough so that joint L2 does not rotate when M1 moves joint L1. Our team decided on choosing M2 with a high gear ratio so that we could guarantee that joint L2 would not be affected by movement of L1 so that the inverse kinematic formula would remain constant for any condition.

The following steps show how inverse kinematics are used to calculate the joint angles from a (x, y) coordinate of the robot.

1. Calculate the displacement "B" from the robot origin to the (x, y) coordinate.
2. Calculate the angle "a" from the robot origin to the (x, y) coordinate.
3. Calculate the angle "$\angle L_2$" that is formed by the triangle when formed using L1, L2, and "B" from step 1 using the cosine law.

$$\angle L_2 = \cos^{-1}\left(\frac{{L_2}^2 - B^2 + {L_1}^2}{-2 * B * L_1}\right)$$

4. If the arm solution is a left arm solution, calculate the joint angle for joint 1.

$$\theta_1 = a - \angle L_2$$

5. If the arm solution is a right arm solution, calculate the joint angle for joint 1.

$$\theta_1 = a + \angle L_2$$

6. Calculate the joint two angle θ_2

$$\theta_2 = \text{atan}\left(\frac{y - L1 * sin(\theta_1)}{X - L1 * cos(\theta_1)}\right)$$

6.2.1 KINEMATICS IMPLEMENTATION

The code for the inverse kinematics of the robot is held in the movement.c module.

```
volatile double xSqr = pow(toolX, 2);
volatile double ySqr = pow(toolY, 2);

volatile double holdSqr;

holdSqr = xSqr + ySqr;

B = sqrt(holdSqr);    // straight line distance from origin to (x,y) point
alpha = RadToPul(atan2(toolY, toolX));                                    // angle of B from origin to (x,y) point
beta = RadToPul(acos((pow(L2, 2) - pow(B, 2) - pow(L1, 2)) / (-2 * B * L1))); // cosine law to find beta

if (scaraState1->scaraPos.armSol == LEFT_ARM_SOLUTION) {                  // left hand solution
    angJ1 = alpha + beta;
    if (angJ1 > MAX_ABS_THETA1_PUL || angJ1 < -MAX_ABS_THETA1_PUL){       // switch solutions if the selected solution was impossible
        angJ1 = alpha - beta;
        scaraState1->scaraPos.armSol = RIGHT_ARM_SOLUTION;                // changed to Right hand solution
        armSolChange = 1;
        if (angJ1 > MAX_ABS_THETA1_PUL || angJ1 < -MAX_ABS_THETA1_PUL)
            exit =1;
    }
}
else if (scaraState1->scaraPos.armSol == RIGHT_ARM_SOLUTION) { // right hand solution
    angJ1 = alpha - beta;
    if (angJ1 < -MAX_ABS_THETA1_PUL || angJ1 > MAX_ABS_THETA1_PUL){  // switch solutions if the selected solution was not possible
        angJ1 = alpha + beta;
        scaraState1->scaraPos.armSol = LEFT_ARM_SOLUTION; // changed to left hand solution
        armSolChange = 1;
        if (angJ1 < -MAX_ABS_THETA1_PUL || angJ1 > MAX_ABS_THETA1_PUL)
            exit =1;
    }
}

angJ2 = RadToPul(atan2(toolY - (L1 * sin(PulToRad(angJ1))), toolX - (L1 * cos(PulToRad(angJ1))))); // calculate joint2 angle
if ((angJ2 < -MAX_ABS_THETA2_PUL || angJ2 > MAX_ABS_THETA2_PUL))
    exit = 1; // error if joint 2 angle is impossible to reach

if (exit == 0) { // if the solution is possible then update structure values
    *ang1 = angJ1;
    *ang2 = angJ2;
}
```

Figure 11: Inverse Kinematics function

6.3 Velocity Profile

When programming a robot to move, a velocity profile is necessary to follow so that the robot can move with smooth motion. We decided on using a sinusoidal velocity profile because it allows for gradual acceleration and deceleration which reduces jerk. A reduction in jerk is beneficial because it can reduce the strain on the physical robot parts and can reduce spikes in movement so that the robot is more precise.

$$a(t) = \frac{a_m}{w} * \sin(wt)$$

Equation 1: Acceleration formula

From the acceleration formula, we evaluate that the position profile would be calculated as follows at a time ‘t’ in the move over the period of ‘T’.

$$\theta(t) = \frac{a_m}{w}t - \frac{a_m}{w^2} * \sin(wt)$$

Equation 2: Position Formula

6.4 JOINT INTERPOLATED MOVEMENT DESIGN

Performing a joint interpolated move means that both joints move from their starting angle to their desired angle over the same period so that all joints finish moving at the same time. This is useful when moving the robot arm to an arbitrary point because it eliminates the possibility of jerk during the move by ensuring that one arm does not finish moving before the other.

Using formulas derived from a sinusoidal velocity profile, these are the steps taken to calculate the position values that when followed by both joints of the SCARA, result in joint interpolated movement.

1. Calculate the angular displacement of the move for each joint.
2. Determine which joint has the most displacement.
3. Calculate the period of the move using the joint that has the most displacement from the formula derived from the displacement equation.

$$T = \sqrt{\frac{D * 2\pi}{a_m}}$$

Equation 3: Period for move formula

4. Verify that the maximum velocity is not exceeded using maximum allowable acceleration and derived from the velocity equation and solving it for time T/2 when the velocity is at its maximum.

$$w_m = \frac{T * a_m}{\pi}$$

Equation 4: Angular velocity formula

5. If the maximum velocity is violated, then perform the following steps A and B.
   A. we recalculate the period for the move using the maximum velocity

$$T = \frac{2 * D}{w_m}$$

Equation 5: Period of move formula using maximum velocity

   B. Calculate the maximum allowable acceleration that will be used during the move

$$a_m = \frac{D * 2\pi}{T^2}$$

Equation 6: Acceleration formula using T and D

6. If step 5 is not performed, then assign the maximum acceleration allowed for the robot to the maximum acceleration allowed for the move.

7. Calculate the acceleration for the joint that has less displacement to move using Equation 4.

   At this point we possess the acceleration values that each motor will use for the move along with the period that it takes to perform the move.

8. Calculate the size of the array needed to store each position for every time that the motor's speed is updated.

$$position\ array\ length = \frac{T}{T\_update} + 1$$

9. Start a loop that performs the position calculation, on each motor from zero to the array length that contains the following steps.

$$\theta(t) = \frac{a_m}{w}\left(arrayIndex * T_{update}\right) - \frac{a_m}{w^2} * \sin\left(w(arrayIndex * T_{update})\right) + \theta_o$$

Equation 7: Motor Position Equation

10. Store the position result of Equation 7 for each motor in their respective position array's to be compared to in the control loop when performing the move.
11. Keep running the loop until the end condition is met.

6.4.1 Joint Interpolated Movement Implementation

The joint interpolated move code is partially shown below and is implemented in firmware in module movement.c.

```
timeForMove = sqrt((abs(deltaD)*2*PI)/A_MAX_PUL); // calculate period (T) for sinusoidal profile
w = (2*PI)/timeForMove;
vMaxMove = (2*A_MAX_PUL)/w; // calc the Vmax for the move

// check if you are within the velocity limit, otherwise you have to recalculate using the set max velocity
if (vMaxMove > W_MAX_PUL){
    vMaxMove = W_MAX_PUL;
    timeForMove = (2*abs(deltaD))/(W_MAX_PUL);
    aMaxMove = (abs(deltaD)*w)/(timeForMove);
    aMaxMove2 = (abs(deltaD2)*w)/(timeForMove);
}
else{ // if the velocity is within the limit, then calculate acceleration values for each motor
    aMaxMove = abs(A_MAX_PUL);
    aMaxMove2 = ((abs(deltaD2)*w)/timeForMove);
}

//-------------- assign variables-------------------
if (masterJoint == 1){ // if arm one moves further, then calculate the array length based on it
    w = (2*PI)/timeForMove;
    arrayLength = (timeForMove/T_UPDATE)+1;

    if (direction1 == 0) // relative to which joint moves the furthest
        aMaxMove = -1*aMaxMove;
    if (direction2 == 0)
        aMaxMove2 = -1*aMaxMove2;

    for(tInc; tInc<arrayLength; tInc++){
        posArray1[tInc] = RadToPul((PulToRad(aMaxMove)*(tInc*T_UPDATE))/w)  -  RadToPul(PulToRad(aMaxMove)*(sin(w*(tInc*T_UPDATE)))/pow(w,2))+gPosCountL1;
        posArray2[tInc] = RadToPul((PulToRad(aMaxMove2)*(tInc*T_UPDATE))/w)  -  RadToPul(PulToRad(aMaxMove2)*(sin(w*(tInc*T_UPDATE)))/pow(w,2))+gPosCountL2;
        if ((posArray2[tInc] > (posArray1[tInc]+ RELATIVE_THETA2_PUL)) || (posArray2[tInc] < (-RELATIVE_THETA2_PUL + posArray1[tInc]))){ // on the fly max theta2 value verification
            exit = 1; // error
            tInc = arrayLength;// exit the calculation
        }
    }
}
else if (masterJoint == 2){ // if arm two moves further
    w = (2*PI)/timeForMove;
    arrayLength = (timeForMove/T_UPDATE)+1;

    if (direction1 == 0)
        aMaxMove = -1*aMaxMove;
    if (direction2 == 0)
        aMaxMove2 = -1*aMaxMove2;

    for(tInc; tInc<arrayLength; tInc++){
        posArray2[tInc] = RadToPul((PulToRad(aMaxMove)*(tInc*T_UPDATE))/w)  -  RadToPul(PulToRad(aMaxMove)*(sin(w*(tInc*T_UPDATE)))/pow(w,2))+gPosCountL2;
        posArray1[tInc] = RadToPul((PulToRad(aMaxMove2)*(tInc*T_UPDATE))/w)  -  RadToPul(PulToRad(aMaxMove2)*(sin(w*(tInc*T_UPDATE)))/pow(w,2))+gPosCountL1;
        if ((posArray2[tInc] > (posArray1[tInc]+ RELATIVE_THETA2_PUL)) || (posArray2[tInc] < (-RELATIVE_THETA2_PUL + posArray1[tInc]))){ // on the fly max theta2 value verification
            exit = 1;// error
            tInc = arrayLength; // exit the calculation
        }
    }
}
```

Figure 12: Joint interpolated move function

6.5 LINE MOVEMENT

The move line command is called to move the tool center point along a straight line in the XY plane to an end point. The theory behind line movement is similar to joint interpolated movement with the difference that the velocity profile is applied to the tool center point instead of the motor angles. An option to select the arm solution is also provided when a move line command is requested.

The following steps are performed to create an array of motor positions that when applied to the motor, make the TCP move in a straight line.

1. Compute the current (x, y) position of the robot's TCP from the current encoder positions of each motor using forward kinematics.
2. Calculate the total displacement (D) from the user provided (x, y) end position and the current (x, y) position using the Pythagorean theorem.
3. Calculate the period for the move using maximum linear acceleration and Equation 3.

$$T = \sqrt{\frac{D * 2\pi}{a_{mLinear}}}$$

4. Calculate the size of the array needed to store each position for every time that the motor's speed is updated.

$$position\ array\ length = \frac{T}{T\_update} + 1$$

5. Assign the maximum linear acceleration to the maximum allowable acceleration for the move.
6. Start a loop that indexes from zero to the array length and contains the following steps.
7. Calculate the displacement from the starting position in terms of (x, y) with respect to the time that the robot is expected to reach this position.

$$d(t) = \frac{a_m}{w}\left(arrayIndex * T_{update}\right) - \frac{a_m}{w^2} * \sin\left(w(arrayIndex * T_{update})\right) + d_o$$

Equation 8: Motor Position Equation

8. Calculate the x and y coordinates using the current displacement in time.

$$x(t) = \frac{d(t) * (x_b - x_a)}{D} + x_a$$

$$y(t) = \frac{d(t) * (y_b - y_a)}{D} + y_a$$

9. Compute the corresponding motor angles for the (x, y) coordinate using inverse kinematics
10. Store the motor angles in terms of pulses in their respective position arrays that will be used in the control loop to be compared with the actual motor angles when a move is performed.
11. Continue running the loop until the end condition is met described in step 7.

6.6 Linear Movement Implementation

The linear move code is partially shown below and is implemented in firmware in module movement.c.

```
// calculate line displacements
deltaX = newLine.pB.x - newLine.pA.x;
deltaY = newLine.pB.y - newLine.pA.y;
deltaD = (sqrt(pow(deltaX, 2) + pow(deltaY, 2))); // pythagreom theorum for line distance in x,y

timeForMove = (sqrt((abs(deltaD)*2*PI)/A_MAX_LINEAR));
w = (2*(PI))/timeForMove;
arrayLength = (timeForMove/T_UPDATE)+1;
aMaxMove = A_MAX_LINEAR;

// fill the position and velocity array targets
for(tInc; tInc < arrayLength; tInc++){

    // calculate linear array in terms of d(t) and then fill X and Y positions
    d = ((aMaxMove * (tInc*T_UPDATE))/w -  (aMaxMove)*(sin(w*(tInc*T_UPDATE)))/pow(w,2));
    xHold = newLine.pA.x + (d*(deltaX)/deltaD);
    yHold = newLine.pA.y + (d*(deltaY)/deltaD);

    if ((abs(xHold*10) < INNER_CIRCLE_BOUNDS) && (abs(yHold*10) < INNER_CIRCLE_BOUNDS)){ // inner circle violation exit condition
        value = 1;
        tInc = arrayLength;
    }
    else{
        returned = scaraIkPulses(&posArray1[tInc], &posArray2[tInc], xHold, yHold, scaraState);
    if (returned == 0){
        if (armSolChange == 1){ // determine if a armSolution change was needed

            // hold values to be used for arm solution change
            armChangeStart.x = xHoldPrev;
            armChangeStart.y = yHoldPrev;
            armChangeEnd.x = newLine.pB.x;
            armChangeEnd.y = newLine.pB.y;
            holdLine = initLine(xHoldPrev, yHoldPrev, newLine.pA.x, newLine.pA.y, 0);//xb yb xa ya npts
            endLine = initLine(newLine.pB.x, newLine.pB.y, armChangeStart.x, armChangeStart.y, 0);

            scaraState->scaraPos.armSol = attemptedArmSolution; // return to original arm solution
            armSolChange =0;
            return(3);
        }
        xHoldPrev = xHold; // store the previous x,y values incase an arm solution change is needed
        yHoldPrev = yHold;

        if ((posArray2[tInc] > (posArray1[tInc]+RELATIVE_THETA2_PUL)) || (posArray2[tInc] < (-RELATIVE_THETA2_PUL + posArray1[tInc]))){ //  theta2 value verification
            value = 1;
            tInc = arrayLength; // exit loop due to error
        }
    }
    else{
        value = 1; // exit calculations if the move is not possible
        tInc = arrayLength;
    }
  }
}
```

Figure 13: Linear move function

6.7 CIRCULAR MOVEMENT

Circular movement of a robot is used in many applications where tracing a shape is required, which is why we decided to implement an option to perform this type of movement. This movement is calculated using the current position of the robot along with the radius, start angle and end angle that the user specifies. This type of movement is similar to a line move in that the variable displacement along the expected total displacement for the move is used to calculate the (x, y) coordinate that the robot should reach which is converted into motor angles.

These following steps are performed to create an array of motor positions that when moved through, result in a circular shape drawn by the tool center point.

