Home-Owned Robotic Arm

Group 10 Authors:

Hani Bdeir - *Computer Engineering*

Hailey Gorak - *Computer Engineering*

Emanuel Alvarez - *Electrical Engineering*

Jackson Jacques III - *Electrical Engineering*

Nkunu Nuglozeh - *Electrical Engineering*

Mentor, Sponsors, and Major Contributors:

Dr. Chung Yong Chan - *Mentor/Coordinator*

Dr. Mike Borowczak - *Review Committee*

Professor Mark Maddox - *Review Committee*

Dr. Varadraj Gurupur - *Review Committee*

Table of Content

Chapter 1: Executive Summary.. 1
Chapter 2: Project Description..2
2.1 Motivation and Background.. 2
2.2 Current Existing Products/Inspiration...3
2.3 Goals and Objectives... 3
2.3.1 Detailed Description of Features and Functionalities:...5
2.3.2 Motor Control Subsystem Hardware Requirements for Features and Functionality:.5
Table 2.3.2 - Hardware Requirement Table.. 5
2.3.3 Teleoperation Subsystem:.. 6
2.3.4 Software:..8
2.4 Engineering Specification..8
Table 2.4 - Specifications Table...9
2.5 Block Diagrams.. 9
2.5.1 Hardware Block Diagram:..9
Figure 2.5.1 - Hardware Block Diagram.. 10
2.5.2 Software Block Diagrams:...10
Figure 2.5.2-1 - Software Block Diagram... 12
2.5.3 House of Quality:...12
Figure 2.5.3 - House of Quality Diagram.. 13
Chapter 3: Research and Investigation...13
3.1 Joysticks:..13
3.1.1 Analog vs. Digital:...13
Table 3.1.1 - Analog vs. Digital Comparison Table...14
3.1.2 Potentiometer vs. Hall Effect:...14
Table 3.1.2 - Potentiometer vs. Hall Effect Comparison Table............................. 16
3.1.3 Applications and Selection Criteria:.. 17
Table 3.1.3 - Part Price and Comparison.. 18
3.2 Buttons:...18
3.2.1 Button Types:... 18

Table 3.2.1-1 - Button Type Comparison Table..21
Table 3.2.1-1 - Button Type Comparison Table Continued.................................. 21
3.2.2 Applications and Selection Criteria:.. 22
Table 3.2.2 - Part Price and Comparison... 22
3.2 Motor Types...23
Table 3.2. - Motor Type Comparison Table..24
3.2.1 Stepper Motor Comparison:...24
3.2.2 NEMA 17 Stepper Motor:... 25
3.2.3 NEMA 23 Stepper Motor:... 25
3.2.4 NEMA 34 Stepper Motor:... 26
3.2.5 Selection Choice:.. 27
Table 3.2.5 - NEMA Stepper Motor Comparison Table..27
3.4 Motor Controller...28
3.4.1 Controller Types:...28
3.4.2 Applications and Selection Criteria:... 29
3.4.3 Conclusion:.. 29
3.5 Sensors:...30
3.5.1 Position Sensor Comparison:..30
3.5.1-1 AS5600 Magnetic Encoder (Chosen):.. 31
3.5.1-2 CUI AMT10:...31
3.5.1-3 MagAlpha MA702:.. 31
3.5.1-4 Selection Choice:..32
Table 3.5.1-4 - Position Sensor Comparison Table..32
3.5.2 Current Sensor Comparison:..33
3.5.2-1 Hall Current Sensors:...33
3.5.2-2 Shunt Resistors:.. 36
3.5.2-3 Rogowski Coil:...37
3.5.2-4 Current Transformers:.. 41
3.5.2-5 Current Sensor Technological Choice:... 43
Table 3.5.2-5 - Current Sensors Technological Comparison Table....................... 43
3.5.2-6 Current Sensor Price Selection Choice:...44
Table 3.5.2-6 - Current Sensor Price Comparison.. 45

3.6 End Effectors:.......... 45
  3.6.1 Vacuum Gripper:.......... 46
  3.6.2 Parallel Gripper:.......... 46
  3.6.3 Angular Gripper:.......... 47
  3.6.4 Electromagnetic Gripper:.......... 47
  3.6.5 Gripper Comparison:.......... 47
    Table 3.6.5-1 - Gripper Type Comparison Table.......... 48
    Table 3.6.5-2 - Hardware Comparison Table.......... 49
3.7 Microcontroller:.......... 50
    Table 3.7 - Microcontroller Comparison Table.......... 52
3.8 Communication Channels:.......... 53
  3.8.1 Serial Communication:.......... 54
    Table 3.8.1 - Communication Channel Comparison Table.......... 55
  3.8.2 Bus Protocols:.......... 55
    Table 3.8.2 - Bus Comparison Table.......... 57
  3.8.3 CAN Buses:.......... 57
    Table 3.8.3 - CAN Bus Comparison Table.......... 60
  3.8.4 USB Communication Technology:.......... 60
    Table 3.8.4 - USB Comparison Table.......... 62
3.9 Software Comparison:.......... 62
  3.9.1 Operating System Software:.......... 63
    Table 3.9.1 - Software Comparison Table.......... 65
  3.9.2 Simulators:.......... 65
    Table 3.9.2 - Simulator Comparison Table.......... 66
  3.9.3 Programming Languages:.......... 67
    Table 3.9.3 - Programming Language Comparison Table.......... 68
  3.9.4 Control Algorithms.......... 68
    Table 3.9.4 - Control Algorithm Comparison Table.......... 70
3.10 Power Supply:.......... 71
  3.10.1 Battery Powered Supplies:.......... 72
  3.10.2 Switching Power Supplies:.......... 75

3.10.3 Regulated Power Supplies:.......... 78
3.10.3 Power Distribution Table:.......... 81
Table 3.10.3 - Power Distribution Table..........81
3.10.4 Power Supply Technological Choice:..........82
Table 3.10.4 - Power Supplies Technological Comparison Table.......... 82
3.10.5 Power Supply Part Choice:..........83
Table 3.10.5 - Power Supply Price Comparison..........83
Chapter 4: Standards and Design Constraints.......... 84
4.1 Hardware Standards and Constraints:..........85
4.1.1 Joystick Standards:..........85
4.1.1-1 ISO 9241-410:.......... 85
4.1.1-2 ISO 10218-1:.......... 86
4.1.1-3 Design Constraints:..........87
4.1.2 Button Standards:..........87
4.1.2-1 IEC 60947-5-1:..........87
4.1.2-2 Design Constraints:..........88
4.1.3 CAN Bus:..........89
4.1.3-1 Standards:.......... 89
4.1.3-2 Constraints:..........89
4.1.4 USB Cable Standards:..........90
4.1.4-1 USB 2.0:.......... 90
4.1.4-2 USB 3.0 and 3.1:..........90
4.1.4-3 USB 4.0:.......... 91
4.1.4-4 Key Features:..........91
4.1.5 NEMA Stepper Motor Standards:..........92
4.1.6 Power Supply:..........93
4.1.6-1 Standards:.......... 93
4.1.6-2 Constraints:..........94
4.1.7 Current Sensors:..........96
4.1.7-1 Standards:.......... 96
4.1.7-2 Constraints:..........96
4.2 Software Standards and Constraints:.......... 97

