BCIT SCHOOL OF ENERGY

Mechatronics and Robotics Engineering Technology

ROBT 4491 : Mechatronics Project

FINAL PROJECT REPORT:
SCARA 2-AXIS MOTION CONTROLLER

Produced By:	
Bryn Rissling	A01319889
Date:	May 16, 2025
For:	Isaiah Regacho, Matthew Rockall

ACKNOWLEDGMENTS

I would like to give my heartfelt thanks to all of the people who made this project possible:

Isaiah Regacho and Autumn Friesen, who provided invaluable insight into the workings of the SCARA, Galil motion controllers, as well as general advice on wiring, circuit design, and mechanical assembly.

Darren Boehler, Galil technical suppourt enginner, who provided critical, timely technical suppourt on Galil C libraries.

BCIT faculty for maintaining the labs and facilities necessary to work on this project, as well as for providing logistical suppourt, getting parts where they needed to go.

ABSTRACT

One of the labs for ROBT 3341 has students programming a control algorithm for a SCARA. This robotic arm has been in use for decades, and the control system for it was extremely out of date. Using a set of Digital to Analog Converters (DAC's), that had to be connected directly to the motherboard of a PC running windows 2000, the system was fragile, inflexible, and severely outdated. At the beginning of the project semester, BCIT approached me and offered to sponsor me to replace these DAC's with a modern, flexible, and robust Galil 2 axis motion controller. With the additional requirement of writing C libraries to allow students to read encoder counts, and command the velocity and position of the motor. I accpted, and over the course of the semester I performed in-depth research on the necessary API's and tools I would need, designed the physical circuits, programmed the necessary libraries, and succesfully completed the project by the may 9th deadline.

CONTENTS

FIGURES

1 INTRODUCTION

Our schools robotics lab uses a SCARA to teach students on the fundamentals of their operation. This SCARA was incredibly old, with a set of DAC's (Digital to Analog Converters) connected directly into the motherboard of a PC running windows 2000. I was approached by the project organisers to replace this outdated hardware with a more modern, flexible, and common motion controller. This would allow the SCARA to be controlled by any modern PC via a simple USB cable, as well as being able to take advantage of the full suite of Galil tools. Such as those for performing inverse kinematics, or for executing smooth motion profiles. This would also make the system far easier to maintain, as the technology is common, and has excellent techincal suppourt from its manufacturer, Galil.

2 SYSTEM OVERVIEW

The SCARA's motors and encoders connect to a set of H-sub's, which then connect to the motion controller, which then connects to a PC via USB. The user then writes C code using libraries that I created. These libraries send commands to the motion controller, which then actuates the motion of the SCARA, giving students control over its velocity, and its position. Typically this programming would be done in Galil's specialized IDE, but for these labs, the students are expected to write with C code to interface directly with the encoder counts. The functions to perform these operations are described below. They will write their program inside of a "student code" function, which will be run after a setup function, and before a closing function.

2.1 SYSTEM REQUIREMENTS

The functional requirements of this system can be split into two parts: The physical requirements, and the software requirements.

For the physical requirements, the Galil motion controller needed to be connected to the SCARA, such that it could properly command motion and read encoder counts. It needed an E-STOP to either fully shut off the system, or command the arm to stop, in a way that overrides any other commands. And it needed a Power Supply Unit (PSU) to provide power to both the motion controller, and the SCARA. See below for a physical block diagram of the system.