1. Determine the starting angle, end angle and the radius of the arc.
2. Calculate the starting (x_a, y_a) coordinate of the arc, using the current pulse value of the robot's motors for each joint and forward kinematics.
3. Calculate the center (x_c, y_c) coordinate of the arc.
4. Calculate the end $(x_b,\ y_b)$ coordinate of the arc.
5. Calculate the displacement of the arc length.

$$D = radius * (\theta_f - \theta_i)$$

6. Calculate the time for the move using Equation 3.

$$T = \sqrt{\frac{D * 2\pi}{a_{mLinear}}}$$

12. Calculate the size of the array needed to store each position for every time that the motor's speed is updated.

$$position\ array\ length = \frac{T}{T\_update} + 1$$

7. Start a loop that indexes from zero to the array length and contains the following steps.
8. Calculate the displacement from the starting position in terms of (x, y) with respect to the time that the robot is expected to reach this position using Equation 8.

$$d(t) = \frac{a_{mLinear}}{w}\left(arrayIndex * T_{update}\right) - \frac{a_{mLinear}}{w^2} * \sin\left(w(arrayIndex * T_{update})\right)$$

9. Calculate the current angle displacement in time from the starting angle of the arc.

$$\theta(t) = \frac{d(t)}{D}\left(\theta_f - \theta_i\right) + \theta_i$$

10. Calculate the (x, y) coordinate in tine using the current angular displacement in time.

$$x(t) = radius * \cos\left(\theta(t)\right) + x_c$$

$$y(t) = radius * \sin\left(\theta(t)\right) + y_c$$

11. Compute the corresponding motor angles for the (x, y) coordinate using inverse kinematics.
12. Store the motor angles in terms of pulses in their respective position arrays that will be used in the control loop to be compared with the actual motor angles when a move is performed.
13. Continue running the loop until the end condition is met described in step 7.

6.8 CIRCULAR MOVEMENT IMPLEMENTATION

The circular move code is partially shown below and is implemented in firmware in module movement.c.

```
// find delta xy between the current coordinate of the robot and the arc center position
deltaXa = radius*cos(PulToRad(thetaStart));
deltaYa = radius*sin(PulToRad(thetaStart));

// find the center (x, y) coordiante of the arc
Xc = scaraStateSet.scaraPos.x - deltaXa;
Yc = scaraStateSet.scaraPos.y - deltaYa;

// find the delta xy between the center coordiante of the arc and the end coordinate of the arc
deltaXb = radius*cos(PulToRad(thetaEnd));
deltaYb = radius*sin(PulToRad(thetaEnd));

// find the end (x, y) coordiante of the arc
Xb = Xc-deltaXb;
Yb = Yc-deltaYb;

deltaD = radius*PulToRad(thetaEnd - thetaStart); // find the linear distance of the arc that is requested

// calculate the time for the move
timeForMove = (sqrt((abs(deltaD)*2*PI)/A_MAX_LINEAR));
w = (2*(PI))/timeForMove;

// calculate the arrayLength and set the acceleration to the maximum TCP acceleration
arrayLength = (timeForMove/T_UPDATE)+1;
aMaxMove = A_MAX_LINEAR;

// fill the position and velocity array targets
for(tInc; tInc < arrayLength; tInc++){

    // calculate linear array in terms of d(t) and then fill X and Y positions
    d = ((aMaxMove * (tInc*T_UPDATE))/w -  (aMaxMove)*(sin(w*(tInc*T_UPDATE)))/pow(w,2));

    // calculate the arc angle with respect to time, while accounting for direction
    if (direction == 1)
        arcAngle = ((d/deltaD)*(thetaEnd - thetaStart) )+ thetaStart;
    else
        arcAngle = ((d/deltaD)*(thetaStart - thetaEnd) )+ thetaStart;

    // calculate the (x, y) coordinate for the corisponding circle angle wrt time
    xHold = radius*cos(PulToRad(arcAngle)) + Xc;
    yHold = radius*sin(PulToRad(arcAngle)) + Yc;

    if ((abs(xHold*10) < INNER_CIRCLE_BOUNDS) && (abs(yHold*10) < INNER_CIRCLE_BOUNDS)){ // inner circle violation exit condition
        value = 1;
        tInc = arrayLength;

   }
   else{
       returned = scaraIkPulses(&posArray1[tInc], &posArray2[tInc], xHold, yHold, scaraState);

       if (returned == 0){
           if (armSolChange == 1){ // determine if a armSolution change was needed

               // hold values to be used for arm solution change
               armChangeStart.x = xHoldPrev;
               armChangeStart.y = yHoldPrev;

               scaraStateSet.scaraPos.theta2 = thetaEnd;
               scaraStateEnd.scaraPos.theta1 = holdArcAngle;

               scaraState->scaraPos.armSol = attemptedArmSolution; // return to original arm solution
               armSolChange =0;
               return(3);
           }
           xHoldPrev = xHold; // store the previous x,y values incase an arm solution change is needed
           yHoldPrev = yHold;
           holdArcAngle = arcAngle;

           // varialbe range joint angle 2 verification
           if ((posArray2[tInc] > (posArray1[tInc]+RELATIVE_THETA2_PUL)) || (posArray2[tInc] < (-RELATIVE_THETA2_PUL + posArray1[tInc]))){
               value = 1;
               tInc = arrayLength;
           }
       }
       else{// exit calculations if the move is not possible
           value = 1;
           tInc = arrayLength;
       }
   }
}
```

Figure 14: Circular move function

6.9 Arm Solution Violation

An arm solution violation occurs when the robot is not able to physically complete the line movement (either in a moveL or moveC command) to an endpoint for the user selected arm solution even though the endpoint is a valid place in the work envelope. The robot is programmed to automatically calculate two new lines using different arm solutions that connect to form the original line while performing a joint interpolated move to switch arm solutions after the first line.

A smaller line up to the violation point is calculated using the user selected arm solution. After this first line move is executed by the robot, it stops and performs a joint interpolated move to the same end point of the first line, only with the different arm solution.

After the joint interpolated move, the program calculates the second line that starts at the violation point and ends at the user specified endpoint using the arm solution opposite to the user selected one.

In an industrial setting this automatic response may not be desired because of space restrictions. For the use of teaching students, this automatic response has been added to demonstrate work envelope restrictions.

When an arm solution violation event occurs, the program will follow these steps.

1. Store the previous (x, y) coordinate of the move that the robot can reach
2. Store the user requested position of the Z-axis for the line move
3. Create a new structure “firstLine” that holds the initial (x, y) coordinate of the users requested move as well as the last (x, y) coordinate that was possible to calculate.
4. Create another new structure “secondLine” that holds the last possible (x, y) coordinate that could be computed and the (x, y) end point coordinate of the users requested line.
5. Calculate the expected motor positions in pulses for the structure “firstLine” that uses the user defined arm solution that was mentioned in step 2.
6. Perform the line move for the “firstLine” data
7. Move the Z-axis tool to the “UP” position
8. Perform a joint interpolated move to the current (x, y) coordinate that the robot is at using the opposite arm solution that the user defined.
9. Return the Z-axis tool to whatever position was requested by the user when performing the line movement.
10. Calculate the expected motor positions in pulses for the structure “secondLine” that uses the user defined arm solution that was mentioned in step 3.

6.10 Pulse Width Modulated Signals

The 12-volt DC motors that are used for joints L1 and L2, provided our team with two methods of controlling the motors speed that we had to decide between. The two options were either using analog control or Pulse Width Modulated (PWM).

A popular method for controlling the motors speed that we considered is analog voltage control. We could have made use of the Digital to Analog converters of the MSP430 to output an analog signal that would be conditioned through a circuit to output 0-12 volts to control the motors.

Another method of controlling motor speed using PWM signals is the option that our team decided on implementing. Using PWM signals involves controlling the duty cycle or high time of a square pulse wave set to a constant frequency. This PWM signal and other directional control signals are applied to an H-bridge circuit that converts the signal voltage level to drive strength of +12 volts and 0 volts while still maintaining the pulse wave. Because of the inertia of the motor when it is rotating, when this signal is applied at a suitable frequency, the PWM signal is effectively seen by the motor as an analog voltage between 0 and 12 volts that directly corresponds to the duty cycle percentage.

7 ROBOT CONTROLLER

In order to control the robot's position accurately, there are a few inputs that are necessary to perform this task. The robot's controller consists of a position setpoint provided by the user and position feedback from the motors. These two inputs are used to determine where the robot arm should be versus where the robot currently is. This difference is referred to as the error signal. Based on the error signal, the controller adjusts the speed of the motor's to either catch up with the position setpoint if the motors are moving too slow while the error is positive or slow down the motors to match the position setpoint if the error is negative.

7.1 ROBOT CONTROLLER DESIGN REQUIREMENTS

Successful control of the robot means that the robot should be able to reach the desired position with no overshoot or undershoot while also keeping the movement as smooth as possible. When the robot is moving to its target, the error signal should be designed to update at a rate fast enough to make use of the high-resolution encoders from the motors. The error signal must not be updated so fast that encoder pulses do not change much over a move because it would be harder to tune due to the small range of inputs. If the error signal is updated too fast, this would also require more memory to be used by the microcontroller to store all the expected positions for a move.

7.2 ROBOT CONTROLLER DESIGN

To allow for accurate movement of the motors, a signal update time of 5mS was determined to be a reasonable length. This time is based on the difference of encoder pulses that can be updated during this time while moving at full speed.

$$\text{pulses at full speed EQN} = \frac{354deg}{1\ second} * \frac{3415.92\ pulses}{1rev} * \frac{1rev}{360\ deg} * \frac{0.005seconds}{1update} = 16\ pulses\ per\ update$$

If the motors are operating at full speed, this range of possible pulses per update allows for more precise application of the PID variables because small changes of motor speed will be possible for a change in position. This is better than having a shorter update time because the PID variables become less effective and are also more difficult to determine due to a smaller change in position. An important factor to consider is that the full speed of the motor is only approached over the longest moves and is where the 16 pulses per update is accurate. Any shorter move will have a smaller number of pulses per update and may lead to inaccurate or jerky movement.

Another aspect that was in consideration when choosing an update rate, is the time taken to calculate the error and update the output signal for both motors after applying the PID control variables. This time was measured to be 2279 clock pulses or 0.11mS and minimal compared to the 100K clock pulses or 5mS time that therefore no adjustments were necessary.

Using timer B0 to control the rate that the update function is run at, an error value is calculated using the setpoint for the corresponding time in the move and using the current position that is provided by the motor encoders. Using the position error, summation error and rate of error, we multiply them by respective constants Kp, Ki and Kd then add the results up to create a percentage of the PWM signal that is sent to control motor speed.

7.2.1 SETPOINT

The setpoint for the move's final position is determined by the user input. From knowing the end position of the motors and the current position of the motors along with the type of move that the user has requested (joint interpolated, line, or circular), an array of setpoint positions is calculated for each motor that store the expected position in pulses for each 5mS increment of the move throughout the period of the move.

7.2.2 POSITION FEEDBACK

Position feedback from the motors is a necessary component in the controller to make sure that the robot follows the setpoint. Position feedback is automatically updated and stored in units of pulses for each motor, further described in section 6.1 (Position Signal Acquisition Design).

7.2.3 ERROR GENERATION

Every time that the 5mS update interrupt goes off, the difference between the setpoint position and the actual motor position is calculated for each motor. This difference in position is referred to as the error signal.

7.2.4 PI CONTROL

Proportional Integral control is what we are using to ensure accurate positioning while achieving the smoothest movement of the joints. Using the position error that is generated every update, a new variable is used that holds the summation of every position error of the move. The rate of change in position error is also stored in a variable that keeps track of the difference of position error between the current update and previous update.

Position error, summation error and rate of error are multiplied by a proportionality constant Kp, an Integral constant Ki and a derivative constant Kd respectively. We found that if we use the rate of error in the control loop, it decreases the accuracy of the response. The rate of error is included in the output calculation, but we set Kd equal to zero so that it does not have any effect. The gain constants were determined by testing through initially starting with zero for Kp, Ki, and Kd, then increasing Kp until the robot moved accurately to its position, then adjusting Ki to increase accuracy and decrease jerkiness in movement. We found that a Kp of 2.3 and a Ki of 0.2 yield the smoothest movement with the least overshoot.

7.2.5 OUTPUT SIGNAL GENERATION

The formula for the duty cycles of the PWM signals used to control the motors speeds combines the result of multiplying position error, summation error and velocity error by their respective gains. The value calculated represents a duty cycle percent between 0 and 100. This process is repeated every update time until the index that the move is finished is reached.

$$PWM\ duty\ cycle\ percent = kP * positionError + kI * positionErrorSummation + velocityError * Kd$$

Equation 9: PWM signal generation

7.3 ROBOT CONTROLLER IMPLEMENTATION

The controller code is fully implemented in firmware and can be found in the updateTimerB.c module.

```
//-------------- Motor 1 ------------------------
if (noMove1 == 0){// proceed if there is a move of the joint for joint 1

    posError1 = posArray1[updateIndex] - gPosCountL1; // motor 1 position error = setpoint - current position
    posError1Change = posError1 - prevError1; // delta position over time (position derivative)
    posError1Integral = posError1Sum; // summation of all errors throughout the move

    // apply gains to the output signal
    sendPWM = ((kP*posError1) + (kI*posError1Integral) + (kD*posError1Change));

    prevError1 = posError1; // store the current error for the derivative of position
    posError1Sum = posError1Sum + posError1; // update the sum of the errors for the integral of position

    if (sendPWM < 0){ // convert sendPWM to a posotive signal with a direction (dir1)
        sendPWM = sendPWM*-1;
        dir1 = 0; // ccw
    }
    if (sendPWM > MAX_PWM) // constrain max limits
       sendPWM = MAX_PWM;
    if (sendPWM> 0 && sendPWM <= MAX_PWM){ // min voltage condition cw
            if (sendPWM >= MAX_VELOCITY) // max speed limit
               sendPWM = MAX_VELOCITY;
    }

    if (dir1 == 1){ // send motor the speed signal based on direction
        mddInputCtrl(CTRLCW);
        timerA0DutyCycleSet(sendPWM); // send the previously pressed dutyCycle
    }
    else{
        mddInputCtrl(CTRLCCW);
        timerA0DutyCycleSet(sendPWM); // send the previously pressed dutyCycle

    }
```

Figure 15: PID control implementation

8 DRIVE SUBSYSTEM

The drive subsystem of the robot consists of the motors and the circuitry behind how they are controlled. We will discuss how the motors were chosen for our robot along with the power supply specifications that are needed. In Our design, we decided on using two HD planetary geared brushed DC motors with a 71:1 gear ratio. To control the power to the motors, we bought a dual H-Bridge driver, as opposed to designing the driver circuits ourselves so that we could direct more of our focus to other parts of the robot.

8.1 MOTORS

For this modular SCARA, we wanted to use motors that were small and inexpensive while still possessing the resolution required to move accurately. We decided on using brushed DC motors because they can give first year students an insight into what they will learn about in the next year.

We wanted to use inexpensive motor's while still getting the best performance out of them, so we decided to purchase ones with planetary gearing. The benefits of planetary geared motors are realized on the output shaft where a reduction in play is the increase in performance. This reduction in shaft play is especially important to minimize because any shaft movement will be magnetized at the end of an arm and can cause significant inaccuracy.

We decided on using motors that came with quadrature encoders as the method for receiving motor position feedback. To maintain a high accuracy of the robot, we purchased the motors with a specification of pulses per degrees that exceeds our desired resolution of the robot to ensure that our limiting factor was not the motors accuracy.

As we were selecting the motor gear ratio, we wanted to confirm that the rated torque was higher than the torque of the arm acting on the robot when moving in the XY plane. We also confirmed that the amount of torque produced by gravity acting on the motor shaft, was below the rated value of the motor for the maximum radial load.

8.1.1 Motor Axial Load

To ensure that the rated radial torque of the motor due to gravity acting on the arm, was not exceeded, we calculated the torque that is acting on the shaft. Noting that the belt that is attached to the second motor, we estimated that the belt would help reduce the torque on the bottom motor by 50% and apply it to the top motor shaft.

Using estimates of the arm mass when printed using 100% fill provided by our CAD program, and the mass of the mechanical parts such as hubs, provided by the component supplier, the following calculation was performed to find the torque acting on motor M1 if M2 is not connected.

$$T_g = T_{L1} + T_{coupler} + T_{L2} + T_{tool}$$

$$T_g = \left(m_{L1} * g * \frac{L_1}{2}\right) + (m_c * g * L_1) + \left(m_{L2} * g * \frac{L_1 + L_2}{2}\right) + \left(m_{tool} * g * (L_1 + L_2)\right)$$

Equation 10: Torque of entire arm on M1 due to gravity

From this equation, we calculated that the torque due to gravity on the arm is 1.84 Nm. This is about half of the maximum radial torque of the motor shaft, which is 3.5 N.