4.2.1 Programming Languages:.... 98
4.2.2 Compatibility:.... 99
4.2.3 Memory and Program Size:.... 99
4.2.4 Processing Power:.... 99
Chapter 5: Comparison of ChatGPT with Search Engines.... 99
5.1 ChatGPT.... 99
5.2 Search Engines.... 101
5.3 Practical Applications and Experiences.... 102
5.4 Pros and Cons:.... 103
5.5 Examples:.... 104
Chapter 6: Hardware Design.... 105
6.1 Joystick Subsystem PCB.... 105
6.1.1 Joystick Modules:.... 105
Figure 6.1.1 - Joystick Module Schematic on Fusion 360.... 107
6.1.2 Mechanical Buttons:.... 107
6.1.2-1 General Purpose Buttons:.... 107
Figure 6.1.2-1 - General Input Buttons Schematic on Fusion 360.... 109
6.1.2-2 Reset Button:.... 109
Figure 6.1.2-2 - Reset Button Schematic on Fusion 360.... 110
6.1.3 Micro Usb Connection:.... 111
Figure 6.1.3-1 - Micro Usb 5-pin Schematic on Fusion 360.... 112
6.1.4 16MHz Crystal Oscillator:.... 112
Figure 6.1.4 - 16MHz Crystal Oscillator Schematic on Fusion 360.... 113
6.1.5 Decoupling Capacitors and LEDs:.... 113
Figure 6.1.5-1 - Power Decoupling Capacitors Schematic on Fusion 360.... 114
Figure 6.1.5-2 - LED Schematic on Fusion 360.... 115
6.1.6 MCU Connections:.... 116
Figure 6.1.6-1 - ATMEGA32U4 Schematic 1 of 2 on Fusion 360.... 117
Figure 6.1.6-2 - ATMEGA32U4 Schematic 2 of 2 on Fusion 360.... 118
Figure 6.1.6-3 - ICSP Header Schematic on Fusion 360.... 119
6.2 Motor Subsystem PCB.... 119

6.2.1 Position Sensor....................120
Figure 6.2.1 - Position Sensor (Rotary Encoder) Schematic on Fusion 360......121
6.2.2 Current Sensor....................121
Figure 6.2.2 - Current Sensor Schematic on Fusion 360....................122
6.2.3 CAN Bus....................122
Figure 6.2.3-1 CAN Bus Schematic on Fusion 360.................... 123
Figure 6.2.3-2 CAN Bus Node Connections.................... 124
6.2.4 Microcontroller Connections.................... 124
Figure 6.2.4 - Microcontroller Connections Schematic on Fusion 360....................125
6.3 Raspberry Pi/Computer Subsystem.................... 126
6.4 Power Distribution Subsystem....................127
Figure 6.4-1 - 24V to 3.3V Buck DC-DC Converter Schematic on Fusion 360.................... 128
Figure 6.4-2 - 3.3V to 5V DC-DC Boost Converter Schematic on Fusion 360...129
Figure 6.4-3 - 24V to 5V DC-DC Buck Converter Schematic on Fusion 360.................... 130
6.5 Mechanical Components....................130
6.5.1 Arm:.................... 130
Figure 6.5.1 - Arm CAD with End Effector.................... 131
6.5.2 End Effector:.................... 131
Figure 6.5.2 - End Effector CAD.................... 133
6.5.2 Base:....................133
Figure 6.5.3 - Base/Gearbox Motor Mount....................135
Chapter 7: Software Design.................... 135
7.1 Flowcharts, Class Diagrams, and State Diagrams.................... 135
Figure 7.1-1 - Simplified Software Flowchart....................137
Figure 7.1-2 - Class Diagram....................138
Figure 7.1-3 - State Diagram.................... 140
Figure 7.1-4 - Joystick Software Flowchart....................141
Figure 7.1-5 - Simulator Software Flowchart.................... 143
7.2 Robot Operating System.................... 143
Figure 7.2-1 - Node Structure Diagram.................... 144
7.3 Node Functionality....................145
Figure 7.3-1 - Joystick Node Data Transfer Diagram....................146
7.4 Microcontroller Code....................147

Figure 7.4-1 Microcontroller Logic....148
7.5 Use Case Diagram....149
Figure 7.5-1 - Use Case Diagram....150
Chapter 8: Fabrication of PCBs and Construction of Prototype....150
8.1 Fabrication of PCBs....150
8.1.1 Joystick PCB Fabrication:....150
Figure 8.1.1-1 - Joystick PCB Trace Diagram on Fusion 360....151
Figure 8.1.1-2 - Joystick PCB Pick-and-Place on JLCPCB....152
Figure 8.1.1-3 - 3D Model of Joystick PCB on Fusion 360....153
8.1.2 CAN Bus PCB Fabrication:....153
Figure 8.1.2-1 - CAN PCB Trace Diagram Fusion 360....155
Figure 8.1.2-2 - Finalized CAN PCB Front, All Surface-Mount Components....156
Figure 8.1.2-3 - Finalized CAN PCB Back, Bypass Capacitors....157
8.2 Construction of Prototype....157
8.2.1 Joystick Prototyping:....157
Figure 8.2.1 - Joystick and Button Prototyping with Arduino Leonardo....158
8.2.2 Motor Prototyping:....158
Figure 8.2.2 - Motor Prototype Connected with H-bridge, Microcontroller, and Gate Drivers....159
8.2.3 CAN Bus Communication Prototyping:....159
Figure 8.2.3 - CAN modules and microcontroller connections....161
8.2.4 Power Supply Prototyping:....162
Figure 8.2.4-1 - Power Supply Connections....162
8.2.5 Regulator Prototyping....164
8.2.6 MCU Prototyping....166
8.2.6.1 Unified MCU Prototyping....167
Chapter 9: System Testing and Evaluation....169
9.1 Testing Methodology....169
9.1.1 Motor Control Accuracy:....169
Table 9.1.1 - Motor Control Accuracy....170
9.1.2 Maximum Load Capacity:....170
Table 9.1.2 - Maximum Load Capacity....171

9.1.3 Peak Power Draw:....172
Table 9.1.3 - Peak Power Draw....173
9.1.4 Response Time:....173
Table 9.1.4 - Response Time.... 175
9.1.5 Software Testing:.... 175
Chapter 10: Administrative Content.... 177
10.1 Budget and Financing.... 177
Table 10.1-1 - Motor Subsystem Budget Table.... 177
Table 10.1-2 - Joystick Subsystem Budget Table.... 178
10.2 Bill of Materials.... 179
Table 10.2-1 - Bill of Materials for Motor Subsystem....179
Table 10.2-2 - Bill of Materials for Joystick Subsystem....180
10.3 Project Milestones....181
10.3.1 Milestones for SD1:.... 181
Table 10.3.1 - Senior Design 1 Milestones.... 181
10.3.2 Milestones for SD2:.... 183
Table 10.3.2 - Senior Design 2 Milestones.... 183
Chapter 11: Conclusion....184
Appendices....187
Appendix A – References.... 187
Appendix B – ChatGPT Prompt Examples.... 189
Appendix C – Programming Code....195
C.1 - Arduino Code for Joystick and Button Prototyping:.... 195
Figure C.1-1 - Setup code for digital and analog inputs on Arduino IDE.... 195
Figure C.1-2 - Running loop to test user inputs on Arduino IDE....197
C.2 Microcontroller and CAN Repository....197
Figure C.2-1 - Code used to send messages across our CAN network for all nodes.... 198
Figure C.2-2 - Code used to receive messages across our CAN network for all nodes.... 199
Figure C.2-3 Setup() code for all motor nodes....200
Figure C.2-4 - Loop() used for nodes 1 and 3....201
Figure C.2-5 - Motor driving function for nodes 1 and 3....201

Figure C.2-6 Code used to send current sensor and encoder data for nodes 1 and 3........ 203
Figure C.2-7 Code used for node 1 initialization........ 204
Figure C.2-8 Code used for node 2 initialization........ 206
Figure C.2-9 - Code used for node 3 initialization........ 208
Figure C.2-10 Code used for motor driving node 2........ 210
Figure C.2-11 Encoder sending code for node 2........ 210

Chapter 1: Executive Summary

Our project focuses on the development of a sophisticated robotic arm designed to perform precise and repetitive tasks in an industrial setting. The primary goal is to create a lower cost robot. At the heart of this project is the implementation of a Bang-Bang control algorithm running on an ESP32. Unlike traditional stepper motor control methods, which often focus solely on position, the Bang-Bang algorithm provides a simple and efficient way to manage motor operation by switching states based on preset thresholds. This method is effective for meeting the requirements of tasks that prioritize simplicity and responsiveness.