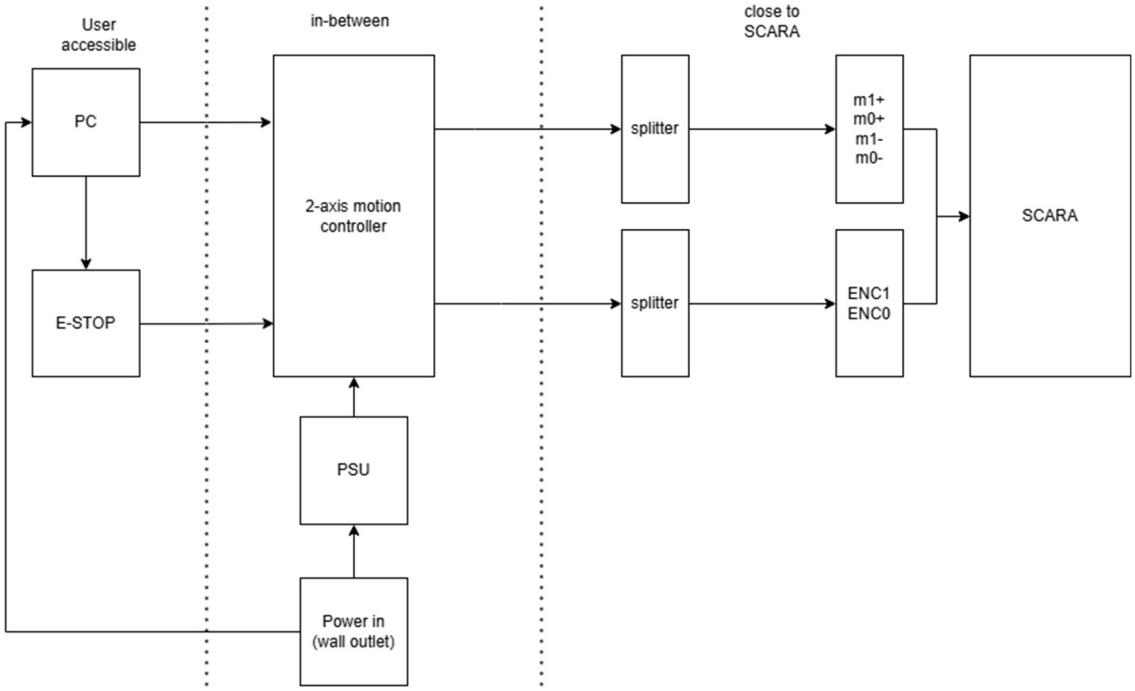


Figure 1: motion controller overview

For the software requirements, I had to program libraries that would allow students to do all the things outlined in the previous years SCARA lab. This included directly reading encoder counts, modifying the current encoder count, commanding the arm to move at a given speed, and commanding the arm to move to a given position. Other functions in use for the lab were acquired from libraries unrelated to the Galil, and as such where outside the scope of this project.

2.2 SOFTWARE

The libraries I wrote for this project are stored in SCARA_interface.c, and their prototypes are stored in SCARA_interface.h. I used a set of libraries that Galil provides to write these functions. These libraries provide me with functions to open and close connections to the motion controller and to send/receive commands. The functions that *I* wrote essentially package these Galil library functions up into a form that's easy for a student to utilize, without them having to do any research into Galil docs. There's also a great deal of software and firmware on the motion controller itself, namely the code required to accurately track and command the motion of the arm, in order to command its velocity, or its position. Further details of what I wrote for this project are listed in section 3.

2.3 PSU

The PSU had to provide enough power to supply both the Galil, and the SCARA. As the Galil consumes far less power than the robotic arm, our primary concern was ensuring that it could supply the motors on the arm. After looking over the specifications of the motor's datasheet, we settled on a 12Amp, 24V PSU sold by Galil, and purchased alongside the other components for this project. The PSU receives power from a wall outlet, and outputs power across 4 screw connectors, 2 for +Vcc, and 2 for Ground. A set of screw holes on the back of the PSU are used to attach it to the backplate of the housing used for this system.

2.4 MOTION CONTROLLER

The motion controller for this system had to be able to control 2 axes via a set of internal amplifiers. It also needed a casing to protect its own internal components. After determining the specific model number and full configuration, we ordered it alongside the PSU, and all cables/connectors needed to interface it with the SCARA. Unlike the PSU, the motion controller had a rail on the back, with a small, high strength spring, that would allow it to mount to a rail attached to the back panel of the system housing. While not fully stationary, the spring applied enough force that was sufficient to keep it in place for our application.

2.5 E-STOP

An extremely important part of this project the Emergency Stop button. The button used is a normally closed pushbutton switch, that when pressed physically cuts off power from going to the Galil. The E-STOP has 2 lines, one of which I use for the amplifier power, and the other I use for the Galil power. This has the additional benefit of easily allowing the user to turn the Galil on/off, allowing for the controller to be fully power cycled, and reset.

2.6 CABLES/CONNECTORS

The necessary cables for this project included:

A pair of 1 meter long, 26 pin cables that provided access to the digital I/O of each axis, which crucially included ports to read the encoder pulses of the motors.