Our estimate does not account for the torque acting in the opposite direction of gravity due to the belt, but we estimate that our calculation would reduce the torque due to gravity by around 50% which would be applied to the other motor M2.

8.1.2 Motor Torque Required

Because the motors are working in the XY plane, the peak torque of the motor needs to be above the torque acting on the arm due to angular acceleration. In the calculation we were able to treat the mass of both arms combined with the coupling shaft parts at one point mass in between L1 and L2 because of their very similar mass and the effect that moving one mass away from the mid point of L1 while moving the L2 mass toward the base of L2 cancels out. No friction due to torque is considered because L1 is directly mounted to the motor shaft, and L2 is driven through a pulley system that is on bearings.

$$T_P = J_{arm} * a_m + T_{friction}$$

$$T_{PM1} = \left(\frac{1}{12} * m{L_1}^2 + m\left(\frac{L_1}{2}\right)^2\right) * a_m + T_{friction}$$

$$T_{PM2} = \left(\left(\frac{1}{2}ma^2\right) + \left(\frac{1}{12}mL^2 + m\left(\frac{L}{2}\right)^2\right) + (mL^2)\right) * a_m + T_{friction}$$

Equation 11: Peak Torque Non-Geared

From this equation, the peak torque of the arm when driven at maximum acceleration is 0.095 Nm, but because we wanted to use a high gear ratio for the SCARA, we decided on using a motor with a gear ratio of 71:1 revolutions of the motor shaft to revolutions of the output shaft. The equation of peak torque with a gear ratio is shown below.

$$T_{PM1} = \frac{\left(\frac{1}{12} * mL_1{}^2 + m\left(\frac{L_1}{2}\right)^2\right) * a_m + T_{friction}}{N}$$

$$T_{PM2} = \frac{\left(\left(\left(\frac{1}{2}ma^2\right) + \left(\frac{1}{12}mL^2 + m\left(\frac{L}{2}\right)^2\right) + (mL^2)\right) * a_m + T_{friction}\right)}{N}$$

Equation 12: Peak Torque Geared

The peak torque of the arm acting on the motor when accelerating at maximum acceleration of 177 degrees per second squared, is 0.001 Nm. This value is far below the rated torque of 0.1 Nm, so we can use this motor for our application.

8.2 Power Supply

The requirements for the power supply of the SCARA are based around the current requirements of the motors used for joints L1 and L2. We knew the motors would use a significant amount of current due to the high gear ratio of the motors compared to the insignificant amount drawn from the devices like the motor driver and stepper driver.

Before ordering the motors, the amount of current that was estimated was approximately the same amount as the Maximum No-load Current rating of the motors of 0.53 amps. We estimated this because of the small amount of torque that acts on the motors due to the weight and friction of the joints when moving in the XY plane Calculated through Equation 10 which is similar to having no load.

Through measuring the current of the motors when an arm is attached, we were able to verify that the current drawn by one motor at constant speed was approximately 0.51 amps and because of the sinusoidal velocity profile, current drawn when accelerating quickly did not change very much.

For the power supply, we use a 12-volt 3-amp power brick that our team had already acquired. This gives our power supply an ability to provide 3 times as much current that is expected to be drawn by the system.

8.3 Dual H-Bridge Driver

The MSP430 can output PWM signals and direction signals to its ports to be used by a motor driver board. A driver board is required to turn these PWM signals into full strength levels of 12 volts with the ability to supply the maximum current that is drawn by a motor of 0.53 amps. The driver is also required to perform directional control of the motor.

We were able to find a dual H-Bridge driver “TB6612FNG” on pololu.com that met our specifications of allowing a maximum motor supply voltage of 15 volts and allowing an average current of 1.2 amps with a peak of 3.4 amps for one motor channel.

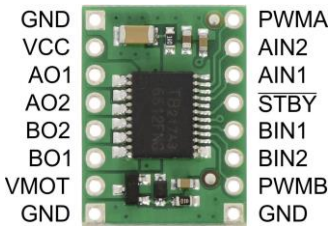

Figure 16: Pololu TB6612FNG

8.3.1 H-Bridge Operation

The H-Bridge is an electrical circuit that is used to provide directional control of a motor as well as increasing the signal voltage to a level that can drive the motor.

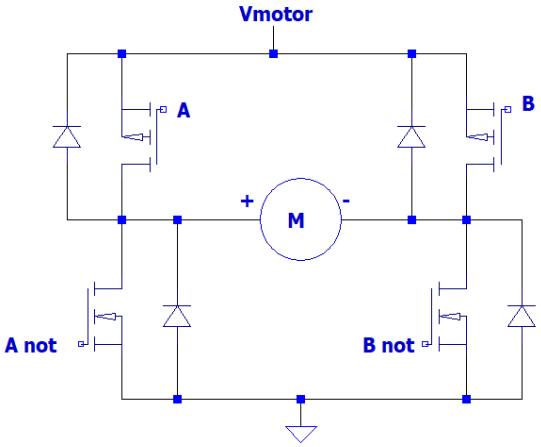

Figure 17: H-Bridge Schematic

Table 4: H-Bridge Control Table

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Table 3 portrays how using the corresponding signal levels for IN1, IN2, PWM, and STBY will result in either short brake, CCW (counter clockwise) operation, CW (clockwise) operation, or a stop function of the motor.

The Pololu dual H-Bridge driver (figure 8) takes in a PWM signal and two directional signals IN1 and IN2. The driver then applies the PWM signal to whichever directional signal is high and uses IN1 and IN2 to control the transistors A and B of the H-Bridge (Figure 9). The result of this H-Bridge circuit is getting a drive strength PWM output that is applied to either the positive side or negative side of the motor depending on the direction specified to control the speed.

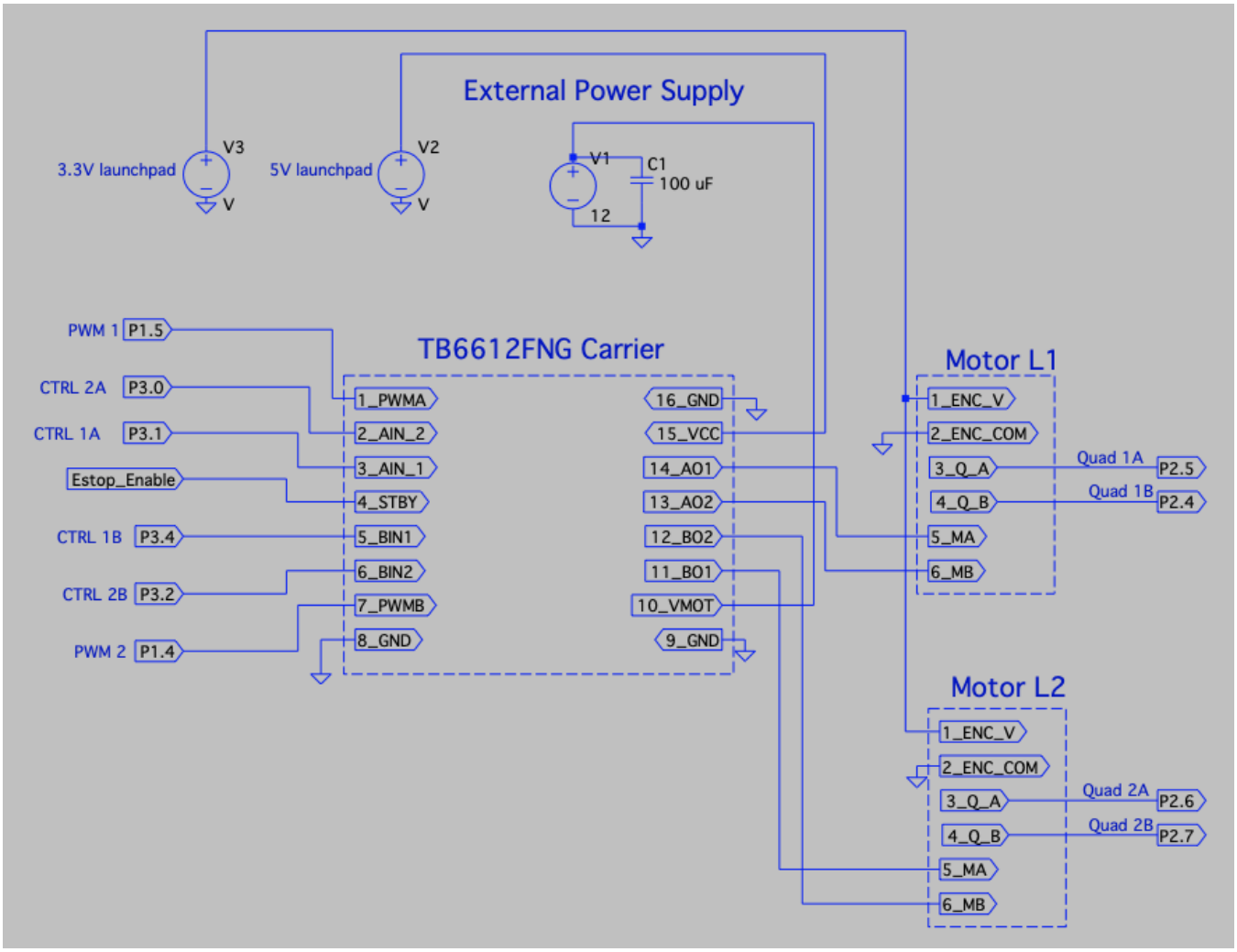

Figure 18: Subcircuit for joint motors and encoders

8.3.2 Emergency Stop

In the interest of safety, a way to stop all movement of the arms and Z-axis is needed if, for example, something gets caught in the arms, or if a user-programmed move drives the arms against each other or drives the tool into the workspace. A multi-pronged approach to emergency stopping is provided for the SCARA.

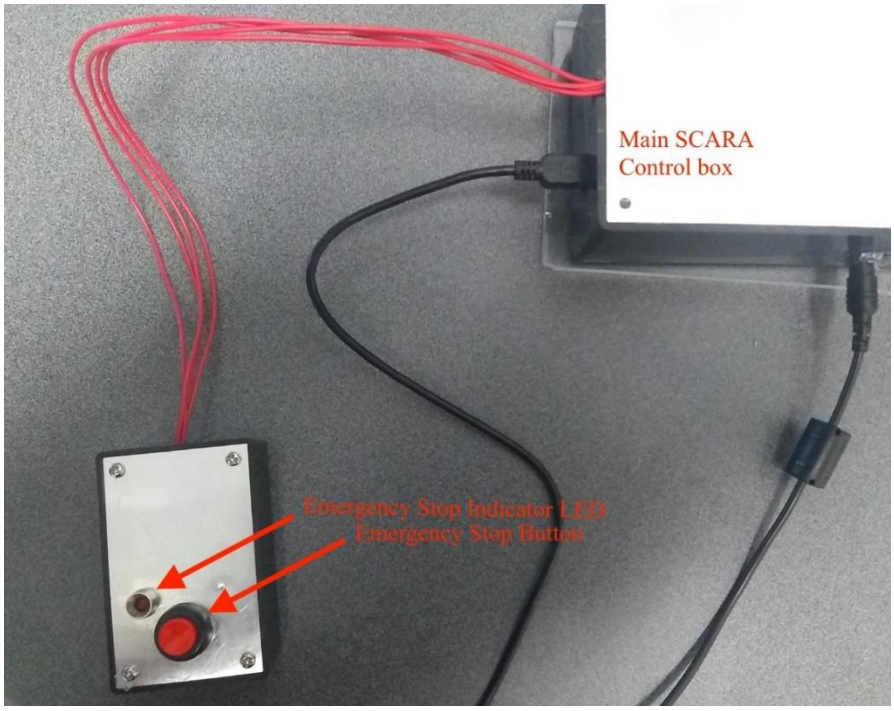

Figure 19: Emergency stop button and indicator light in separate box on tether

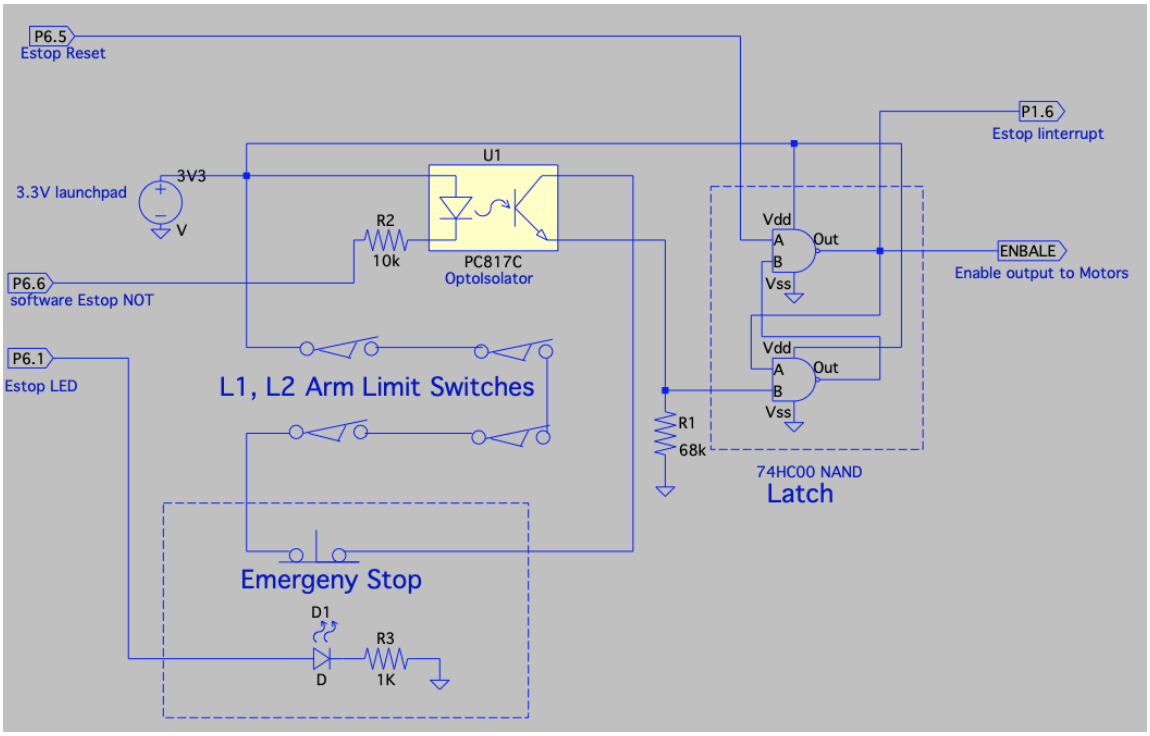

Figure 20: Subcircuit for emergency stop

The emergency stop system can be activated in any one of several ways. A separate box, on a tether from the main SCARA control box, contains an emergency stop button and an indicator light (figure 19). Pressing the button, however briefly, will engage the emergency stop system. Four limit switches are mounted on the SCARA, two on each arm segment. If the arms are commanded to move past their range of travel, they will press against one of the limit switches and activate the emergency stop. Finally, an emergency stop signal can be sent from a button on the user interface (see section 5). In all cases, a single momentary activation causes the motors to immediately stop and for the system to be held in the stopped state until it is reset.

The SCARA emergency stop circuit is based on a series of normally closed switches. If any one of them opens, the emergency stop condition is triggered. Logic-level voltage (3V3) from the msp430 is routed through the four arm-mounted limit switches, through the separate physical emergency stop button, and then through an opto-isolator kept in the open state by a low signal from an msp430 output signal (figure 20). Setting this signal high from the microcontroller will thus open the circuit. Finally, the circuit leads into a latch made from two NAND gates. The output of the latch is set low, and held low until reset, after even a momentary interruption in the stop circuit. The output of the latch goes to the motor drivers where it acts as an enable signal (figure 18; figure 22), and also to an interrupt-enabled input pin on the msp430.

An interrupt running on the microcontroller (in eStopLimitSwitch.c) monitors the output of the latch. Upon being triggered by a high-to-low transition, the code turns on the emergency stop indicator LED, and sets a global variable in the code. Functions implementing movements of the arms or Z-axis check this variable periodically and break early if the variable is set.

Once the latch output has been set, it must be reset by a high-low-high pulse on the reset input while the emergency stop input (the set input) is held high. This is accomplished by an output signal from the msp430, at which time the emergency stop LED is also turned off.