The Bang-Bang algorithm drives the robotic arm's motors by quickly switching between states to achieve the desired performance. This approach minimizes computational requirements, making it a practical choice for real-time motor control on the ESP32. While it is not as precise as more complex algorithms, the Bang-Bang method is well-suited for applications where consistent and rapid adjustments suffice. This simplicity ensures reliability and reduces the risk of overheating, a common challenge in motor control systems. By focusing on straightforward control, the robotic arm maintains functionality and responsiveness even under varying loads, meeting the requirements of industrial environments.

Our robotic arm is driven by three stepper motors, each managed by the Bang-Bang control algorithm. The use of an ESP32 as the central processing unit enables fast processing and communication, ensuring control commands are executed promptly. This setup allows the robotic arm to perform the necessary movements efficiently, making it suitable for various industrial applications such as assembly lines, material handling, and simple precision tasks.

In addition to the motor control system, our project includes a custom-designed joystick subsystem for teleoperation. This subsystem features dual joysticks for intuitive control, allowing the operator to manipulate the robotic arm's movements across multiple axes. The joystick subsystem also includes several buttons for executing specific commands, such as emergency stop, record and replay functions, and activating the end effector. The record and replay functionality is particularly noteworthy as it allows the robotic arm to automate repetitive tasks, mimicking modern robotic systems used in industrial assembly lines. The USB interface ensures reliable and fast communication between the joystick controller and the ESP32, enabling real-time control.

The end effector of the robotic arm is designed to provide versatile functionality, capable of handling various tasks such as gripping, lifting, and manipulating objects. This component is crucial for the robotic arm's application in different industrial scenarios, enhancing its adaptability and usefulness.

Our project also includes a robust power supply system to ensure reliable operation. The power supply is designed to provide stable voltage and current to all components, minimizing the risk of power-related issues during operation. This system is crucial for maintaining the consistency and reliability of the robotic arm's performance, especially in demanding industrial environments.

In summary, our project aims to deliver a highly functional and versatile robotic arm system that leverages the simplicity and adequacy of Bang-Bang control algorithms and robust hardware design. By addressing the limitations of traditional stepper motor control methods and incorporating innovative features such as the custom joystick subsystem and end effector, we aim to create a solution that meets the demands of modern industrial environments. This project not only enhances productivity and safety but also showcases the potential of streamlined control techniques in improving robotic systems

Chapter 2: Project Description

2.1 Motivation and Background

In the age of AI and automation, we stand on the cusp of a transformative era where robotics is set to revolutionize industries, streamline operations, and enhance human capabilities. Yet, as we look around, we find that there is a significant gap in the market for robotic hardware solutions that are both highly functional and cost-effective. Current industry models often require substantial investment, making them inaccessible for smaller businesses or research initiatives.

This is where our mission comes into play. We aim to provide a low-cost solution that retains the advanced features typically associated with high-end robotic systems. Our focus is on creating an industrial automation arm that redefines affordability while maintaining exceptional dexterity and precision. By leveraging innovative design principles and efficient hardware integration, our arm is built to perform tasks with a degree of precision and adaptability necessary for a wide range of applications.

Through thoughtful engineering and careful selection of components, we are designing a system that balances performance with affordability. Our approach is to democratize robotics, opening doors for industries, researchers, and educational institutions to access cutting-edge robotics without breaking the bank.

2.2 Current Existing Products/Inspiration

FANUC's M2000iA series of industrial arms are designed to hold heavy payloads and survive harsh environmental hazards. These arms are manufactured and sold to companies that focus on production, and require no other internal sensors or programming that integrates machine learning or robot vision. The company FANUC produces many more of these robotic arms with varying specific uses, such as their M800iA series, which holds a considerably lighter payload while boasting a higher level of agility. None of these series incorporate any machine learning or sensors to function properly.

ABB is another leading company in the production of robotic arms similar to what our project aims to achieve. Their IRB 1200 was used to assist in COVID19 vaccine production. The primary function of the arm is to be compatible with the needs of the buyer, since the end effectors of these products are variable according to the demand.

Similar products are constantly being updated by technology focused corporations, such as Intel, who are attempting to integrate machine learning and computer vision into robotic arms for safer and more precise movement. However, these practices are not industry standard and are not being implemented by many companies. These products are often deployed next to human employees, where they assist their engineering counterparts with various automated tasks.

Our product was inspired by these devices, primarily due to the lack of safety standards for these robots that has resulted in many unfortunate accidents for the humans that work alongside them. Our design is not aimed to just mimic these arms, but to add a solution to the safety concerns of these devices that are often overlooked, or simply not implemented. While many companies have achieved some truly incredible feats in regards to their machine's dexterity, compatibility, and adaptability, there is still an overwhelming majority that do not take the proper safety precautions.

2.3 Goals and Objectives

The main objective of this project is to design and build a dexterous, safe, and dynamic robotic arm. This robotic arm aims to achieve high precision and reliability in various tasks by utilizing precise position control. Below we have listed the main goals of our project in order of complexity and necessity to be completed, including stretch goals for possible future upgrades.

- **Basic Goals:**
    - Our robot will have precise position control of all three motors.
    - The robotic arm will detect its own position and have the ability to correct itself accordingly.
    - A joystick teleoperation device will be used for movement input with a record and replay button, emergency (E-Stop) off-button, and end effector activator button.
    - An end effector will be added to perform movements involving object manipulation, such as grabbing or picking up.
    - The robot should halt and reset when the emergency button is pressed.
- **Stretch Goals:**
    - Use deep learning and PyBullet/ROS Sim environment (Some Computer vision required) for visual object recognition.
    - Add force-feedback to our tele-operational device, allowing the user to feel resistance as the arm encounters it.
    - Integrate machine learning algorithms to adaptively improve the robot's performance over time
- **Objectives:**
    - We aim to detect the position of our robotic arm using rotary encoders as positional sensors.
    - The movements of our robotic arm will be controlled using a tele-operational device.
    - Precise position control will be achieved with a Bang Bang algorithm.

- The data received from the robotic arm's hardware will be sent through a CAN bus to communicate with our software.

2.3.1 Detailed Description of Features and Functionalities:

Our project will include precise and accurate motor control. It will consist of three separate motors, which will drive the joints and movement of our robotic arm. Each motor will utilize its motor control PCB that we will design and send out to fabricate. All of which will be connected to the central processing unit.

Included in the functionality of the arm will be real-time teleoperation controls, designed to be operated from a custom controller. Using advanced kinematic algorithms, we will be able to simulate smooth and flexible movements in the arm. The system will also incorporate features such as emergency stop functionality and record and replay capabilities, allowing the robot to perform repetitive tasks autonomously. The potential end effector of our robotic arm will be determined by the end usage, as the end effector should be interchangeable and compatible with many different products.

2.3.2 Motor Control Subsystem Hardware Requirements for Features and Functionality:

The Motor Control PCB is engineered to provide high precision and accuracy essential for controlling the motors of our robotic arm. This PCB features several critical hardware components and functionalities designed to meet the specific needs of our application, ensuring both operational excellence and safety in human interactive environments. Below is a table of the features and the required hardware we need to support the features.

Table 2.3.2 - Hardware Requirement Table

Feature	Required Hardware
High Precision and Accuracy in Control	● High Frequency Microcontroller ● Position Sensor ● Current Sensor ● Motor Driver

Feature	Required Hardware
High Speed Communication Across Boards and Teleoperation Device	CAN bus Transceiver IC such as mcp2515
High Power Handling	● Power Input/Output Ports ● Regulators for motor and microcontroller power
Firmware Programmability	Programming Interface
Precise Motor Feedback	Rotary Encoder Inputs Current Sensor
Future Sensor Integration	Provisions for Additional Sensor Inputs

2.3.3 Teleoperation Subsystem:

The Teleoperation subsystem was a critical component of the Home-Owned Robotic Arm project, designed to enable intuitive and precise control of the robotic arm through a custom-built teleoperation unit. This unit, inspired by the control systems of heavy machinery such as excavators, utilized dual joysticks to provide the operator with a natural and efficient interface for manipulating the arm in three dimensions. The left and right joysticks worked in combination to move the robotic arm in a way that was fluid and dexterous. Additionally, purpose-built end effectors, such as a parallel claw apparatus accompanied by an activator button, increased the capabilities of the robot.