Power connectors, including a 2 pin connector to provide power to the amplifiers, and a 6 pin connector to provide power to the Galil.

A pair of 4 pin motor connectors to connect the output of the amplifiers to the input of the SCARA motors.
Notably only 2 of the 4 are in use, since these cables are generalized for both DC and AC motors. The motors of the SCARA are DC, and so only use 2.

3 SOFTWARE LIBRARIES

Here are detailed descriptions of each of the functions I wrote for this project. More technical definitions are available in both Student_code.c , and the associated User Guide.

setMotorSpeed: Commands a given motor to move at a given speed. The motor selection and speed are both represented by integers, and are passed into the function when it was called. The speed is entered in units of encoder counts per second, and the motor selection can be either 0 or 1, depending on the motor being selected.

independantMove: Moves each motor to a given position at a given speed. Halts the program until the motor arrives at its destination.

getEncoderCount: Returns the encoder count of the corresponding motor passed into its argument. Using the same selection as setMotorSpeed, either 0 or 1.

findIndex: Moves the given motor to the nearest index pulse in the positive direction. Uses Galil internal algorithms to move to the index pulse, then slowly move back to account for any overshoot. Afterwards, it defines the position as 0. This function is necessary for performing homing sequences.

setPosition: Defines the current position of a given motor as a given encoder count. Useful for certain homing sequences, or for performing relative motion.

limpArm/unlimpArm: By default, when the arm is still, it applies whatever force it can to resist moving from its current position. And if it is moved from its current position, it quickly moves back to where it was before. The limpArm command undoes this control, and shuts off all power going to the arm, allowing students to move the arm without resistance. unLimpArm reapplies this force, and prepares the arm for further use.

3.1 SOFTWARE DESIGN

My design process for this project started out right at the beginning, with a set of block diagrams and pseudocode. Although I didn't know what specifically I had to do to write the necessary functions, I did have a general idea of what Galil offered, and so was able to plan out what I needed to research well in advance of reading any Galil API documents.

You can find my original pseudocode in the appendix. This was the base from which I determined what I needed to research in order to accomplish the requirements of this project. The final program includes far more than what is described. Since once I started writing my code, I discovered additional functionality that needed to be added.

3.1.1 HARDWARE DESIGN

For this system, the primary challenge was correctly wiring the motion controller to the SCARA. To achieve this, I noted all of the connections that I would need to make, looked through the user guide provided by Galil, and created a wiring diagram that illustrated the connections that I needed to make, shown below.