The SCARA class in SCARA.py contains functions that call the firmware emergency stop and emergency stop reset functions. An emergency stop can be sent from the Python command line, or from the SCARA control panel. Similarly, the reset function is also available from the control panel. Note that the firmware reset function is needed for all emergency stops, not just those triggered from software.

9 Z-Axis Subsystem

The Z axis of the robot is considered its own subsystem because it has its own driver, motor, and carriage. The Z-axis mechanism is connected to the end of the robot's L2 joint, where a motor can move the lead up and down in the Z-axis. The robot's tool is attached to the end of the Z-axis joint. The lead can move because of a stepper motor that is powered using 12 volts. The stepper driver carrier converts 3.3-volt signal pulses outputted by the microcontroller into 12-volt pulses of the correct phase to drive the motor.

9.1 Z-Axis Design Requirements

The first requirement of the Z-axis subsystem was that it needed to be lightweight. We required that the system had to be light enough to ensure that the radial load specifications of the motor shafts were not exceeded since the Z-axis is mounted on the end of joint L2 and can create a large torque by small changes in weight.

The second design requirement was that the motor had to be easy to control. Because we only needed to control the motor in two directions by a set position, using a closed loop position control system would over complicate the design and would not be necessary.

The Z-axis of the robot had to be able to move a minimum of 2 cm, which would give our team enough space to design around this length to build a usable Z-axis tool.

The last design requirement is that the Z-axis had to run from our 12-volt power supply brick and use a minimal amount of current because there is not a significant amount of weight that the motor needs to move.

9.2 Z-Axis Design

Starting with the design requirement that the Z-axis subsystem should be easy to control because of its simple movement, A small stepper motor was first considered. Because there is no need for position feedback, as we can keep track of the position based on the number of pulses that are sent to the motor. No control loop is needed which simplifies the process.

From the distance of travel required by the axis, we were able to find some stepper motors with a built-on lead of 8 cm. This was the option that our team decided to go with because of the small weight of the mechanism and because the structure was made of metal which is rigid enough to hold a tool along with fulfilling the other design requirements.

Figure 21: Z-axis stepper motor lead screw

The motor-lead mechanism that is used in the SCARA is a 2 phase, 4 wire, stepper motor lead screw slide table with a linear guide and can travel 80 mm. The motor is powered using 12 volts.

To drive the motor using 12-volt pulses in a 2-phase fashion, we decided to use the A4988 stepper motor driver from Pololu. This motor driver meets the motor's specifications, as it can provide 12 volts to the motor. Another benefit of using this motor driver is that it takes care of controlling the output to the correct phase of the motor, while the MSP430 must only send steps of movement along with a direction signal. This decreases the complexity of this system making it easier for students to understand and allowing our team to focus on other components of the robot given the limited time frame.

9.3 Z-Axis Firmware Implementation

The msp430 firmware (ZaxisCtrl.c) provides inputs for step, direction, and enable to the A4988 motor driver used to control the Z-axis. In addition, the Emergency stop signal (section 8) is applied to the reset pin. The step signal is generated by a timer A on the msp430; the associated interrupt function counts the pulses as they are applied to the stepper motor. There is no position feedback from the stepper motor, so the step count is the only positional information for the Z-axis. The direction signal (up or down) is set by the msp430 before starting the timer that generates the step signal.

9.4 Z-Axis Hardware Implementation

Normally one set of coils in the stepper motor is always energized at any given time, even when a move is not in progress. The current involved can cause the coils to heat up, although the constant energization does act to hold the stepper motor in position against any opposing forces. For the SCARA, the weight of the tool is small enough and the gear ratio large enough that the position of the tool is not anticipated to change in the absence of coil

energization. Therefore, a signal from the msp430 is directed to the enable pin of the driver. The signal is kept high until a movement is about to be started and is set low again when a movement is completed.

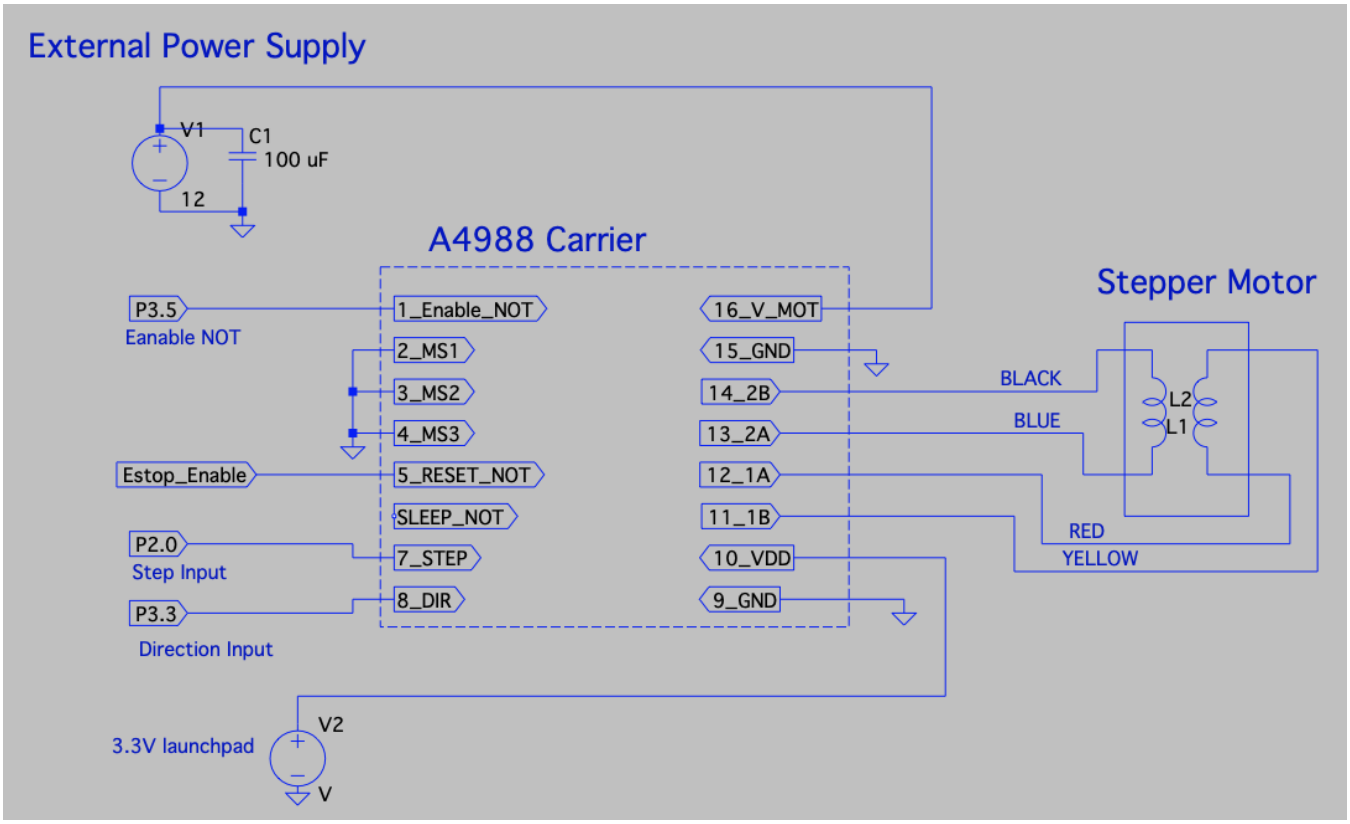

Figure 22: Z-Axis Circuit Schematic

10 BUDGET

One goal of this project was to produce a robot that would be affordable enough to allow for multiples of it to be built for a classroom setting. Keeping this goal in mind, we designed the robot using simple components that not only kept the cost down, but also give the students who will be operating this robot a chance to recognize the components that went into this project that they have learned about. Keeping the cost of this project down was another reason as to why our team did not opt to buy more precise motors, or a more capable microcontroller.

Table 5: Bill of Materials

16-May-22

Bill of Materials for one unit

Description	Supplier	Part#	Unit Price	Quantity	Price	Notes
118 RPM HD Planetary Gear Motor w/Encoder	Servo City	638324	$77.39	2	$154.77	One for each of L1 and L2
TB6612FNG Dual Motor Driver Carrier	Pololu	713	$12.84	1	$12.84	L1 and L2 motor driver
Stepper motor with lead screw plus carriage	Yanmis	Yanmiskysuwcgtz8	$23.80	1	$23.80	Z axis motor
A4988 Stepper Motor Driver Carrier	Pololu	1182	$10.97	1	$10.97	Z-axis motor driver
3mm HTD Pitch Plastic Hub Mount Timing Belt Pulley	ServoCity	3402-0014-0036	$5.15	1	$5.15	L2 belt drive
3406 Series 3mm HTD Pitch Timing Belt	ServoCity	3406-0015-0005	$38.69	1	$38.69	L2 belt drive
1309 Series Sonic Hub (1/4" Bore)	ServoCity	1309-0016-0250	$9.02	3	$27.05	Hubs for L1 and L2 shaft
1309 Series Sonic Hub (6mm D-Bore)	ServoCity	1309-0016-1006	$9.02	2	$18.03	Hubs for Motor shafts
0.250" (1/4") x 3.00" Stainless Steel Precision Shafting	ServoCity	634162	$1.79	1	$1.79	L2 belt drive shaft
Snap-Action Switch with 16.3mm Roller Lever	Pololu	1404	$1.74	4	$6.96	Limit switches for L1 and L2
msp-EXP430F5529LP	Digikey	296-36506-ND	$21.65	1	$21.65	microcontroller development board
shielded ball bearings 25x37x7	Bearingscanada	61805-2Z	$16.31	2	$32.62	bearings for L2 joint rotation
project box LX-642	Lee's Electronics	10363	$6.99	1	$6.99	for microcontroller and motor drivers
project box LX321	Lee's Electronics	10802	$3.50	1	$3.50	emergency stop
protoboard 170	Lee's Electronics	1058	$3.60	2	$7.20	microcontroller mount
plastic nylon spacers (pkg of 10)	Lee's Electronics	6991	$1.50	1	$1.50	microcontroller mount
push button momentary switch	Lee's Electronics	3277	$1.35	1	$1.35	emergency stop
led 5mm with holder cove	Lee's Electronics	5610	$1.20	1	$1.20	emergency stop
double row headers	Lee's Electronics	2147	$1.20	2	$2.40	microcontroller mount
74HC00 QUAD 2 INPUT NAND GATE	Lee's Electronics	71683	$1.00	1	$1.00	emergency stop
JST connectors, XH, 6 pin	Lee's Electronics	285961	$2.00	2	$4.00	motor connections
power supply 12V 3A	Lee's Electronics	109091	$16.00	1	$16.00	power supply
6" C clamp	Home Depot	1000816968	$4.19	1	$4.19	C clamp for base
nuts/bolts/misc. harware	Home Depot	N.A.	$20.00	1	$20.00	general construction
3D printer filament	3D Printing Canada	StandardPLA175Grey1kg	$6.00	1	$6.00	print robot body
				Total:	**$429.65**	

11 System Verification

To determine if the performance of the robot matches the specifications that are attributed to it, our team has tested the combined function of the robot as well as individual subsystems independently from one another. To verify that the combined function of the robot works, we tested a specific set of moves that are designed to indicate failure points. For individual subsystems, we verify that the expected responses are observed through specific inputs that mimic the inputs that are possible from the robot.

11.1 System Operation

To test the range of inputs of the assembled robot, a variety of commands were sent to the robot that were chosen based on the maximum variety of input arguments for a given command. To test the accuracy of the physical part of the robot with a tool attached, we performed tests with different move commands to see the resolution of the end point. We also tested the accuracy of the motors using the firmware to track their position while performing a move.

11.1.1 Physical Operation

Arguments for the positions were based on results where arm solution changes were not necessary and were necessary because this is where a failure point would most likely occur as they are the most complex moves. They were also based on movements comprised of valid start and end points, but the path that the robot TCP would travel along would be invalid. This includes movement that crosses over the zone that takes up the robot base, or the area outside of the work envelope.

	demo				arm solution	expected movement	result
	moveJ	45	-45.0			move	correct
	moveJ	-45	45.0			move	correct
	moveJ	110	180.0			move	correct
	moveJ	-110	-180.0			move	correct
	moveJ	45	-135.0			no move (arm collapse)	correct
	moveJ	-45	135.0			no move (arm collapse)	correct
	moveL	30	0.0		L	move	correct
	moveL	-17	-20.0		L	move	correct
	moveL	15	-15.0		R	move with previous arm solution	correct
	moveL	25	0.0		R	move	correct
	moveL	-17	20.0		L	beginning switch, arm solution change	correct
	moveL	15	15.0		L	move with previous arm solution	correct
	moveL	-7	-22.0		R	no move (inner boundary)	correct
	moveL	-11	13.0		R	no move (outter boundary)	correct
	moveL	21	23.0		R	no move (outter boundary)	correct
	moveL	18	-24.0		R	no move (outter boundary)	correct
	moveC	0	360.0	11	L	move	correct
	moveC	0	135.0	11	R	beginning swtich	correct
	moveC	-56	180.0	9	L	beginning switch, arm solution change	correct
	moveC	100	-100.0	31	R	arm solution change	correct
	moveJ	0	0.0				
	moveC	0	180.0	12	L	no move (inner boundary)	correct
	moveC	-180	0.0	5	L	no move (outter boundary)	correct

Figure 23: System operation movement

The results shown above in Figure 23 demonstrate that the robot behaves as expected when certain input conditions are requested.

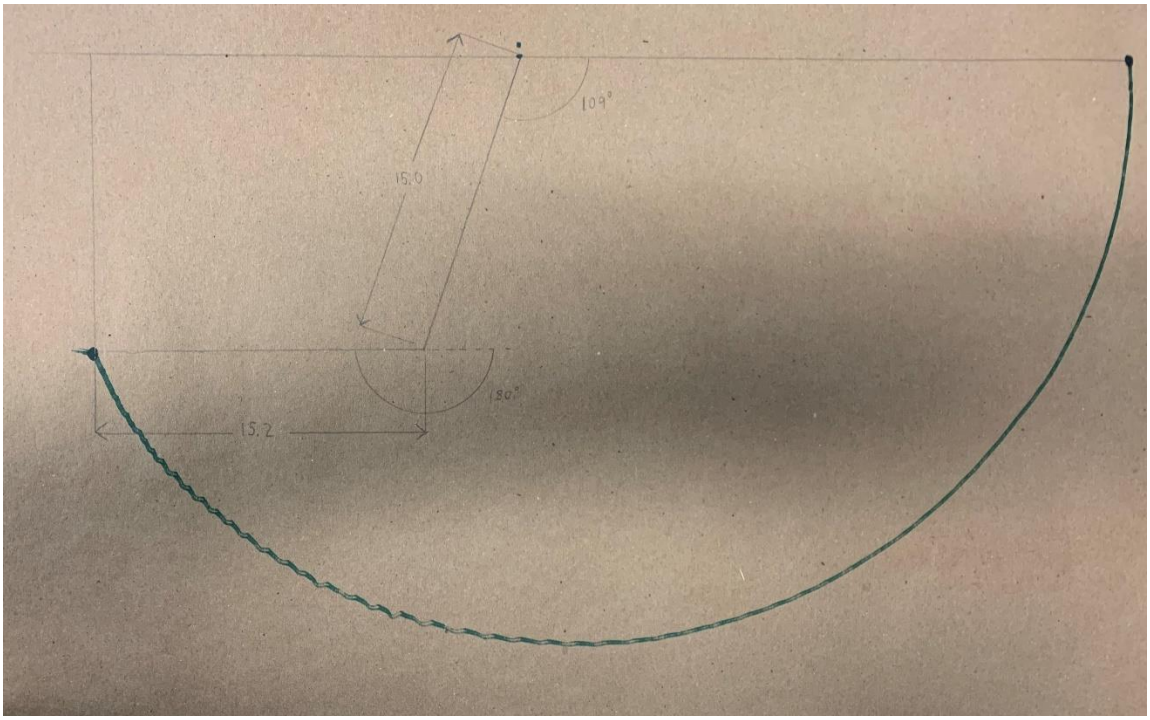

Figure 24: Joint Interpolated Move with Tool

A joint interpolated move is shown above in Figure 24 that started with each joint at zero degrees. The robot was commanded to move joint L1 to -110 degrees and joint L2 to -180 degrees. The end coordinate is from the trace is (20.1, -14.2), and the expected end coordinate is calculated to be (-20.4, -14.3). We found that the discrepancy between the predicted end point and the measured end point is due to two reasons. Imperfect tuning of the PID control variables can influence the end point of a move. From our testing of the controller firmware, we observe that end angles can deviate as much as half a degree from their target. A slight upwards tilt of the robot arm also explains the shorter arm lengths observed on the paper. This is due to the tension of the belt acting mostly on L1 which causes an upwards tilt of approximately one degree or an elevation of the end point of 0.25 cm.