To ensure seamless operation, the Teleoperation subsystem interfaced reliably with the central computing unit via a low-latency USB communication protocol. This communication was vital for promptly transmitting commands from the operator to the robotic arm. The subsystem processed inputs from the joysticks and buttons and converted them into desired control signals, which were sent to a Raspberry Pi. The Raspberry Pi executed a basic Bang-Bang algorithm, transmitting the joystick information to specific ESP32 microcontrollers that directly controlled the motors.

Safety was a paramount concern for the project. Therefore, the Teleoperation subsystem incorporated an essential safety feature, an emergency shut-down button. Furthermore, the system included a record and replay feature, allowing the operator to record sequences of movements and replay them as needed. This functionality was particularly useful for repetitive tasks, improving efficiency and consistency.

In terms of hardware, the subsystem was equipped with a high-performance microcontroller capable of handling input processing, communication, and safety features. During the prototype phase, an Arduino was used to validate the design. Once functionality was confirmed, a custom PCB using an ATmega32U4 microcontroller was fabricated. The PCB included power management systems to handle the power requirements of the joysticks and microcontroller, ensuring stable operation and effective management of power loss or spike scenarios. Input ports for connecting the joysticks and output ports for communication with the central computing unit and other robotic arm components were integrated into the design.

Software integration involved developing firmware to run on the Teleoperation subsystem. This firmware processed joystick inputs, managed communication with the central computing unit, and implemented safety protocols. Initially, this was developed on the Arduino platform, with plans to adapt it to the custom PCB later. The USB connection ensured efficient and reliable data exchange between the PCB and the Raspberry Pi, which served as the central processing unit.

Testing and validation of the Teleoperation subsystem were comprehensive. Initial bench testing verified the functionality of the joysticks and communication interfaces using the Arduino-based prototype. Integration testing ensured the subsystem worked seamlessly with the motor control subsystem and the central computing unit, guaranteeing smooth and precise operation of the robotic arm. Key performance metrics such as response time, communication reliability, and input processing accuracy were rigorously tested to ensure the system met the project's specifications.

By carefully designing, implementing, and validating the Teleoperation subsystem, the team created an intuitive, reliable, and safe control system for the Home-Owned Robotic Arm project, significantly enhancing its functionality and ensuring it met the needs of its users.

2.3.4 Software:

A primary computing device, such as a Raspberry Pi, was interfaced with our motor's PCB via a communications bus, specifically a CAN bus. This main computer was responsible for conducting most mathematical computations, which include processing inputs like the current position, motor current readings, and motor ID. Based on these inputs, it computed the target position and ensured safe rotational limits throughout the movement, according to the task requirements and positional feedback. Additionally, teleoperation inputs were received from the controller PCB, which communicated with the main computer through a USB connection.

Software was also used for simulation purposes. By defining a virtual environment which mimicked our robotic arm in terms of axis and physical ability, the movements of a virtual robot were simulated alongside the movements of our actual design. An advantage to using this method was that the physics environment was able to be defined with any parameters we needed, and we could test using any other viable robot model. This way we could adapt our simulations as our design evolved. However, we ended up with a rather simplistic robotic model to save on processing speed. The simulation serves to be feedback for the system operators, and has the potential to be turned into a control terminal for mapping all robotic movement.

Our simulator depended on data from our positional sensors (encoders), which took the data that was fed back to our Raspberry Pi computing device and used it to move the 3D model. This data was also used by our ESP32s to control and drive our motors using a closed-feedback loop, making our motors move while ensuring that a maximum and minimum limit could never be passed to prevent our 3D printed robotic frame from breaking.

2.4 Engineering Specification

Based on the goals and features in section 2.3, we needed measurable specifications for our design decisions to ensure that the robotic arm meets the required performance standards. These specifications encompass various aspects of the system, including motor control accuracy, load capacity, and response times. The following table lists these specifications in detail:

Table 2.4 - Specifications Table

Specification	Measure
Motor Control Accuracy	±10 degree
Load Capacity (W)	0.01kg<W<0.5 kg
Response Time of Autonomous Operation	< 500 ms
Response Time Teleoperation	< 500ms
Arm Reach	< 0.152 Meter
Peak Power Draw	<550 watt

2.5 Block Diagrams

To demonstrate the flow of our project and the purpose of each component, we have created diagrams for both the hardware components and the software that we utilized in the project. This includes a diagram for our interacting components within the software section, providing a comprehensive visual representation of our system architecture and operational flow. These diagrams help to ensure clarity and coherence in our design approach.

2.5.1 Hardware Block Diagram:

The hardware block diagram below provides a clear overview of the system design. It features four main sections: Joystick PCB, Motor PCB, CAN Bus PCB, and Power Supply. The Joystick PCB Housing includes components like dual joysticks and an interface circuit for operator input, which has its own MCU and sends signals to the central CPU, that being a Raspberry Pi. The Motor PCB Housing contains motor controllers and sensors to manage the robotic arm's movements. The Power Supply ensures all parts receive stable power. Each section is color-coded to show which team member is responsible and the status of each component, specific power lines, whether it needs to be acquired, is under investigation, is being designed, prototyped, or completed. This layout helps in understanding the project and keeping track of progress.

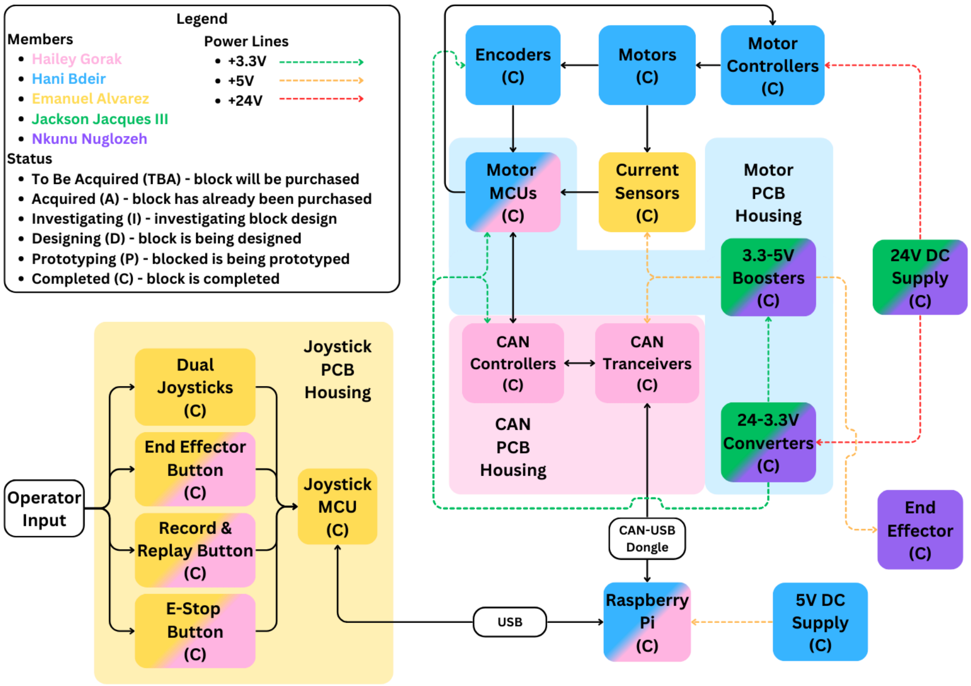


Figure 2.5.1 - Hardware Block Diagram

2.5.2 Software Block Diagrams:

Below is a color-coded description of the division of labor for the flow of software and coded processes of our project. The movement of our arm is determined by the analog input of our joystick, which also has the capability to record and replay movement from the joystick. This feature allows for repetitive tasks to be automated, increasing efficiency and consistency in operations. Using a mix of the ROS interface and the PyBullet library, we were able to simulate our robotic arm in real-time, along with the analog input and operation of our device. This feature is capable of being used for feedback loops between the simulator and the robotic arm, and given less time constraints, could be implemented to adjust the motor movements while the human operator is moving the device.