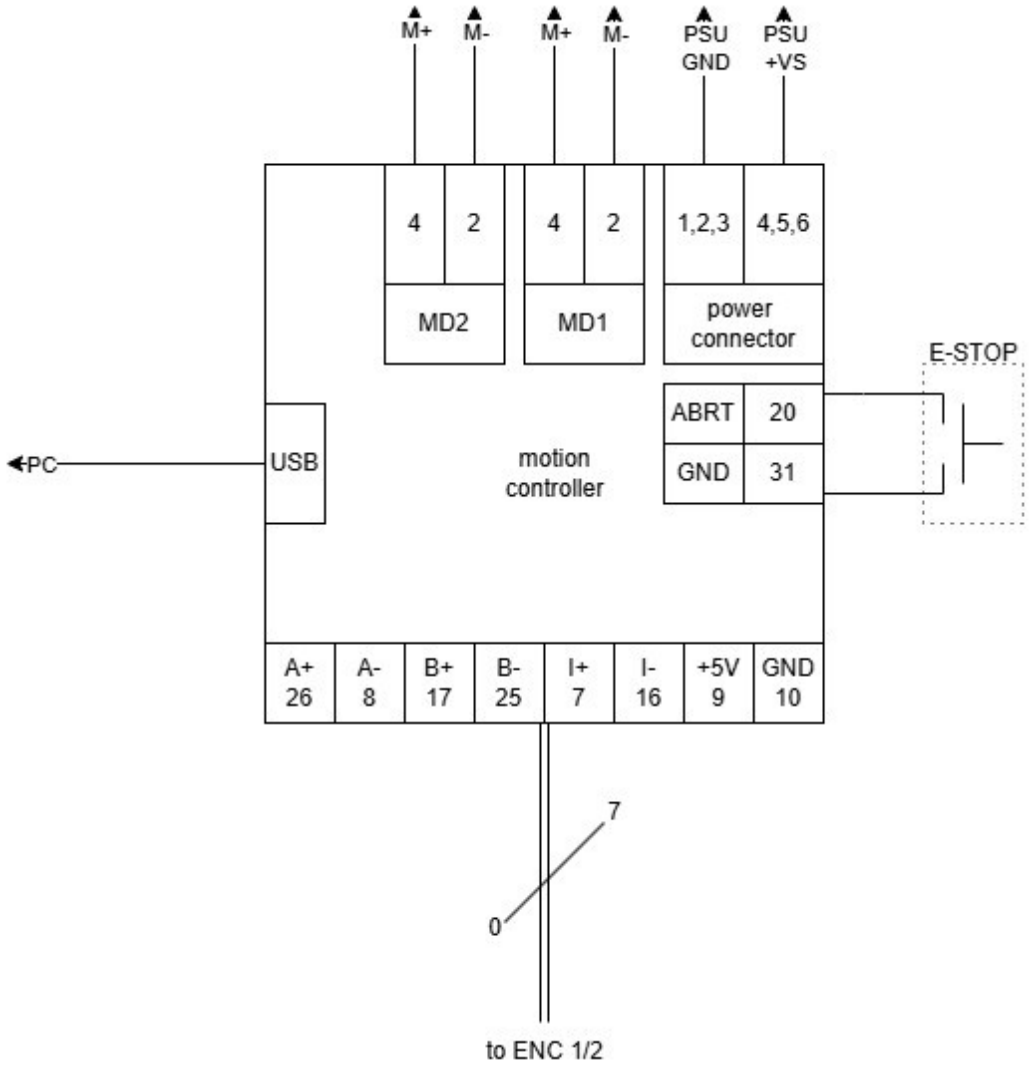


Figure 2: Motion controller wiring diagram.

This was the primary diagram that I worked off for the project.

Note: During the process of wiring the circuit, I decided to use the E-STOP to shut off power to the power connectors, rather than to close a connection to the ABRT pin. The schematic is otherwise accurate to the final build.

3.1.2 HARDWARE IMPLEMENTATION

Here are some notes on how I wired the circuit:

To connect the power of the Galil to the PSU, I needed a cheap, fast, and malleable method of connecting wires. For this purpose, I decided to use several Wago connectors, which currently connect the output of the power supply to the input of the E-STOP. And the output of the E-STOP to the power input of the Galil.

To screw the power supply onto the back panel of the system housing, I had to remove the back panel, and drill 4 holes into it. Due to personal error, these 4 holes are misaligned with where they need to be, and as such, I could only fit 3 screws into the PSU. Since the power supply won't be moving very much, I decided that this wasn't a substantial problem, and as such I moved forward with other, more important parts of the project.

4 System Verification

For this system, the only real verification that was necessary was ensuring that the functions I wrote operated correctly. To this end, I wrote a simple test program to demonstrate that the functions I created worked correctly. The full program can be found in Demo_code.c and Demo_code.h respectively. The demo code is run inside of student code, the same way future students will write their code, and it exclusively uses the library functions that I wrote for students to use.

It starts by running studentBasicTest, which commands the arm to move at varying velocities, waiting for a set period of time in between each command. Each velocity that it sets is the opposite of the one it set for that motor before, such that by the time the program finishes it arrives back at where it started. Although it accrues errors over multiple repetitions, it arrives close enough to confirm that the velocity control works as intended.

Afterwards, it tests the position control functions by commanding the first arm segment to move to +90 degrees, and the second arm segment to move to -45 degrees. It then immediately commands the arm to move to -90 and +45 degrees respectively. It also uses getEncoderCount to read the encoder count of the motors, calculates what angles they correspond to, and prints them to the command window. It does all of this with sufficient accuracy to confirm the operation of these functions.

After that it limps the arm, and sets the encoder counts of both arm segments to 0 twice. In testing I place break points in between these resets, move the arm, and demonstrate that it reads the position correctly. The program then unlimps the arm, and continues on with the program.