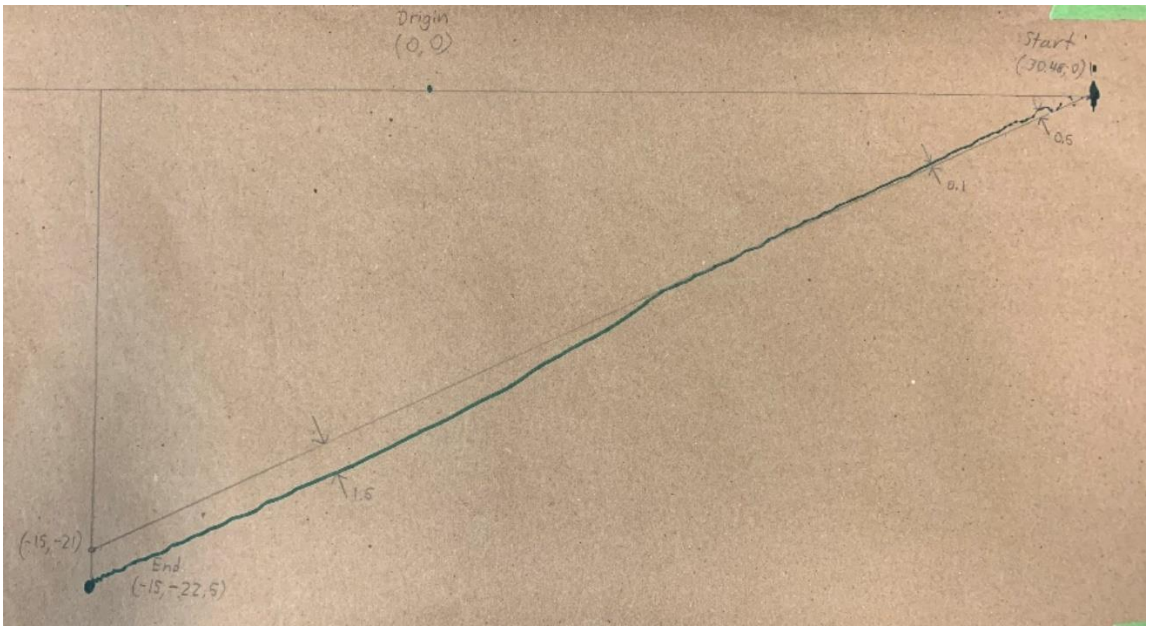

Figure 25: Linear Move with Tool

The green line that was drawn shown in Figure 25 started from the position of each joint at zero degrees and the robot was commanded to move to (-17, -20) using a left arm solution. We used this move because it is the longest linear move that is possible, and it will show the resolution for a worst-case scenario. From the start of the line on the right side of the paper, there is an immediately an unstable section with a maximum deviation of 0.5 cm from the desired path. This instability in the move occurred because the joints were fully extended, so any small change in movement from the motors results in a large change at the end point of the arm. Another reason for the instability is due to the large starting torque required by the motors, which causes a sudden change in acceleration resulting in jerky behaviour of the tool. After the initial jerkiness of the movement, the tool continues along a straight line, deviating 0.1 cm from the desired path. At approximately the halfway point of the line, the tool is shown to start deviating from the course and is observed to have a maximum deviation of 1.5 cm. We attribute this deviation to the instability of the z-Axis lead mechanism that the tool is mounted to. Up until the halfway point in the line, the orientation of joint two is such that the tool is being pulled along the path. At the point where the deviation occurs, joint two's orientation changes and the tool starts to be pushed in a different direction and causes the deviation.

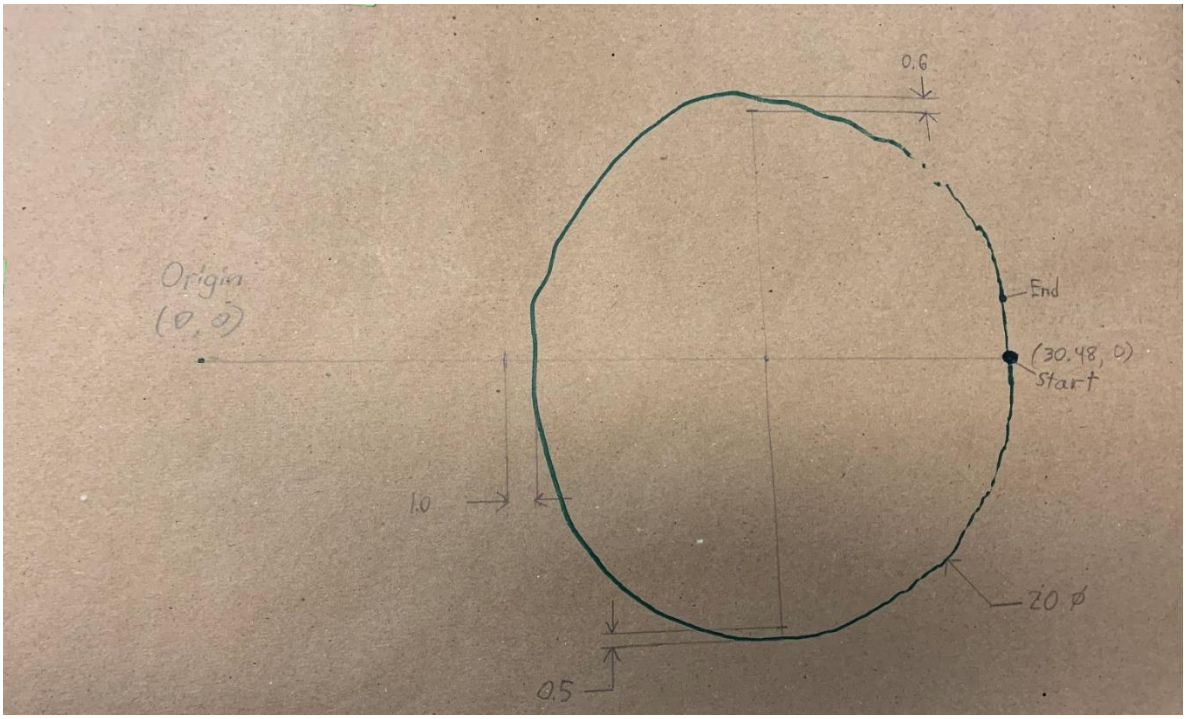

Figure 26: Circular Move with Tool

In Figure 26 shown above, a circular move is demonstrated by the robot moving from a starting angle of zero degrees to an end angle of -360 degrees to show the worst-case resolution through a full move of an arc. The radius of this circle was commanded to be 10 cm, which is the largest full circle that can be performed by the robot and the sign attributed to the -360 degree end angle is the clockwise direction that the circle is drawn in. Just above the starting point of the robot, there is a connection between the start point and the end point of the robot. This jerk in movement occurred at the start of the move because of imperfect tuning of PID control variables and jerkiness caused by the large starting torque required to get the motors moving. The circle is observed to have a larger radius than expected at the -90 and -270 degree points and has a smaller radius at the -180 degree point closer to the base of the robot. This elliptical shape is caused from the inclination of the arm due to too much tension in the belt. The circle is also observed to have ended slightly before completing full 360 degrees of movement. This error can be explained by the small amount of shaft play at the end of the motors combined with the firmware error causing L2 to have a final angle slightly less than the expected end angle.

11.1.2 FIRMWARE OPERATION

The Firmware's ability to process user given data into target positions and then to send command signals of the appropriate size and at the right time, is what determines the performance of the firmware. Testing was split up into multiple parts, so that verifying each component of the firmware was possible.

Testing the ability to process user given data into target positions was the first task that we had to verify computation was successful. This was tested by entering values into the python interface for the possible movements and observing whether the MSP430 was able to store the data correctly and call the corresponding functions. Using breakpoints, we observed that the program was successfully able to interpret and store data through python.

If the entered command was known to be impossible to perform because the range was out of the work envelope, the inner boundary around the base was violated, joint L2 would violate joint L1, or if arm angles limits were violated, the program was expected to quit the command and the move would not be performed. We observed that the program operated as expected by quitting the command if invalid data was entered or if invalid data was calculated for a move.

While testing the robot in the real world, we verified that the control loop variables were accurate by storing the position errors for each motor in an array. For a given move, we observed the desired position, the actual position and the error values for a given move. Being able to observe the discrepancy of how far off the robot was from its desired position throughout the move, we were able to accurately tune the corresponding gain values to achieve smooth, accurate control of the robot.

0x002400	posArray1,	posArray1,	posArray1,	posArray1,	posArray1								
0x002400	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00241C	0	0	0	0	0	0	0	0	0	0	0	0	-1
0x002438	-1	-1	-1	0	0	0	-1	-1	-1	-1	-1	-2	-1	-1
0x002454	-1	-2	-2	-2	-3	-3	-2	-2	0	1	0	0	-2	-2
0x002470	-2	-2	-3	-3	-3	-2	-2	0	0	-1	0	0	-1	-1
0x00248C	-2	-3	-3	-2	-1	0	0	-1	-2	-2	-3	-3	-2	-1
0x0024A8	0	1	1	-1	-3	-3	-3	-2	-1	1	0	-1	-2	-3
0x0024C4	-2	-1	-1	0	0	0	0	-1	-3	-4	-4	-2	0	1
0x0024E0	1	0	-2	-3	-4	-5	-4	-1	1	3	4	1	-1	-3
0x0024FC	-4	-5	-4	-1	1	2	1	-1	-4	-5	-4	-2	0	2
0x002518	3	2	-1	-2	-4	-3	-3	0	0	1	0	-2	-3	-3
0x002534	-2	-1	0	1	0	-1	-2	-2	-2	-2	-1	0	0	0
0x002550	-1	-1	-2	-2	-2	-2	-1	0	0	-1	-1	-1	-2	-3
0x00256C	-2	-1	0	1	1	0	-3	-4	-4	-2	0	1	1	0
0x002588	0	-2	-2	-2	-2	-1	0	0	0	-1	-1	-2	-3	-3
0x0025A4	-2	0	2	2	0	-2	-3	-3	-2	0	0	0	0	-1
0x0025C0	-2	-2	-2	-1	0	0	0	0	-1	-1	-1	-1	0	0
0x0025DC	-1	-1	-2	-2	-2	-1	0	0	0	0	0	-2	-2	-2
0x0025F8	0	1	1	0	-1	-3	-3	-2	0	1	2	1	0	-2
0x002614	-3	-3	-2	0	2	2	1	-1	-2	-2	-2	-1	0	1
0x002630	2	1	0	-2	-2	-2	-1	0	1	1	0	-1	-1	-1
0x00264C	-1	0	1	1	0	0	-1	0	0	0	-1	0	0	0
0x002668	1	0	-1	0	0	0	0	1	0	-1	0	1	1	1
0x002684	0	0	-1	-1	0	0	1	1	2	0	0	-1	0	0
0x0026A0	0	0	1	1	1	1	1	1	0	-1	-1	1	1	1
0x0026BC	0	-1	0	0	1	1	1	1	0	0	0	1	1	2
0x0026D8	1	0	0	0	1	0	1	1	1	0	0	0	1	1
0x0026F4	2	0	0	0	0	1	2	3	1	0	-1	-1	0	2
0x002710	3	2	0	-1	-2	-1	1	2	3	2	2	0	0	-1
0x00272C	-1	1	2	3	2	0	0	0	1	0	1	2	2	2
0x002748	2	1	1	1	0	-1	-1	1	2	2	3	3	2	1
0x002764	0	-1	0	0	1	3	2	0	0	-1	-1	0	1	3
0x002780	2	3	2	2	1	2	3	2	2	1	1	1	0	0
0x00279C	1	0	0	0	0	-1	-1	0	2	3	2	2	2	2
0x0027B8	2	2	2	1	1	1	1	1	1	1	1	1	1	1
0x0027D4	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0x002BD2	posArray2,	posArray2,	posArray2,	posArray2,	posArray2								
0x002BD2	0	0	0	0	0	0	0	0	0	0	0	0	0
0x002BEE	0	0	0	0	0	0	0	0	0	-1	-1	-1	-1	-1
0x002C0A	-1	-2	-2	-1	-1	-2	-2	-2	-2	-2	-2	-3	-3	-2
0x002C26	-3	-1	0	0	-1	-1	-2	-2	-2	-2	0	0	-1	-2
0x002C42	-2	-3	-4	-3	-1	0	1	1	-1	-2	-3	-4	-3	-2
0x002C5E	0	2	0	-1	-2	-4	-4	-2	0	2	1	1	-1	-3
0x002C7A	-3	-3	-3	-2	0	1	0	-1	-3	-5	-5	-3	0	2
0x002C96	2	0	-2	-3	-5	-5	-2	0	2	2	0	-1	-3	-4
0x002CB2	-3	-3	-2	-1	-1	-1	-1	-1	-1	-1	-1	-2	-3	-3
0x002CCE	-2	-1	0	0	-1	-3	-3	-4	-4	-2	0	1	0	-1
0x002CEA	-1	-2	-3	-4	-4	-2	-1	0	0	-1	-2	-3	-4	-4
0x002D06	-3	-1	-1	0	0	-2	-3	-3	-3	-2	-2	-1	-1	-1
0x002D22	-1	-2	-2	-3	-3	-2	-2	-2	-1	-1	-1	-2	-3	-2
0x002D3E	-3	-2	-2	-2	-1	-1	-1	-1	-2	-2	-3	-3	-2	-1
0x002D5A	-1	-1	-1	-2	-3	-3	-2	-2	-1	-1	-1	-1	-1	-2
0x002D76	-2	-2	-3	-3	-2	-1	0	-1	-1	-1	-2	-2	-3	-3
0x002D92	-2	-1	-1	-1	0	-1	-1	-2	-3	-2	-1	-1	0	-1
0x002DAE	-2	-2	-2	-2	-2	-1	-1	-1	-1	-1	-2	-2	0	0
0x002DCA	-1	-2	-1	-1	-1	-2	-1	-1	-1	-1	0	-1	-1	-1
0x002DE6	-1	-1	-1	0	-1	-1	-1	-1	-2	-1	-1	0	0	0
0x002E02	0	-1	-1	-2	-1	-1	0	0	1	0	-1	-2	-2	0
0x002E1E	0	0	1	1	0	-1	-1	-1	0	0	0	1	0	0
0x002E3A	0	0	0	0	0	1	1	0	0	-1	0	0	0	1
0x002E56	1	1	0	-1	0	1	0	0	0	0	0	1	1	0
0x002E72	0	0	0	1	1	1	0	0	1	2	1	1	1	0
0x002E8E	0	0	0	1	2	2	2	1	0	0	1	1	1	1
0x002EAA	2	2	2	2	1	0	-1	-1	1	2	2	3	2	1
0x002EC6	0	1	1	1	2	2	2	1	0	1	2	2	2	1
0x002EE2	1	1	1	1	0	1	2	3	2	2	1	1	0	1
0x002EFE	2	3	2	1	0	1	2	2	2	2	1	1	1	2
0x002F1A	2	1	0	1	2	2	2	3	2	2	1	1	1	1
0x002F36	2	3	3	1	0	0	0	1	2	3	3	3	2	1
0x002F52	1	1	0	-1	0	3	4	3	4	3	3	2	2	1
0x002F6E	1	0	1	0	0	0	0	1	2	3	3	1	1	2
0x002F8A	2	1	1	1	1	1	1	0	0	0	0	1	2	3
0x002FA6	4	4	4	4	3	3	3	3	2	2	2	2	2	2
0x002FC2	2	2	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27:Position error of moveJ command

Figure 27 shows the position error for each joint motor over the joint interpolated move from 0 to -110 degrees for M1 and 0 to -180 degrees for L2. The physical result is also shown in figure 24. This figure shows how the PID control keeps the position error of each joint on target around zero error. At the bottom row of column eight for joint L1, the final position is off target by one pulse or 0.1 degrees. At the bottom row of column eight for joint L2, the final position error is zero which means that it is exactly where it is supposed to be. The culmination of the errors

in both joints results in a resolution of 0.5 mm if both joints are lined up, according to the motors and software which is within the specification of the robot.