The programming done for our arm also involves solving mathematical operations based on positional encoder input to correctly judge the robotic arm's current motor angle. This encoder data allows us to set limits for our robotic arm's movement in all three directions, sending data from our encoders, which processes the data. This data is then sent across our communication bus to our Raspberry Pi 5 computer, which uses the angular data in several ways, including adjustment of simulation, setting upper and lower movement limitations, and allowing the arm to find its default home position.

The flow of our software operation begins at the joysticks, which are simple analog inputs. These inputs are detected directly by our computer, which scales the inputs down from 0-65565 to the same scale that our positional encoders use. These values vary from motor to motor, and are based on the real physical reach and limits we wished to set for each joint. These values are usually between 500 steps, however, and could range from 1900-2400 for a single joint. Once these values have been processed, we use a CAN communication bus to send the scaled data to our ESP32 microcontrollers. These microcontrollers detect the specific input ID, and trigger the corresponding function. Among the functions that are triggered via the microcontroller include the motor driving logic, the emergency stop, and packaging of encoder and current sensor data. Packaged data is sent across the same CAN lines.

Each operation can be seen in the flowchart below, including software decisions and which device is currently making the decision. Microcontroller behavior is in charge of our closed-loop functions, making sure our motor targets reach their absolute values, which allow them to then return to a home value. The encoder data read from the current motor values then gets sent over to the Raspberry Pi, which uses the values to change the simulator's movements, displaying the robot in real time to the human operator. We can see that on our microcontroller side, each microcontroller node is sent a code to determine if a button has been pressed or if joystick data has been run through to the microcontrollers.

The triangular nodes on the graph below represent actions that the program takes after making a decision, represented by the rectangular blocks. The program handles most of the decisions and actions, while the users only input analog joystick controls.

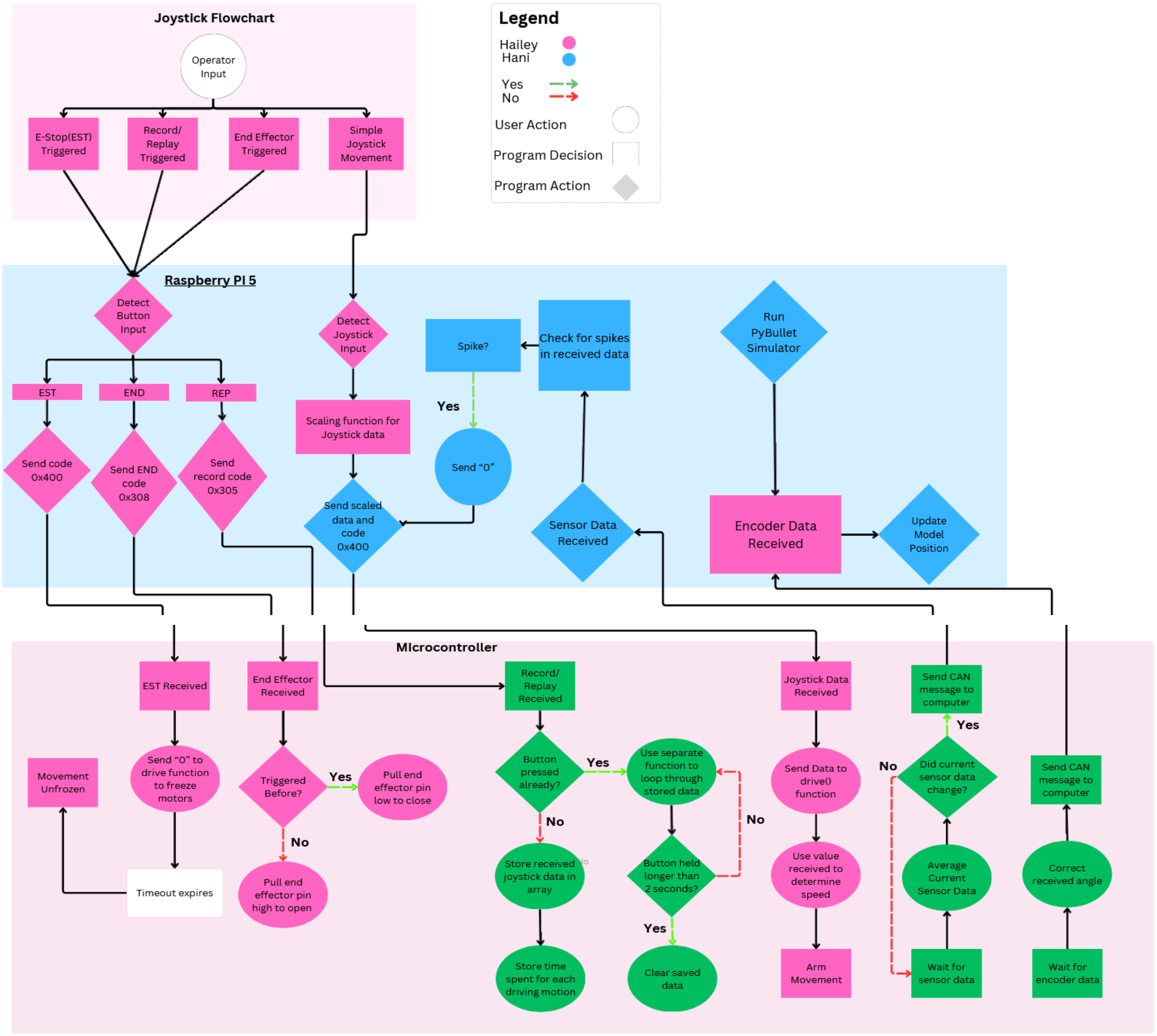
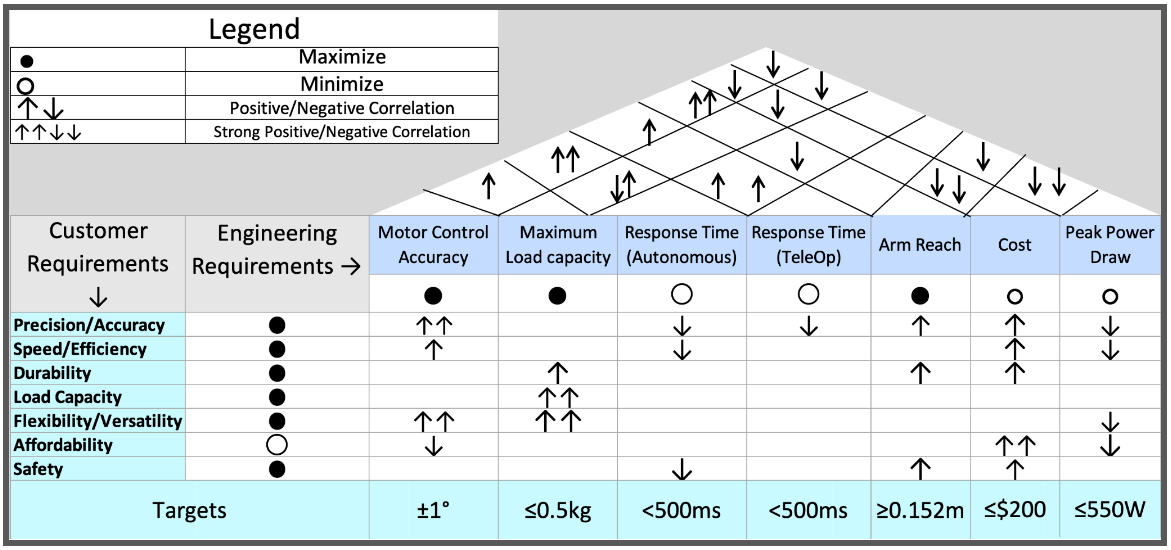


Figure 2.5.2-1 - Software Block Diagram

2.5.3 House of Quality:

The House of Quality diagram in Figure 2.5.3 below illustrates the correlation between customer requirements and engineering specifications for our robotic arm project. Its purpose was to translate customer needs, such as precision, efficiency, load capacity and safety, into measurable targets like motor control accuracy, response time, and peak power draw, which guided our design decisions. The diagram ensured that each engineering requirement aligned with and supported our overall project goals.