Finally, it uses a set of functions that are included in SCARA_interface.c, but not core to the requirements of this lab. Using contour movements and an inverse kinematics algorithm I wrote, it draws a set of lines to create a square. A minor bug in my algorithm causes the third line segment to frequently skip, drawing an arc from the start of the line segment to its end. It draws individual lines without issue though, and since this type of movement isn't part of the core requirements, this bug is acceptable for the final submission. But worth noting, in case it gets used in the future.

5 Project Summary

The goal of this project was to replace the old DAC's used to control our robotics lab SCARA, with a more modern 2 axis motion controller. And to write a set of C libraries that would allow students, not familiar with Galil API's, to directly read encoder counts, and directly command the velocity, and position of the SCARA.

5.1 System Specifications

TO RETURN TO In this section, I will detail the physical dimensions of the system housing, and the details of cable length, including the primary power cable.

Physical dimensions:

Length:16”

Width:18”

Height5”

Maximum external cable length:

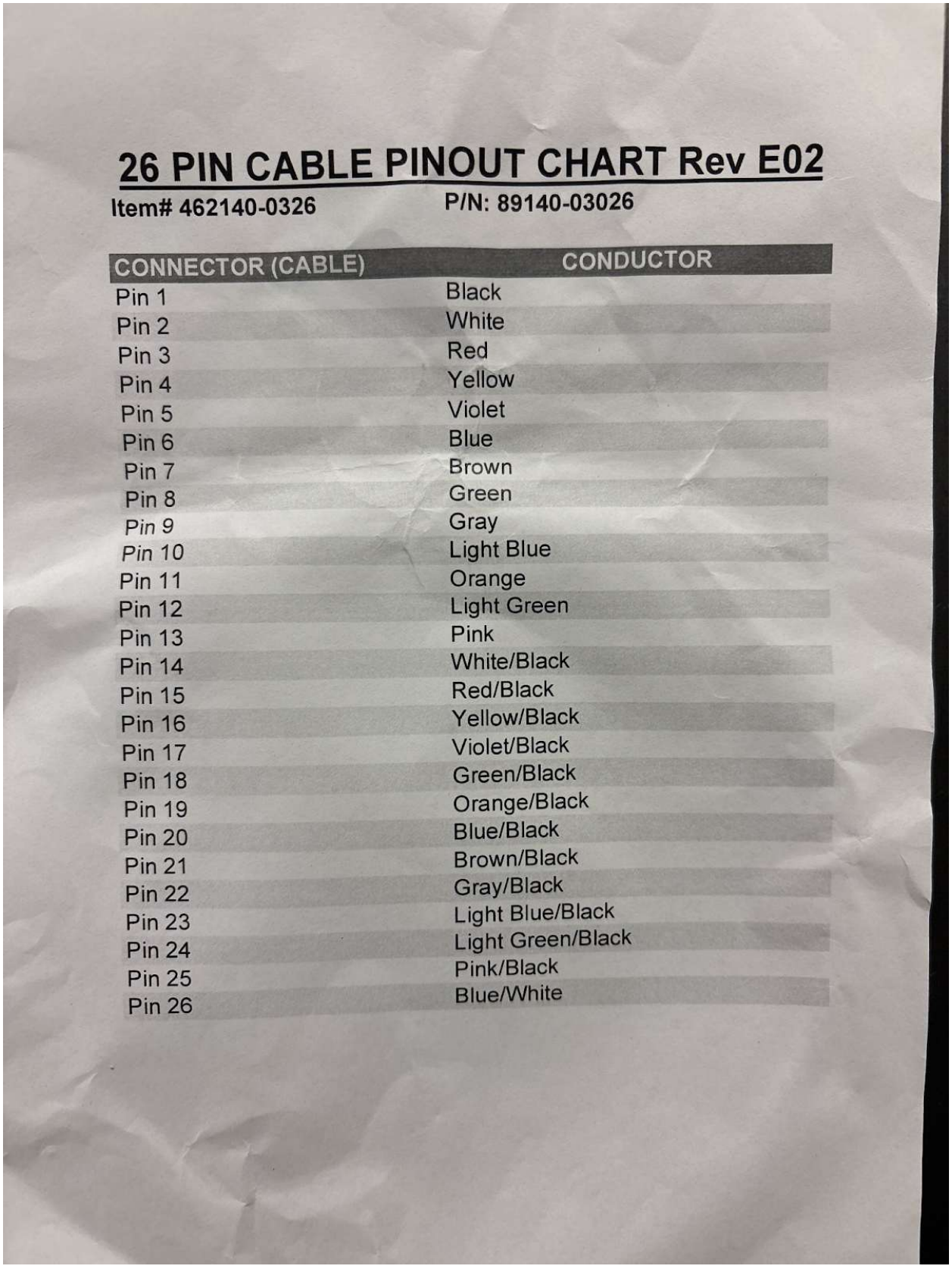
Power cable:48”