Even though the endpoint of the robot is accurate if ideal hardware was used, the error in the motor positions can fluctuate by as much as 5 pulses during the move as shown in figure 27. This half degree of movement in each motor can lead to a resolution of 5.3 mm at the end of the arm if the arms are lined up together.

0x002400	posArray1,	posArray1,	posArray1,	posArray1,	posArray1								
0x002400	0	0											
0x002404	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00241E	0	0	0	0	0	0	0	0	0	1	1	1	0
0x002438	0	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1
0x002452	-1	-1	-2	-2	-2	-3	-3	-2	-3	-2	-1	0	0
0x00246C	-1	-1	-2	-2	-3	-3	-3	-2	-2	0	0	0	1
0x002486	0	-1	-1	-2	-2	-3	-2	-1	-1	0	-1	-2	-2
0x0024A0	-3	-3	-2	-1	0	1	0	-1	-3	-4	-4	-2	0
0x0024BA	1	1	0	-1	-3	-3	-1	-1	0	-1	-1	-1	-1
0x0024D4	-3	-3	-2	0	0	0	0	0	-1	-2	-3	-4	-3
0x0024EE	-1	0	1	2	1	-1	-3	-4	-5	-3	-1	1	2
0x002508	2	0	-3	-4	-3	-2	0	1	1	0	-3	-4	-6
0x002522	-5	-2	1	2	3	1	-2	-4	-5	-4	-2	0	2
0x00253C	1	-1	-3	-5	-4	-3	-1	1	2	1	0	-2	-2
0x002556	-2	-2	-2	-1	0	0	-2	-2	-2	-1	-1	-1	-1
0x002570	-1	0	0	0	-1	-1	-2	-2	-2	-1	-1	-1	0
0x00258A	0	-1	-1	-2	-2	-1	0	0	-1	0	-1	-1	-2
0x0025A4	-3	-2	0	0	0	0	-1	-2	-2	-2	-1	0	0
0x0025BE	0	0	-1	-1	-2	-2	-1	-1	0	0	0	0	-1
0x0025D8	-1	-1	0	0	-2	-2	-2	-1	0	0	0	0	0
0x0025F2	-2	-2	-2	-1	0	1	1	0	-2	-3	-3	-1	1
0x00260C	2	1	0	-1	-2	-2	-2	-1	1	2	1	0	-2
0x002626	-2	-2	-1	0	1	1	0	-1	-1	-1	0	0	0
0x002640	0	0	0	0	0	0	-1	0	1	0	0	-1	-1
0x00265A	0	1	0	0	1	0	-1	0	0	0	1	0	1
0x002674	0	0	0	-1	0	1	0	1	0	0	-1	0	0
0x00268E	0	1	1	1	0	-1	-1	0	1	1	1	0	0
0x0026A8	0	0	0	1	1	1	0	1	0	0	-1	-1	1
0x0026C2	1	1	0	1	0	-1	0	0	2	1	1	1	0
0x0026DC	0	-1	0	0	1	2	3	1	1	0	0	0	0
0x0026F6	0	0	0	1	2	1	2	2	1	0	0	0	1
0x002710	1	1	1	1	0	0	1	2	2	1	1	0	1
0x00272A	0	0	1	0	1	2	1	1	1	1	0	0	1
0x002744	2	2	2	2	1	0	0	0	-1	0	1	2	3
0x00275E	3	2	1	0	0	0	0	1	3	2	1	1	0
0x002778	-1	-1	-1	1	2	3	2	2	1	1	1	1	2
0x002792	2	2	2	1	1	1	0	0	0	0	-1	-1	0
0x0027AC	0	1	1	2	2	2	2	2	2	1	1	1	1
0x0027C6	1	1	1	1	1	1	1	1	1	1	1	1	1
0x0027E0	1	1	0	0	1	1	1	1	1	1	0	0	0

0x002BD2	posArray2,	posArray2,	posArray2,	posArray2,	posArray2								
0x002BD2	0	0	0	0	0	0	0	0	0	0	0	0	0
0x002BEC	0	0	0	0	0	0	0	0	0	0	-1	-1	-1
0x002C06	-1	-1	-1	-2	-2	-3	-3	-3	-3	-1	-1	0	0
0x002C20	-1	-1	-1	-2	-2	-3	-3	-3	-2	-2	-1	0	-1
0x002C3A	-1	-2	-3	-3	-2	0	0	0	0	-1	-2	-3	-3
0x002C54	-3	-1	0	1	1	0	-1	-3	-4	-4	-3	-1	1
0x002C6E	1	1	-1	-2	-3	-4	-3	-1	0	0	1	0	-2
0x002C88	-4	-5	-4	-2	0	1	1	0	-2	-3	-4	-4	-2
0x002CA2	0	1	1	-1	-3	-3	-3	-3	-1	0	0	0	-1
0x002CBC	-2	-3	-3	-3	-2	-1	0	-1	-2	-2	-3	-2	-2
0x002CD6	-2	-2	-1	-1	-1	-1	-1	-2	-3	-3	-1	-1	-2
0x002CF0	-3	-3	-3	-2	-1	0	-1	-2	-2	-3	-2	-2	-1
0x002D0A	-1	-1	-2	-3	-4	-3	-2	-1	-1	-1	-1	-2	-2
0x002D24	-3	-2	-1	-1	-1	-2	-2	-2	-4	-3	-2	-2	-1
0x002D3E	-1	-1	-1	-2	-2	-3	-2	-1	-1	-1	-2	-3	-2
0x002D58	-2	-2	-3	-3	-2	-1	0	0	-2	-3	-3	-2	-2
0x002D72	-1	-1	-1	-1	-3	-3	-2	-3	-2	-2	-1	0	0
0x002D8C	-1	-2	-3	-3	-2	-2	-2	-1	-1	0	0	-2	-2
0x002DA6	-3	-3	-1	-1	0	0	-1	-2	-3	-2	-2	-2	0
0x002DC0	0	0	-1	0	-1	-2	-2	-1	-1	-1	-2	-1	-1
0x002DDA	-1	-1	-1	-1	-1	-2	-1	-1	-1	0	0	-1	-1
0x002DF4	-2	-3	-1	-1	0	1	0	0	-1	-1	-2	-2	-1
0x002E0E	0	0	1	0	0	-1	-1	0	-1	0	0	1	0
0x002E28	-1	-1	0	0	0	0	0	0	0	0	0	0	1
0x002E42	0	1	0	0	0	-1	0	1	1	1	0	0	-1
0x002E5C	-1	0	1	1	1	0	0	0	1	1	0	0	1
0x002E76	1	0	0	0	0	0	1	2	1	0	1	0	0
0x002E90	1	0	1	2	1	1	0	0	0	1	2	1	1
0x002EAA	1	1	1	2	2	2	1	-1	0	1	1	2	2
0x002EC4	3	1	1	1	0	0	0	1	1	2	2	3	2
0x002EDE	2	1	2	1	1	0	0	2	2	3	2	2	1
0x002EF8	1	0	0	1	2	2	2	2	2	2	2	2	1
0x002F12	1	1	1	3	2	2	1	1	1	2	1	2	1
0x002F2C	2	1	2	2	1	2	3	2	1	1	2	2	2
0x002F46	1	2	2	1	1	0	0	0	1	2	2	3	3
0x002F60	2	2	1	1	1	1	0	0	-1	0	1	3	3
0x002F7A	3	3	3	3	3	2	2	3	3	2	2	2	2
0x002F94	2	2	1	1	1	1	1	1	1	1	2	2	2
0x002FAE	2	2	2	2	1	1	1	1	1	1	1	1	0
0x002FC8	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28: Position error of moveL command

Figure 28 shows the position error for each joint motor over the linear move from (30, 0) to (-17, -20). Similar to the moveJ position error values, this linear move is also accurate with reaching the final joint position within one pulse for L1 and exactly hitting the target for L2 and is within the specification of the robot according to the motor positions and firmware.

0x002400	posArray1,	posArray1,	posArray1,	posArray1,	posArray1												
0x002400	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x002424	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0x002448	2	2	1	1	1	1	1	2	2	1	1	0	1	0	-1	-2	-1	-1
0x00246C	-1	-1	0	0	0	1	1	1	2	2	2	2	2	1	0	0	0	0
0x002490	1	1	2	2	3	1	0	-1	-1	0	0	0	0	-1	0	1	1	2
0x0024B4	2	3	2	1	1	0	-1	-3	-2	0	1	2	2	2	0	-1	-3	-2
0x0024D8	-1	1	2	2	2	1	0	0	0	0	0	1	1	0	-1	-1	-1	0
0x0024FC	2	2	3	2	0	-1	0	-1	0	0	0	-1	-1	0	-1	0	2	3
0x002520	1	0	0	1	1	1	0	-1	-1	-1	0	1	0	0	0	2	3	3
0x002544	1	0	-2	-1	-2	-1	1	1	1	1	0	-1	-1	0	2	2	2	0
0x002568	-1	-1	-1	1	2	1	0	0	0	0	0	0	1	1	2	1	0	0
0x00258C	0	-1	-1	-1	0	1	1	2	2	1	0	-1	0	1	1	2	1	0
0x0025B0	-1	-1	-1	0	2	1	1	1	0	0	0	0	1	-1	-1	-1	0	1
0x0025D4	2	2	1	1	0	-1	-1	0	0	1	1	1	1	0	0	0	0	2
0x0025F8	2	1	0	-1	-1	0	0	0	-1	-1	1	1	1	2	1	0	0	0
0x00261C	-1	0	-1	-1	1	1	1	2	1	1	-1	-1	-1	-1	0	0	1	1
0x002640	2	2	2	0	0	-1	-2	-1	-1	1	1	2	1	1	0	-1	0	-1
0x002664	-1	-1	-1	-1	1	2	2	2	1	-1	-1	-2	-1	-1	0	0	0	0
0x002688	0	0	0	-1	-1	-1	0	0	-1	0	1	1	1	0	-1	-1	-2	-2
0x0026AC	-1	-1	0	0	0	1	0	-1	-1	-2	-1	-2	-2	-1	-1	0	0	-1
0x0026D0	-1	-1	-1	-1	0	0	-1	-2	-1	-1	-2	-2	-2	-2	-3	-3	-4	-5
0x0026F4	-5	-5	-6	-6	-6	-5	-4	-2	-3	-4	-5	-6	-5	-5	-5	-4	-3	-3
0x002718	-4	-4	-3	-4	-4	-4	-4	-4	-5	-4	-4	-3	-3	-2	-3	-3	-4	-4
0x00273C	-4	-4	-3	-2	-1	-1	-2	-3	-3	-3	-3	-3	-2	-1	-1	-1	-1	-2
0x002760	-1	-1	-1	1	0	0	0	0	0	0	0	0	1	0	0	-1	0	1
0x002784	1	1	0	0	1	0	1	1	1	1	2	1	1	1	1	0	0	0
0x0027A8	1	2	2	1	1	0	0	1	1	3	2	2	1	0	-1	0	1	1
0x0027CC	1	1	1	1	1	1	1	1	1	2	1	1	-1	0	0	1	0	0
0x0027F0	1	1	1	2	2	2	0	-2	-1	0	1	2	3	1	0	-1	-1	0
0x002814	1	3	2	0	-1	-2	-1	1	2	0	1	1	1	0	1	0	0	0
0x002838	1	2	1	0	-2	-2	0	1	0	1	1	0	0	0	1	2	1	1
0x00285C	0	-1	-1	-1	-1	-1	-1	0	1	2	3	2	2	1	0	-1	-1	-2
0x002880	-1	1	1	1	2	1	0	-1	-1	-1	-2	-1	0	0	1	1	1	1
0x0028A4	1	0	-1	-1	-2	0	0	0	-1	0	1	1	2	1	0	-1	-1	-1
0x0028C8	-1	1	1	0	0	-1	-1	-1	-1	2	2	2	2	1	1	0	0	0
0x0028EC	-1	-1	-2	-1	0	2	2	2	0	0	-1	0	-1	-1	-2	-1	0	1
0x002910	0	0	-1	-1	-1	0	2	2	2	1	1	0	0	0	-1	-1	-2	-1
0x002934	1	1	1	1	0	0	-1	0	1	0	1	1	1	1	1	0	1	1
0x002958	1	0	0	0	-1	-1	0	0	-1	-1	-1	-2	-1	-1	1	1	2	1
0x00297C	1	0	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	0	2	2	1	1
0x0029A0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0
0x0029C4	0	-1	-1	-1	-1	-1	-1	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1
0x0029E8	0	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1
0x002A0C	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0x002A30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0x002BD2	posArray2,	posArray2,	posArray2,	posArray2,	posArray2												
0x002BD2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x002BF6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x002C1A	0	0	1	1	1	1	1	1	1	1	1	1	0	1	0	0	-1	-1
0x002C3E	0	0	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	-1	-1	0
0x002C62	-1	-1	-1	-1	-1	-1	-1	-2	-1	-2	-1	-2	-2	-1	-1	-2	-1	-2
0x002C86	-2	-1	0	1	2	1	1	2	1	1	1	1	0	0	0	0	0	-1
0x002CAA	-1	-1	-1	-1	-1	-1	-1	-2	-2	-1	0	0	1	1	1	0	0	0
0x002CCE	-1	-1	-1	-1	-2	-1	-2	0	0	1	2	2	1	1	1	0	0	-1
0x002CF2	0	-1	-1	-1	-1	-2	-1	-1	-1	0	0	2	2	1	0	-1	-2	-1
0x002D16	-1	0	1	0	1	1	0	0	0	-1	-1	-1	-1	-1	-1	0	1	1
0x002D3A	0	-1	0	0	-1	-1	-2	-3	-1	2	3	3	2	1	1	0	-1	0
0x002D5E	-1	0	-1	-1	-2	-2	-2	-1	1	2	4	3	2	1	0	-1	-1	-2
0x002D82	-1	-1	-1	0	0	1	0	-1	-1	0	1	1	0	0	-1	-2	-3	-1
0x002DA6	0	1	2	1	0	0	-1	-1	-2	-3	-3	-2	0	3	4	4	3	1
0x002DCA	0	0	-1	-1	-2	-3	-3	0	2	2	1	0	-1	-1	-2	-2	-1	0
0x002DEE	1	1	1	0	-1	-1	-1	-1	0	-1	0	0	1	1	0	-1	-3	-3
0x002E12	-2	0	1	2	2	1	0	-2	-4	-3	-2	0	1	1	2	2	2	0
0x002E36	-1	-3	-4	-4	-2	0	3	4	3	1	-1	-3	-5	-4	-2	0	2	3
0x002E5A	3	2	1	-2	-3	-4	-3	-2	-1	1	2	2	3	1	-1	-4	-4	-4
0x002E7E	-2	0	3	2	2	1	0	-2	-2	-4	-4	-2	0	1	2	1	1	-1
0x002EA2	-2	-3	-3	-3	-1	0	1	0	0	-2	-2	-2	-1	-2	-1	-1	-1	0
0x002EC6	0	-1	-2	-3	-2	-2	-1	-1	-1	-1	-1	-2	-2	-2	-3	-2	-1	0
0x002EEA	0	-1	-2	-3	-2	-2	-2	-1	0	0	0	-1	-2	-2	-1	-1	0	0
0x002F0E	1	0	1	0	1	0	1	1	1	1	0	1	2	2	2	2	3	3
0x002F32	3	2	1	1	1	2	2	3	4	5	3	3	2	1	1	2	3	4
0x002F56	4	4	2	1	1	1	1	1	4	5	6	6	5	5	4	4	5	4
0x002F7A	4	5	5	5	4	3	2	2	3	3	5	5	4	3	2	2	2	2
0x002F9E	2	3	3	2	2	1	2	2	3	3	3	1	1	1	1	1	2	3
0x002FC2	2	1	1	1	1	1	1	1	1	2	2	1	1	0	-1	1	1	1
0x002FE6	1	2	1	0	0	0	0	0	2	2	1	1	-1	0	0	-1	-1	1
0x00300A	1	1	2	1	1	0	-1	-1	0	0	0	1	1	0	1	0	0	0
0x00302E	0	0	-1	-1	0	2	2	1	0	-1	-1	-1	-1	-1	0	1	1	1
0x003052	1	1	0	-1	-2	-2	-1	0	1	2	1	0	0	0	-1	-1	-1	0
0x003076	-1	0	1	1	0	-1	-1	0	0	-1	-1	-1	-1	0	1	2	1	0
0x00309A	-1	-2	-2	0	-1	0	0	1	1	1	0	-1	-3	-1	-2	1	0	1
0x0030BE	2	1	0	-1	-1	-3	-2	0	1	1	1	0	-1	-1	0	-1	-1	-1
0x0030E2	0	0	1	0	1	0	-1	-1	-2	-1	-1	1	1	1	0	-1	-1	-1
0x003106	-2	1	1	0	1	-1	-1	-1	-1	0	-1	0	0	0	-1	0	-1	-1
0x00312A	0	1	1	1	0	-2	-2	-2	-1	0	2	3	2	0	-2	-2	-2	0
0x00314E	0	0	0	-1	-1	-2	0	1	1	1	-1	-1	-1	-1	-1	0	2	1
0x003172	1	0	0	-1	-1	-2	-1	-1	0	1	0	1	1	-1	-2	-3	-2	-1
0x003196	-1	1	1	1	0	1	0	0	-2	-1	0	0	1	1	2	0	-2	-2
0x0031BA	-2	-2	0	-1	-1	1	1	1	2	2	1	-1	-2	-3	-3	-3	-3	-2
0x0031DE	-2	-2	-2	-1	0	0	0	0	1	0	-1	-2	-1	-1	-1	-1	-1	-1
0x003202	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0	0	0	0
0x003226	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29: Position error of moveC command