Legend	
●	Maximize
○	Minimize
↑↓	Positive/Negative Correlation
↑↑↓↓	Strong Positive/Negative Correlation

Customer Requirements ↓	Engineering Requirements →	Motor Control Accuracy	Maximum Load capacity	Response Time (Autonomous)	Response Time (TeleOp)	Arm Reach	Cost	Peak Power Draw
		●	●	○	○	●	○	○
Precision/Accuracy	●	↑↑		↓	↓	↑	↑	↓
Speed/Efficiency	●	↑		↓			↑	↓
Durability	●		↑			↑	↑	
Load Capacity	●		↑↑					
Flexibility/Versatility	●	↑↑	↑↑					↓
Affordability	○	↓					↑↑	↓
Safety	●			↓		↑	↑	
Targets		±1°	≤0.5kg	<500ms	<500ms	≥0.152m	≤$200	≤550W

Figure 2.5.3 - House of Quality Diagram

Chapter 3: Research and Investigation

3.1 Joysticks:

3.1.1 Analog vs. Digital:

Analog joysticks provide continuous variable input, which means they can detect a range of positions along the X and Y axes. This feature allows for finer control and more precise movements, making them ideal for applications that require smooth and proportional control, such as robotic arms, gaming controllers, and simulation equipment. Analog joysticks typically use either potentiometers or Hall-effect sensors to measure position. Potentiometers vary resistance based on the joystick's position, translating to an analog voltage signal. Hall-effect sensors, on the other hand, use magnetic fields to generate an analog signal corresponding to the joystick's position. This continuous input allows for nuanced control of speed and direction, essential in applications requiring high precision.

Digital joysticks, in contrast, provide discrete input signals, typically in four or eight directions. Each direction is a binary signal, either on or off, which simplifies the interface and reduces the complexity of the control system. This makes digital joysticks suitable for applications where precise proportional control is less critical. They are commonly used in basic directional control for machinery or simple user interfaces. For example, in a basic machinery control setup, a digital joystick's binary signals might be sufficient to direct movement without needing the fine control an analog joystick provides.

Table 3.1.1 - Analog vs. Digital Comparison Table

Feature	**Analog Joystick (Chosen)**	**Digital Joystick**
Control Precision	High (Continuous Input)	Low (Discrete Input)
Ease of Use	Moderate (Requires Calibration)	High (Plug and Play)
Durability	Moderate (Drift)	High (Less Complex)
Cost	Low to High	Low to High
Applications	Robotic Arms, Simulations	Basic Machinery

In most real-world applications, analog joysticks are preferred when nuanced control is crucial. For instance, in aviation simulators, analog joysticks enable pilots to practice with realistic, smooth control inputs, closely mimicking real aircraft behavior. On the other hand, digital joysticks are often found in simple arcade games where the precise direction rather than the smoothness of control is paramount.

3.1.2 Potentiometer vs. Hall Effect:

Hall effect joysticks operate based on the Hall effect principle, where a magnetic field induces a voltage across an electrical conductor. This voltage is proportional to the magnetic field's strength and is used to determine the joystick's position. Key components include a magnet attached to the joystick shaft and a Hall sensor that measures the magnetic field's strength. These joysticks are known for their robustness, precision, and durability. Hall effect sensors are contactless, which means they do not suffer from mechanical wear and tear, leading to a significantly longer lifespan compared to potentiometer-based joysticks. Hall effect joysticks can achieve lifespans exceeding 10 million cycles, making them highly suitable for applications requiring long-term reliability and minimal maintenance [1][2][5].

Hall effect sensors are valued for their high precision. They offer a typical resolution of 12 bits and linearity around ±0.5%, ensuring accurate and consistent performance. A 12-bit resolution means that the joystick can detect 4,096 distinct positions along its range of motion, which allows for very fine control. Linearity of ±0.5% means that the output is very consistent and closely follows the actual position of the joystick, with only minor deviations. This level of precision is crucial for applications in industrial controls, high-level robotics, and modern gaming where more precise inputs are critical [2][5]. Additionally, Hall effect joysticks are highly resistant to environmental factors such as dust, dirt, and moisture, often rated with IP protection levels of IP65 or higher. This makes them ideal for use in harsh environments where durability and reliability are paramount [2].

However, Hall effect sensors can experience drift, which is a gradual change in the sensor's output signal over time without any actual change in the measured input. Drift can be caused by various factors such as temperature fluctuations, aging of the sensor materials, or changes in the magnetic field strength. This drift can affect the long-term accuracy and reliability of the sensor, requiring periodic recalibration to maintain precision [4][5].

Potentiometer joysticks function by having a physical wiper that moves across a resistive element, changing the resistance and thus the output voltage proportionally to the joystick's position. While this technology is prone to mechanical wear over time, leading to reduced durability, it is simpler and more cost-effective. Potentiometer joysticks typically have a lifespan of 1 to 2 million cycles, significantly shorter than their Hall effect counterparts, but offer nearly the same capabilities [3][5].

Potentiometer joysticks offer lower precision compared to Hall effect joysticks. They usually provide a 10-bit resolution and linearity around ±1-2%. A 10-bit resolution means that the joystick can detect 1,024 distinct positions, which is adequate for many applications but not as precise as a 12-bit resolution. Linearity of ±1-2% means there is a bit more deviation from the actual position, making them less accurate than Hall effect joysticks. Although sufficient for less demanding applications, this lower precision can be a drawback in scenarios where fine control is necessary [3]. Furthermore, potentiometer joysticks are more susceptible to environmental conditions. Dust and moisture can interfere with the mechanical parts, leading to degraded performance and reliability over time [3][5].

In terms of cost, Hall effect joysticks are generally more expensive due to their advanced technology and superior durability. Prices for Hall effect joysticks typically range from $60 to over $120 but some third-party replacement parts for modern console controllers can be found for as low as $20 for a pair. In contrast, potentiometer joysticks are more affordable, with wide prices ranging from $2 to $50, making them a cost-effective option for applications where high precision and long-term durability are not as critical [2][3][5].

Table 3.1.2 - Potentiometer vs. Hall Effect Comparison Table

Feature	**Potentiometer (Chosen)**	**Hall Effect**
Control Precision	Moderate (10-bit resolution, ±1-2% linearity)	High (12-bit resolution, ±0.5% linearity)
Lifespan	Moderate (1-2 million cycles)	High (>10 million cycles)
Environmental Resistance	Moderate (IP50-IP65)	High (IP65 or higher)
Cost	Low ($2-$50)	Moderate to High ($20-$120)
Applications	Consumer electronics, basic machinery	Industrial controls, gaming, robotics

3.1.3 Applications and Selection Criteria:

Given the project requirements and budget constraints, potentiometer joysticks present a viable option for controlling stepper motors. While they offer lower precision and durability compared to Hall effect joysticks, their performance is sufficient for many applications. The cost savings can be significant, allowing the allocation of resources to other critical components of the project. Potentiometer joysticks are particularly suitable for applications where cost is a critical factor and extreme precision is not necessary.

Potentiometer joysticks function by varying the voltage output based on the position of their wiper, which can be read by an analog pin on a microcontroller, such as an Arduino. This varying voltage can be used to control the position and direction of a stepper motor, making potentiometer joysticks an effective tool for precise motor control in applications such as robotics and CNC machines.