Digital I/O: 24”

Motor power: 24”

E-STOP:48”

Cable pinout:



26 PIN CABLE PINOUT CHART Rev E02

Item# 462140-0326 **P/N: 89140-03026**

CONNECTOR (CABLE)	CONDUCTOR
Pin 1	Black
Pin 2	White
Pin 3	Red
Pin 4	Yellow
Pin 5	Violet
Pin 6	Blue
Pin 7	Brown
Pin 8	Green
Pin 9	Gray
Pin 10	Light Blue
Pin 11	Orange
Pin 12	Light Green
Pin 13	Pink
Pin 14	White/Black
Pin 15	Red/Black
Pin 16	Yellow/Black
Pin 17	Violet/Black
Pin 18	Green/Black
Pin 19	Orange/Black
Pin 20	Blue/Black
Pin 21	Brown/Black
Pin 22	Gray/Black
Pin 23	Light Blue/Black
Pin 24	Light Green/Black
Pin 25	Pink/Black
Pin 26	Blue/White

5.2 Recommendations/Future Work

Exposed wiring cleanup:

One of the cables for the DIO ports has the caps that previously prevented wires from touching removed. When they do make contact, it causes a short circuit, and problems with the system. I strongly recommend that a permanent insulation solution be implemented to prevent this.

Permanent housing placement:

Currently, the housing for the control system is not attached to anything. While the box has the capacity to be mounted to a metal panel, I did not have the time to install such a system. Currently, it needs to be placed on a table next to the SCARA. In the future, it would be beneficial to mount it to the side of the SCARA's table, similar to how the Gantry robot is set up.

Further C library development:

The library functions I wrote do not have any error checking, and they have many inefficiencies. It's also likely that other methods of calling the functions or passing/retrieving data would be better than what I've created. For all of these reasons, it would be beneficial for future engineers to further iterate on what I've written. Adjusting them for ease of use, or to add functionality that I haven't implemented.

6 Conclusion

Despite the relative mundanity of this project, and the relatively small scale corresponding to it being a solo project. Its worth celebrating that It was completed successfully and on time. It also didn't require an excessive amount of overtime, nor did I lose any sleep trying to finish it. I met the demands of my sponsor, and delivered a fully functioning robotic arm, to be used in SCARA labs for years to come. The code that I wrote and the wiring that I implemented will be used by students for years to come. And serve as the bedrock for all future expansions of this system.

I learned as much working on this project as I did from any course I took as part of the Mechatronics and Robotics program. And as a capstone project it's something that I'm immensely proud of. I find it greatly satisfying that my final project is also a major contribution to this program. And that future students, following the same path I took, will have their experience radically altered by the work I have done. No longer having to deal with the inconveniences of an incredibly old PC.

7 Apendix

Pseudocode for the first draft of the software libraries I wrote:

getEncoderCount(encoder){

Declare a GDataRecord struct called Data

Declare an int called retVal

Update data using Grecord

If encoder = 0, set retVal to ecoderA of the Data Struct

Else, set retVal to encoderB of the Data struct

Return retVal

}

setMotorSpeed(motor, speed){

If motor = 0, use the jog command to set axis 0 to speed

Else, use the jog command to set axis 1 to speed

}

Start-up code{

Open connection to controller, using Gopen

Set record rate, using gRecord

SetMotorSpeed(0, 0)

SetMotorSpeed(1, 0)

Anything else that needs to happen to allow students to set motor speeds, and read encoder counts.

}

Power-down code{

setMotorSpeed(0,0)

setMotorSpeed(1,0)

Anything else that needs to happen to safely stop the SCARA.

Close communications, using Gclose

```
}

Student code{

//STUDENT CODE GOES HERE

}



Main{

Run start-up code

while(E-STOP isn't triggered){

Run student code

}

Run power-down code

}
```