Figure 29 shows the position error for each joint motor over the circular move from (30, 0) to move -360 degrees back to the starting point with a specified radius of 10 cm. The physical result of this move is shown on paper in figure 19. Similarly to the linear and joint interpolated move conditions, the robot reaches its final position within 1 pulse for L1 and with no position error in L2 which is within the specification of 5 mm resolution of the end point.

Looking at figure 29, the position errors for joint L1 around approximately the halfway point of the move deviate by a maximum of 6 pulses, but more significantly the PID control loop is not able to correct for the error immediately and time is taken to reach a lower position error. This discrepancy is at the same point in figure 26 when the robot deviates most from the calculated path around the 180-degree mark or halfway point of the circle. This leads us to believe that the deviation of physical movement is not only caused by the physical structure but is also due to the programming of the movement and work needs to be done to resolve this lack of accuracy.

11.2 Performance Metrics

The design features for the SCARA robot that were agreed upon in the proposal for this project were as follows:

- The robot should operate in the XY plane
- The length of the two arms should be the same with a minimum of 6 inches and a maximum of 8 inches.
- The robot should have a maximum resolution of 5mm at full extension
- The robot must be designed to be powered by a 12V power supply.
- Use of one microcontroller to send and receive signals for the entire robot.
- The microcontroller should be able to interface to a Windows 10 PC.
- A library function that provides the ability to read the encoder counts of the robot motors.
- A library function that provides the ability to send the robot a voltage.
- A library function that allows the robot to set its home position.

From these features described in the proposal, the robot that we have produced adheres to these features, and the instances of performance were measured in the following manner.

The robot was specified to operate in the XY plane, and we were able to achieve this with the addition of another degree of freedom in the Z-Axis. The joints of the robot were specified to be created between six and eight inches of length each. The robot joints were created with a length of 6 inches or 15.24 cm.

The resolution of the robot is required to have resolution of 5mm or less at the maximum arm's extension at the end of a move. Through the control of the motors being within 1 pulse or 0.1 degrees at the end of a move, the resolution of the TCP falls within this 5mm range according to the motor position and firmware.

When a move is performed with the physical pen tool on paper, the worst resolution that we observed was 1cm from the target at the end position. Observing this, the worst-case resolution of this robot is 1.1 cm which is over the desired target for this project.

The power supply specified in the proposal required that it provides 12 volts. The robot is powered using a 12V power supply that is capable of outputting 3 amps which provides enough current even if an additional load is added to the robot.

The robot makes use of one microcontroller, the MSP430 which adheres to the feature that only one microcontroller be used to control this robot. The microcontroller interfaces to the user through python on a windows 10 PC and follows the interfacing requirement outlined.

Functions are provided through the user interface that give the user the ability to perform joint interpolated movement, linear movement and circular movement, z-Axis movement and allow the user to send a move to be performed by the robot if an array of positions is created.

12 PROJECT SUMMARY

The purpose of this project was to design and build a robot that instructors could use to design labs around for 1st year students of the mechatronics and robotics program. We designed the robot using components taught about in the first and second year of the program to demonstrate how techniques taught are implemented into a robot and to give first year students a perspective of what components or theory's they will learn about in the second year of the program. The robot is replicable because of its low cost of $430 and 3D printed structure, which allows a whole class to have access to the same robot. This allows for an instructor to create more in-depth labs where first year students can get more hands-on time with a robot.

12.1 SYSTEM SPECIFICATIONS

The Modular SCARA robot is a 3 degree of freedom robot that operates in the XY axis, with the ability to travel a small amount in the Z-Axis. The robot is made up of two joints that can reach 30.48 cm or 12 inches when aligned. The first joint is 15.24 cm in length and has an operational range of movement of ±110°. The second joint is also 15.24 cm in length and has an operational range of movement of ±180°. The entire system is powered by a power supply brick that outputs 12 volts and can output 3 amps. The user interface allows for easy communication from the user to the robot through python. The robot uses the MSP-EXP430F5529LP microcontroller to receive data from the user and to control the movement of the robot. The MSP430 is operated at 20 MHz to ensure that position signals from the encoders are not missed and that calculations to update motor speeds are done before the next update. The update rate of the robot is 5mS. The two main motors for the robot arm are rated for 12 volts. The current draw of the robot when the arms are attached is observed at a maximum of 0.51 amps per motor when doing a coordinated move. The quadrature encoders for the two joint motors have a resolution of 3415.92 pulses per revolution or 9.4886 pulses per degree on the output shaft while using quadrature. The gear ratio for each joint motor 1:71 that uses a planetary gearbox to reduce instability of the output shaft.

12.2 RECOMMENDATIONS

Through completion of this project our team has noted improvements that should be made to the robot that were not implemented due to the time constraint on this project.

The first recommendation that we have is to develop a Z-Axis mechanism that is more stable. Currently the limiting factor to the Z-Axis movement and resolution of the robot is the instability of the Z-Axis lead screw mechanism. Finding a solution to this issue would be top priority if we had more time to complete this project. Fixing this issue would improve the physical resolution of the robot and would help showcase the accuracy that the motors are designed for.

The second recommendation that our team holds is that more metal should be used for the structure of this robot to increase the rigidity. We particularly notice the issue of minor structural warping through the base of the robot due to the counter torque of the motors when they are excited. Adding a metal brace along the spine of the robot would help to eliminate this issue and would increase the robot's resolution.

The third recommendation that we have is to use a motor driver with a higher current specification. When driving our motors at full strength, the motor driver that we are using cuts power at approximately 0.25 second intervals if a sudden increase in power is required and results in jerky movement of the robot arm. We had to limit the maximum PWM duty cycle to 35% which eliminates this issue.

If the structure of this robot is developed so that it is more stable and someone desires to increase the resolution and repeatability of the robot, we would recommend that a higher resolution motor is selected. Using the MSP430, the time to update both motor's positions allow for an increase in motor resolution. Currently we are limited by the small number of motor pulses that are updated at the current rate. Increasing the motor's resolution would allow for a faster update rate of approximately

If future work was done on this robot to increase its usability, our team would recommend implementing a keyboard arrow function that could be used to jog the robot. This function would greatly increase the robot's overall function

and ability to create programs for specific purposes. Our team would have added this feature to the robot if more time was allocated for this project.

13 CONCLUSION

The Robot that was proposed was successfully designed and built adhering to the design requirements specified in the proposal with exception to the robot's physical resolution. The robot is easy to move around, easy to replicate and cheap enough for multiples to be built. One major issue that is still present with the robot is that the motor driver cannot support full power if the motors require it.

APPENDIX 1 BILL OF MATERIALS

SCARA Bill Of Materials 16-May-22

Bill of Materials for one unit

Description	Supplier	Part#	Unit Price	Quantity	Price	Notes
118 RPM HD Planetary Gear Motor w/Encoder	Servo City	638324	$77.39	2	$154.77	One for each of L1 and L2
TB6612FNG Dual Motor Driver Carrier	Pololu	713	$12.84	1	$12.84	L1 and L2 motor driver
Stepper motor with lead screw plus carriage	Yanmis	Yanmiskysuwcgtz8	$23.80	1	$23.80	Z axis motor
A4988 Stepper Motor Driver Carrier	Pololu	1182	$10.97	1	$10.97	Z-axis motor driver
3mm HTD Pitch Plastic Hub Mount Timing Belt Pulley	ServoCity	3402-0014-0036	$5.15	1	$5.15	L2 belt drive
3406 Series 3mm HTD Pitch Timing Belt	ServoCity	3406-0015-0005	$38.69	1	$38.69	L2 belt drive
1309 Series Sonic Hub (1/4" Bore)	ServoCity	1309-0016-0250	$9.02	3	$27.05	Hubs for L1 and L2 shaft
1309 Series Sonic Hub (6mm D-Bore)	ServoCity	1309-0016-1006	$9.02	2	$18.03	Hubs for Motor shafts
0.250" (1/4") x 3.00" Stainless Steel Precision Shafting	ServoCity	634162	$1.79	1	$1.79	L2 belt drive shaft
Snap-Action Switch with 16.3mm Roller Lever	Pololu	1404	$1.74	4	$6.96	Limit switches for L1 and L2
msp-EXP430F5529LP	Digikey	296-36506-ND	$21.65	1	$21.65	microcontroller development board
shielded ball bearings 25x37x7	Bearingscanada	61805-2Z	$16.31	2	$32.62	bearings for L2 joint rotation
project box LX-642	Lee's Electronics	10363	$6.99	1	$6.99	for microcontroller and motor drivers
project box LX321	Lee's Electronics	10802	$3.50	1	$3.50	emergency stop
protoboard 170	Lee's Electronics	1058	$3.60	2	$7.20	microcontroller mount
plastic nylon spacers (pkg of 10)	Lee's Electronics	6991	$1.50	1	$1.50	microcontroller mount
push button momentary switch	Lee's Electronics	3277	$1.35	1	$1.35	emergency stop
led 5mm with holder cove	Lee's Electronics	5610	$1.20	1	$1.20	emergency stop
double row headers	Lee's Electronics	2147	$1.20	2	$2.40	microcontroller mount
74HC00 QUAD 2 INPUT NAND GATE	Lee's Electronics	71683	$1.00	1	$1.00	emergency stop
JST connectors, XH, 6 pin	Lee's Electronics	285961	$2.00	2	$4.00	motor connections
power supply 12V 3A	Lee's Electronics	109091	$16.00	1	$16.00	power supply
6" C clamp	Home Depot	1000816968	$4.19	1	$4.19	C clamp for base
nuts/bolts/misc. harware	Home Depot	N.A.	$20.00	1	$20.00	general construction
3D printer filament	3D Printing Canada	StandardPLA175Grey1kg	$6.00	1	$6.00	print robot body
				Total:	**$429.65**	

APPENDIX 2 MSP430 RESOURCES

The SCARA is controlled by the MSP430 f5529 microcontroller on an MSP-EXP430F5529 launchpad evaluation board. The following is a list of the timers, pins, and interrupts used by the SCARA. This information is provided especially for users who might want to extend the function of the SCARA, for example, by adding a tool.

MOTORS

Pulse Width Modulation (PWM) is used to control the drive strength to the motors. Pulse width timing is done with timer A 0, using its associated interrupt. Thus, these pins need to be able to output a timing signal.

- L1 is controlled by CCR3 on P1.4
- L2 is controlled by CCR4 on P1.5

Direction of movement is controlled by outputs to the selection pins on the H-bridge motor driver. When INA is high and INB is low, the connected motor turns counter-clockwise; when INA is low and INB is high, the motor turns clockwise. Be aware that the motor for L2 is mounted upside down relative to the motor for L1. In the code, the meaning of clockwise and counter-clockwise for L2 is inverted so that clockwise and counter-clockwise are always defined relative to a position above the SCARA arm. INA and INB for both motors are done using pins connected to port 3.

- INA L1 on P3.0
- INB L1 on P3.1
- INA L2 on P3.2
- INB L2 o P3.4

Quadrature Signal Decoding

Quadrature decoding is done using pins on port2. Each motor has two quadrature channels, channel A and channel B, which are monitored on the port 2 interrupt. As for the motors, the inversion of the encoder for L2 is accounted for in the code so that, for both motors, counts become more positive with counter-clockwise rotation defined from a perspective above the SCARA arm.

- Chan A L1 on P2.5
- Chan B L1 on P2.4
- Chan A L2 on P2.6
- Chan B L2 on P2.7

Z-axis Control

The Z-axis is controlled using a stepper motor with a motor driver that has STEP, DIRECTION, and ENABLE inputs. The steps are generated using Timer A 1 and uses the Timer A 1 interrupt to track position. The direction and enable are controlled with GPIO outputs.

- Z-axis STEP on 2.0
- Z-axis DIRECTION on P3.3
- Z-axis ENABLE on P3.5

Emergency Stop and Limit Switches

The emergency stop system consists of four limit switches, a physical emergency stop button, and an opto-isolator used as a GPIO-driven switch triggerable from software, all connected in series to a physical latching circuit. When any switch in the series is interrupted, even momentarily, output is latched low until a reset input is pulsed from high to low to high. The output of the latch goes directly to the enable outputs of the arm drive motors and the Z stepper motor, so they are immediately stopped. The stopped signal from the latch is reset by a GPIO output from the microcontroller. The emergency stop input is done on port 1, using the port 1 interrupt. The other signals are done on port 6

- Emergency Stop input on P1.6
- Emergency Stop reset on P6.5
- Software Stop on P6.6
- Emergency Stop LED on P6.1

Update Time Control

The timing loop uses no pins, but does make use of Timer B.

Tool Control

Given that it is difficult to predict what tools might be used with the compact SCARA, care was taken to leave as many resources free for tool usage as possible. Interrupt enabled pins are available on port1 and port 2. Timer A 2 and its pins are left untouched for tool use. The pins for the I2C port have not been used for GPIO and are available for I2C controlled hardware on a tool.