To control a stepper motor using a potentiometer, the following components are typically required: a stepper motor, a motor driver, a potentiometer and a microcontroller. The connections include wiring a potentiometer’s middle pin to an analog input on the Arduino, connecting the outer pins to 5V and GND, and interfacing the stepper motor with the motor driver according to its specifications. The motor driver's control pins are then connected to the Arduino's digital output pins.

Using the Arduino IDE, a simple program can be written to read the potentiometer value and map it to a suitable range for controlling the stepper motor's speed. Controlling multiple motors would then just be a matter of having more potentiometers. An example code snippet that demonstrates how to achieve this can be seen in Appendix C.1, where we see the initialization of all the analog inputs and then a function to read them in an infinite loop. The IDE’s serial monitor can be used to read the output produced for prototyping.

This method provides a simple and cost-effective way to achieve variable speed control for our stepper motors. Potentiometer joysticks, due to their affordability, are well-suited for projects where budget constraints are a consideration. By using potentiometers to control stepper motors, we can implement effective and precise control for various applications, making them a valuable component in our project.

After evaluating various options, we have chosen the Ikpek 8pc potentiometer joysticks. This decision is based on several factors, including positive reviews and cost-effectiveness. The Ikpek joysticks have received an average rating of 4.4 stars from 64 reviews on Amazon, indicating their reliability and user satisfaction. Additionally, the pricing of $12.99 for 8 pieces makes them an economical choice compared to other options, such as the 10K 3D joysticks from DigiKey, which are priced at $2.30 each plus $45 shipping, and the Hall Effect joysticks, which are $14.99 for 4 pieces with no reliable reviews. These considerations make the Ikpek potentiometer joysticks the most suitable option for our project, balancing cost and quality effectively.

Table 3.1.3 - Part Price and Comparison

Feature	Ikpek Joysticks (Chosen)	10K 3D Joysticks	Ikpek Joysticks
Technology	Potentiometer	Potentiometer	Hall Effect
Reviews	4.4 Stars (64 Reviews)	0 Reviews	0 Reviews
Quantity	8pcs	1pcs	4pcs
Cost	$12.99	$2.30 + $45 Shipping	$14.99
Seller	Amazon	Digikey	Amazon

3.2 Buttons:

3.2.1 Button Types:

Buttons are essential components in our joystick controller, providing the interface for user inputs including our record and replay capability, emergency-off feature, and end effector activator. There are several types of buttons, each with unique characteristics that make them suitable for different applications.

Mechanical buttons are widely used in various applications due to their simplicity, reliability, and strong tactile feedback. These buttons operate through a metal or plastic contact mechanism that completes an electrical circuit when pressed, providing a clear and responsive user experience. The tactile feedback offered by mechanical buttons is one of their most significant advantages, as it gives users a distinct and satisfying sensation that confirms the button press. This feature is particularly important in applications where users need to receive immediate feedback, such as in keyboards and game controllers.

Mechanical buttons are constructed with several key components: the actuator, the contacts, and the housing. The actuator is the part of the button that the user presses. When the actuator is pressed, it moves the contacts together to complete the circuit. The housing encases these components, protecting them from external factors like dust and debris. Over time, however, the repeated physical motion can lead to mechanical wear, and the presence of dust or other contaminants can affect the button's performance and longevity.

Despite these potential drawbacks, mechanical buttons remain a popular choice due to their robustness and ease of use. Their reliability in delivering consistent performance makes them suitable for a wide range of consumer electronics. For example, in keyboards, the distinct click and tactile response of mechanical switches are preferred by many users over membrane or capacitive alternatives. Similarly, game controllers benefit from the precise and immediate feedback of mechanical buttons, which can enhance the gaming experience by providing a more responsive and immersive control system. Moreover, mechanical buttons are relatively easy to integrate into electronic systems. They typically require straightforward wiring and simple circuit designs, which can reduce the complexity and cost of manufacturing. This ease of integration, combined with their tactile benefits, makes mechanical buttons a preferred choice for many designers and engineers.

Membrane buttons are a popular choice in applications requiring a flat, sealed interface. These buttons operate through a flexible membrane layer that, when pressed, makes contact with a circuit underneath to complete it. This design offers several advantages, including low cost, sealing against dust and moisture, and customizable design options. Membrane buttons are often used in environments where protection from contaminants is critical, making them ideal for applications in remote controls, microwave ovens, industrial control panels, and many keyboards.

The construction of membrane buttons typically involves three layers: the top layer (graphic overlay), the middle layer (spacer), and the bottom layer (circuit). The graphic overlay is the part that users interact with, often printed with labels or icons indicating the button's function. The spacer layer contains holes that align with the button positions, allowing the top layer to press down onto the circuit layer when activated. The circuit layer, usually made of conductive traces printed on a flexible substrate, completes the electrical connection when pressed.

One of the key benefits of membrane buttons is their low manufacturing cost. The materials used are inexpensive, and the production process is straightforward, making them a cost-effective solution for mass-produced electronics. Additionally, the sealed design of membrane buttons provides excellent protection against dust, dirt, and moisture, which can prolong the lifespan of the device in harsh environments.

However, membrane buttons also have some disadvantages. They generally offer lower tactile feedback compared to mechanical buttons, which can be less satisfying for users who prefer a more distinct response. Additionally, the lifespan of membrane buttons is shorter, as the flexible membrane can wear out over time with repeated use. Despite these drawbacks, the balance of cost, protection, and ease of customization makes membrane buttons a suitable choice for many applications.

Capacitive buttons are an advanced type of button that detect touch through changes in capacitance, eliminating the need for physical pressure. This technology allows capacitive buttons to sense the presence of a finger or conductive object through a non-conductive material such as glass or plastic, making them ideal for modern, sleek designs in consumer electronics. Capacitive buttons are commonly used in smartphones, touchpads, and contemporary home appliances, where aesthetics and seamless interaction are paramount.

The working principle of capacitive buttons involves creating an electric field around the button area. When a conductive object, such as a human finger, comes close to the button, it disturbs the electric field, causing a change in capacitance. This change is detected by a microcontroller, which registers it as a button press. Because there are no moving parts, capacitive buttons offer exceptional durability and a long lifespan. They are resistant to mechanical wear and can operate reliably even in harsh environments, making them suitable for applications requiring longevity and minimal maintenance.

One of the main advantages of capacitive buttons is their versatility in design. They can be seamlessly integrated into smooth surfaces, providing a modern and clean look. Additionally, they can be used through various non-conductive materials, allowing for innovative design possibilities such as touchscreens and control panels behind glass or plastic surfaces.

However, capacitive buttons come with certain disadvantages. They are more expensive than mechanical or membrane buttons due to the complex circuitry and components required for their operation. Additionally, capacitive buttons can be less responsive in wet conditions or when the user is wearing gloves, as these factors can interfere with the capacitive sensing. Despite these challenges, the advantages of durability, modern design, and advanced functionality make capacitive buttons a preferred choice for high-end applications.

Table 3.2.1-1 - Button Type Comparison Table

Feature	**Mechanical Buttons (Chosen)**	**Membrane Buttons**	**Capacitive Buttons**
Ease of Integration	Easy to integrate with straightforward wiring. Typically requires simple connections.	Easy to integrate. Typically comes connected with a ribbon wire and headers.	Complex integration. Requires additional components and software support for capacitance sensing.
Tactile Feedback	Strong tactile feedback, providing a distinct click or press sensation.	Moderate tactile feedback, less pronounced than mechanical buttons.	Low to no tactile feedback. Typically relies on visual or auditory indicators.
Durability	Moderate durability. Susceptible to mechanical wear and environmental contaminants.	Good durability. Sealed against dust and moisture but has a shorter lifespan than mechanical buttons.	High durability. No moving parts, resistant to wear and environmental factors.