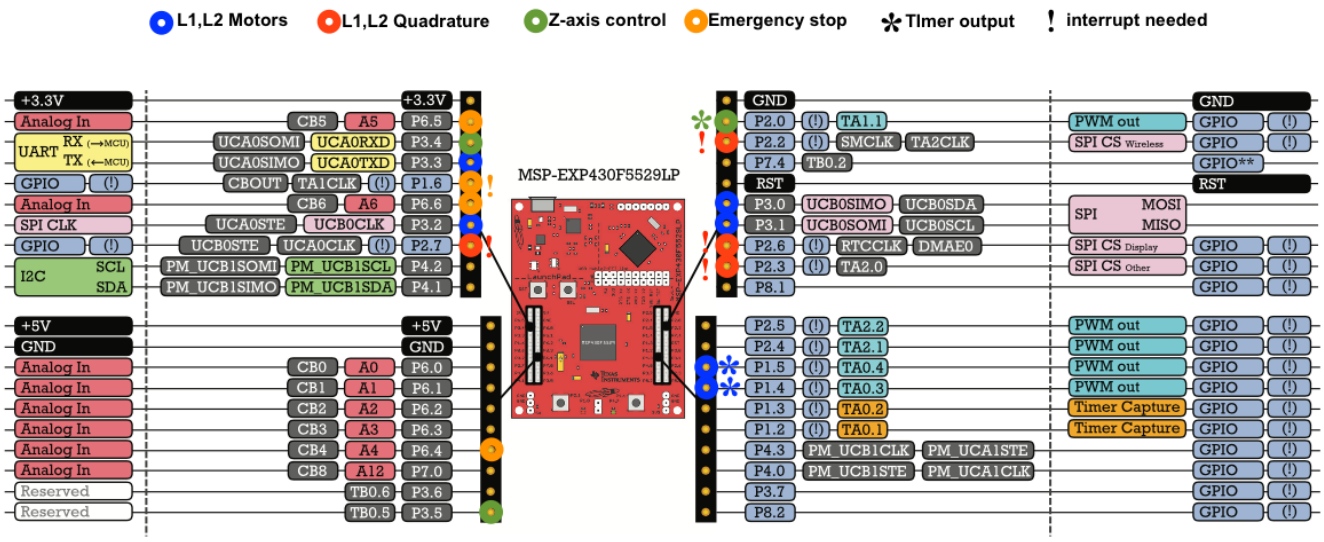

Figure 1: MSP430 pinout diagram

Figure 1: Pin usage on the MSP-EXP430F5529 launchpad evaluation board for the different functional modules of the compact SCARA. Pins used are circled with a colour corresponding to the functional module (See legend). Pins using PWM output are marked with an asterisk, and pins using an interrupt are marked with an exclamation point. Un-circled pins are available for use by a tool. Modified from Texas Instruments slau536.

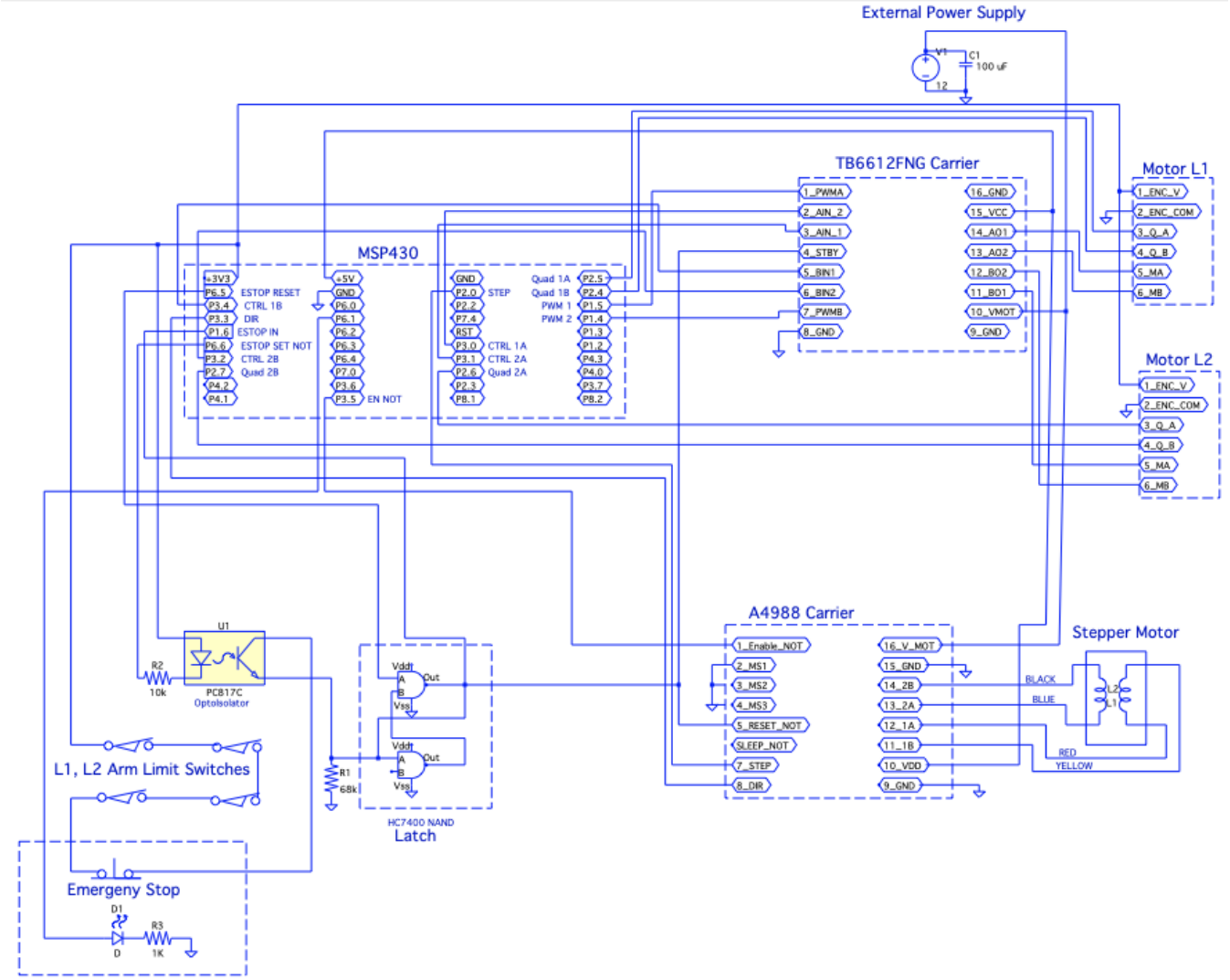

Figure 2: System schematic

APPENDIX 3 USER MANUAL

This document describes the operation of the compact Selective Compliance Robot Arm (SCARA). By analogy with a human arm, in the following discussion L1 refers to the upper arm segment, and L2 refers to the lower arm segment. The proximal end of L1 is fixed, and its distal end forms an elbow joint with L2. The distal end of L2 holds a tool of some sort, and a rail-mounted stepper motor provides movement in the Z-axis. Most of the function of the robot arm involves moving the tool to defined locations.

INSTALLING THE CODE

The control of the SCARA is divided into two parts, firmware that runs on the msp430 microcontroller and user interface code that runs on a host PC. All the code is available on the compact SCARA GitHub, https://github.com/mattWonn/compactSCARA.

The code that runs on the microcontroller is written in C in Code Composer Studio (CCS). The SCARA will already have this code pre-installed, and CCS is not needed to operate the SCARA. However, if the code is updated, the user may wish to install it on the SCARA, using the following procedure.

1. Download Code Composer studio from Texas Instruments

   https://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html

1. Clone the compact SCARA project from the GitHub link above. This can be done from within Code Composer Studio, using the git repository tab, or can be done manually. You will need a GitHub account. The free personal account is fine. The SCARA downloads as a CCS project, ready to be compiled.
2. With the compact SCARA connected to the host computer with a USB cable, compile the compact SCARA project and load it onto the msp430. This will replace the original firmware on the msp430.

Some users may wish to modify the C code themselves. The code is organized into the following modules which roughly correspond to the functional modules.

C Code Modules for Compact SCARA:

SCARA	Contains the main function, global variables, and constants
UcsControl	Uses the msp430 universal clock system to set frequency of main clock to 20 MHz
libUART1A	Interrupt-driven code to send and receive data over the serial port at 115200 baud
BinaryCmdInterp	Uses libUART1A to get commands from a host computer and return results
motorsPWM	Uses Timer A 0 to generate PWM signals to drive the motors, and sets direction with GPIO outputs to the motor driver's direction inputs
quadEncDec	Interrupt functions to decode quadrature inputs for the two motors into positions
movement	programs the arm movements in the X,Y plane
ZaxisCtrl	controls the stepper motor for the Z-axis
eStopLimitSwitch	controls limit switch and emergency stop button

PYTHON INSTALLATION

The user interfaces with the SCARA through a host computer running Python commands. Python is a free, open source, cross-platform programming language https://www.python.org. A typical Python environment consists of

the Python interpreter, a collection of software libraries (which are referred to as packages in the Python ecosystem), the Python package installer PIP, and a development environment. There are many ways that a Python environment can be installed:

1. A "batteries included" umbrella package such as Anaconda https://www.anaconda.com/products/distribution which includes most of the common Python libraries
2. From a package manager such as apt-get on Linux or homebrew on MacOs. On these systems, you will likely already have a full Python distribution installed.
3. As individual components from Python.org or other sources
4. As a plug-in for an integrated development environment, such as Visual Studio

Be sure to have a Python version of at least 3.5. Although Python 2.7 is no longer distributed, older computers may still have it installed. The SCARA code will NOT run with Python 2.7

The python serial library and the python tkinter library must also be installed for the SCARA. If not present, they can be installed with PIP. If you install Python with Anaconda, you will already have these libraries.

The Python code files, Modular SCARA.py and SCARA.py,, are downloaded along with the CCS project when compactScara is cloned from GitHub.

RUNNING COMPACT SCARA

Before running the SCARA code, make sure that the msp430 launchpad in the compact SCARA control box is connected to the host computer with a USB cable. The MSP430 launchpad has a USB/serial controller and presents as a standard serial port to the host computer. Commands are sent from Python to the SCARA over this serial interface, so it is important to know how the host system identifies the MSP430. Whether from the command line or from the GUI, the first step in running SCARA code is initializing a SCARA object with the correct serial port. The available serial ports can be enumerated in the Windows device manager, or, on Linux or MacOs, with a terminal command:

```
ls /dev/tty.*
```

Note that the MSP430 presents two serial ports, one of which is the application UART, which is the correct port, with the other being a debug interface, which will not work with the SCARA.

The SCARA can be operated in two ways, as a text-based interface from the command line of the Python interpreter running SCARA.py, or from a Graphical User Interface (GUI) consisting of a control panel provided by Compact SCARA.py. The control panel code imports the SCARA class from SCARA.py, creates a SCARA object, and provides an interface to it, so the GUI and the text-based interface will perform similarly.

The SCARA.py class includes the following functions:

__init__(self, port, baud):	Initialization function that sets up the serial port for communication with the SCARA, using the given serial port and baud. The baud must be 115200 to match the baud running on the microcontroller. Also initializes a dictionary of state information for the SCARA. If SCARA.py is run as main, an instance of SCARA named "robot" is created with the class values for default port and baud.
zeroEncoders (self):	Zeroes the counts on the quadrature encoders for the two motors
getPos (self):	Gets the position of both encoders and stores the positions in the dictionary of states where they can be accessed, for example, as robot. state ['L1pos'] and robot.state ['L2pos'] .
EmStop (self):	Triggers an emergency stop from software. Power to all motors is cut and the code running on the msp430 stops responding until the emergency stop condition is lifted by calling EmStopReset.

EmStopReset (self) Resets the emergency stop condition.

zeroZaxis(self) Sets the position of the Z-axis stepper motor as the new zero position.

getZpos (self): Gets the position of the Z-axis stepper motor and stores it in the dictionary of states where it can be accessed, for example, as robot.state ['Zpos']

gotoZpos (self, position, doConfirm = 0): Commands the SCARA to move the Z-axis to the given position, which is relative to the zero position previously set. If doConfirm is set, the function does not return until the move has been completed.

moveJ (self, endAng1, endAng2): Commands the SCARA to perform a joint interpolated move to the arm position defined by the two angles, endAng1 for joint 1, and endAng2 for joint 2.

moveJcoord (self, xPos, yPos, armSol): Commands the SCARA to perform a joint interpolated move to the arm position defined by the given X and X coordinates. The inverse kinematics from (x, y) to angle 1, angle 2 are performed on the msp430. The move will be attempted using the requested arm solution, left arm or right arm, but if the position can not be reached with requested arm solution, the other arm solution will be used.

moveL(self, xPos, yPos, armSol): Commands the SCARA to move the tool center from its current position along a straight line path to the position defined by the given X and X coordinates. The inverse kinematics from (x, y) to angle 1, angle 2 are performed on the msp430. The move will be attempted using the requested arm solution, left arm or right arm, but if the position can not be reached with the requested arm solution, the other arm solution will be used. The SCAR may pause while doing the move to change arm solutions.

Running SCARA.py itself from the command line, that is, as the main function and not as an imported file, will create a SCARA object named robot. robot can be used to control the SCARA by directly calling the functions in the SCARA class, e.g., robot.zeroEncoders(). Moreover, the SCARA class can be imported into a Python script, and the SCARA object can be called, for example, in a loop to do a pick and place operation.

Running the file Compact Scara.py will create a control panel built using the tkinter Python library (fig. 1). The control panel provides most, but not all, of the functionality from the SCARA class.

When the program is first started, a pop-up menu appears allowing the user to select the MSP430 serial port. The control panel will not function until a serial port is selected.

To perform a joint interpolated movement using the control panel, the desired values for joint 1 angle and joint 2 angle are entered in degrees into the corresponding entry boxes, and the moveJ button is clicked.

To perform a joint interpolated movement to a point defined by (x, y) coordinates instead of joint angles, enter values in the x and y position boxes, and click the MoveJ coordinate button. The SCARA will use the arm solution selected with the arm solution radio buttons, if possible, else the alternate arm solution will be used.

To perform a linear move from the current position to an (x, y) coordinate, enter the (x, y) position as for a move J coordinate move, but click the Move Linear button.

To perform a movement in an arc, enter the starting and ending arc angles in degrees, the arc radius in centimeters, and click the Move Circular button. As for the other movement types, the arc movement will start from the current position.

To move the tool in the Z-axis, enter a value, positive for up or negative for down, in the Move Tool by entry box and click the Move Z by Distance button. Clicking the Home-Z axis button will define the down position of the tool, i.e., where it is engaged with the workspace.

Clicking the emergency stop button will immediately stop all movement of the arms and the Z-axis. The SCARA will be in the stopped state until the continue button is pressed.

Clicking the continue button will reset the emergency stopped state, either one caused by the emergency stop button on the control panel, or by the physical emergency stop button or the limit switches.

CAD MODELS

When 3D printing the joints L1 and L2, they must be printed lengthwise in the Z-axis of the 3D printer to ensure the proper fitting of the bearings on the circular mounting points. The base of the SCARA should be printed on its side so that there is no raft to clear in the place of the bottom motor M1. A file or sandpaper may be needed to round off the edges slightly to allow for a snug fit of the bearings. Solidworks part files for the SCARA base, joint L1, joint L2, tool mount, and sharpie tool can all be found on https://github.com/mattWonn/compactSCARA.

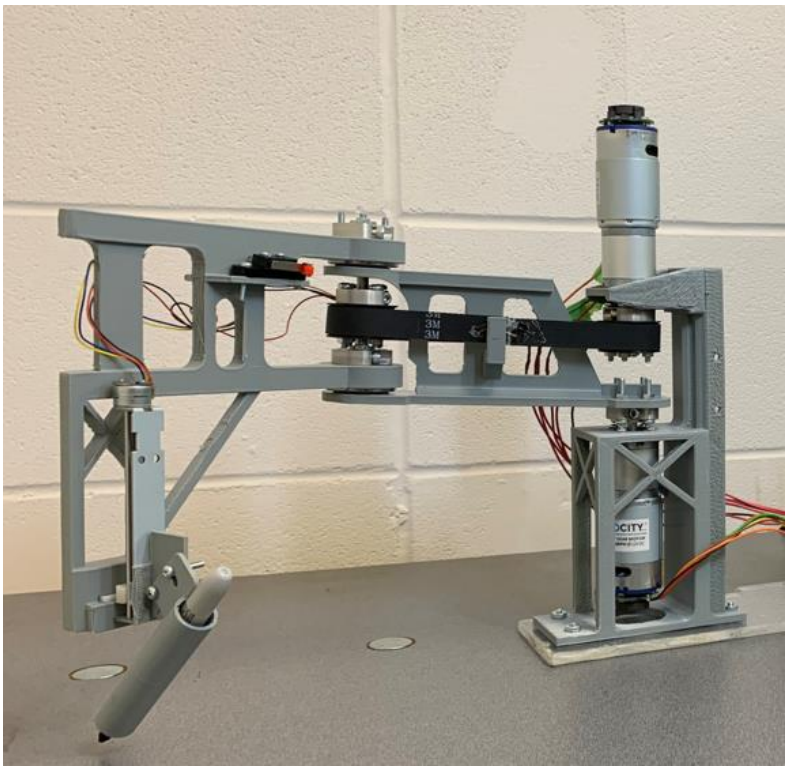

Figure 1: SCARA assembly