Table 3.2.1-1 - Button Type Comparison Table Continued

Feature	Mechanical Buttons (Chosen)	Membrane Buttons	Capacitive Buttons
Cost	Low to moderate cost. Affordable and cost-effective for most applications.	Low cost. Inexpensive to produce and integrate, making them budget-friendly.	Medium to high cost. More expensive due to complex circuitry and additional components.
Applications	Keyboards, game controllers, consumer electronics where strong feedback is important.	Keyboards, remote controls, microwave ovens, industrial control panels requiring flat and sealed interfaces.	Smartphones, touchpads, modern home appliances where sleek design and touch sensitivity are prioritized.

3.2.2 Applications and Selection Criteria:

Given the project requirements, the selection of buttons must balance cost, durability, and ease of integration. Mechanical buttons are suitable for applications requiring strong tactile feedback and straightforward integration. Membrane buttons are ideal for environments requiring sealed interfaces to protect against dust and moisture. Capacitive buttons are preferable for modern designs prioritizing touch sensitivity and durability, ensuring a sleek user experience in high-end applications.

After evaluating various options, we have chosen mechanical buttons for our project due to their strong tactile feedback and ease of integration. These buttons have proven reliability and user satisfaction in various consumer electronics applications, including keyboards and game controllers. Additionally, the cost-effectiveness of mechanical buttons makes them an economical choice, allowing us to allocate resources to other critical components of the project. They are the most common form of PCB mounted buttons and will be integral in maintaining user-friendly control interfaces in our design. Furthermore, their robust design ensures longevity and consistent performance in various operational conditions.

Table 3.2.2 - Part Price and Comparison

Feature	Daoki 4-pin Push Button (Chosen)	16-Key Membrane	HiLetgo Touch
Technology	Mechanical	Membrane	Capacitive
Reviews	5 (458 Reviews)	4.7 (35 Reviews)	4.4 (116 Reviews)
Quantity	100pcs	4pcs	10pcs
Cost	$5.99	$7.95	$8.49
Seller	Amazon	Amazon	Amazon

3.2 Motor Types

The selection of the appropriate motor type is crucial for the performance, efficiency, and cost-effectiveness of the robotic arm. Three main types of motors were considered: Brushed DC Motors, 3-Phase Brushless DC (BLDC) Motors, and 2-Phase Stepper Motors. Each type has its own set of characteristics that make it suitable for different applications.

Brushed DC motors are simple and widely used electric motors that use mechanical commutation to change the direction of current flow in the rotor. They offer simple control and drive circuitry, low initial cost, high reliability for simple applications, and a wide speed range with variable voltage. However, they require periodic maintenance due to brush wear, have lower efficiency compared to brushless motors, limited lifespan due to mechanical commutation, and potential for electrical noise due to brush arcing.

BLDC motors use electronic commutation instead of mechanical brushes, offering improved efficiency and reliability. They boast high efficiency and performance, low maintenance requirements, longer lifespan due to absence of brushes, excellent heat dissipation, and a high power-to-weight ratio. On the downside, BLDC motors require more complex and expensive control electronics, have a higher initial cost, involve a more complex design and manufacturing process, and require rotor position feedback for precise control.

Stepper motors are brushless DC motors that divide a full rotation into a number of equal steps, allowing for precise positioning without feedback in many applications. They offer high torque at low speeds, precise positioning without feedback in many applications, simple open-loop control for basic operations, excellent low-speed torque characteristics, and are cost-effective for positioning applications. However, stepper motors have lower efficiency at high speeds, potential for missed steps under high loads, and higher power consumption when holding position.

Based on the comparison, Stepper Motors are the most suitable choice for our robotic arm project. While they share the brushless design of BLDC motors, steppers are optimized for precise positioning and high torque at low speeds, which aligns perfectly with our project's requirements. Given our volume constraints, steppers offer the highest torque density at low speeds. Their ability to provide accurate control without complex feedback systems meets our need for precision while keeping costs manageable. The inherent holding torque of stepper motors is particularly advantageous for a robotic arm that needs to maintain positions steadily. Although typical BLDC motors offer higher overall performance, especially at high speeds, the stepper's combination of precise positioning, high low-speed torque, and relatively simple control makes them the optimal choice, balancing our specific performance needs with project constraints of volume, cost, and complexity.

Table 3.2. - Motor Type Comparison Table

Feature	2-Phase Stepper (Chosen)	Brushed DC	3-Phase BLDC
Torque at Low Speed	Very High	Medium	High
Efficiency	Medium	Medium	High
Control Complexity	Medium	Low	High
Maintenance	Low	High	Low
Cost	Medium	Low	High

3.2.1 Stepper Motor Comparison:

In selecting the appropriate stepper motor for our robotic arm project, we focused on motors that adhere to NEMA (National Electrical Manufacturers Association) standards, ensuring compatibility and reliability. Our selection process involved a thorough evaluation of three NEMA standard motor sizes: NEMA 17, NEMA 23, and NEMA 34. Each motor size offers distinct advantages and trade-offs in terms of torque, size, cost, power requirements, and overall suitability for our specific application.

The selection of an appropriate stepper motor is crucial for the performance, efficiency, and cost-effectiveness of our robotic arm. Factors such as torque output, physical dimensions, power consumption, and cost all play significant roles in determining the most suitable motor for our needs. We conducted a comprehensive analysis of each motor option, considering not only their individual specifications but also how they aligned with our project's unique requirements and constraints.

3.2.2 NEMA 17 Stepper Motor:

The NEMA 17 stepper motor (0.8Nm, 2.3A, 42x42x67mm) represents the smallest and most cost-effective option among the three considered. This motor is widely used in smaller 3D printers, CNC machines, and compact robotics applications due to its balance of size and performance.

Advantages of the NEMA 17 include its compact size, which makes it ideal for applications where space is at a premium. Its lower current draw of 2.3A means it can be powered by smaller, less expensive motor drivers and power supplies. The NEMA 17 we evaluated offers the highest torque in its class at 0.8Nm, which is impressive for its size.

However, despite these advantages, the NEMA 17's torque output falls short of our project requirements. While it could potentially be used in less demanding joints of the robotic arm, it lacks the necessary power for the main load-bearing axes. Additionally, to achieve the required torque using NEMA 17 motors, we would need to incorporate gearing systems, which would increase complexity, cost, and potential points of failure.

The NEMA 17, while not suitable for our main drive system, could potentially find use in end-effector mechanisms or other auxiliary systems within our robotic arm where high torque is not a primary requirement.

3.2.3 NEMA 23 Stepper Motor:

The NEMA 23 stepper motor (2.4Nm, 4A, 57x57x82mm) presents an excellent balance of performance and cost, making it a strong contender for our robotic arm project. This motor size is commonly used in larger 3D printers, CNC routers, and various industrial automation applications.

One of the standout features of this NEMA 23 model is its impressive torque output of 2.4Nm, which meets our project's requirements without the need for additional gearing in most axes. This high torque-to-size ratio is crucial for our application, as it allows for direct drive configurations, simplifying our mechanical design and reducing potential points of failure.

The motor's current draw of 4A is manageable with standard motor drivers, striking a good balance between power and ease of integration. Its dimensions (57x57x82mm) offer a significant upgrade in power from the NEMA 17 while still maintaining a relatively compact form factor, crucial for our space-constrained design.

Importantly, the NEMA 23 provides the best cost-to-torque and volume-to-torque ratios among the options considered. This efficiency in both space utilization and cost makes it an attractive option for our project, where we need to balance performance with budget constraints.

The primary challenge with the NEMA 23 is that it may be slightly underpowered for the most demanding joints in our robotic arm, potentially requiring gearing for these specific applications. However, its overall balance of characteristics makes it a strong candidate for the majority of our arm's joints.

3.2.4 NEMA 34 Stepper Motor:

The NEMA 34 stepper motor (4.8Nm, 6.0A, 86x86x80mm) offers the highest torque output among our considered options. This motor size is typically used in heavy-duty CNC machines, large-format 3D printers, and industrial automation systems where high torque and precision are paramount.

The most significant advantage of the NEMA 34 is its substantial torque output of 4.8Nm. This high torque capacity ensures that the motor can handle even the most demanding joints in our robotic arm without the need for gearing, potentially simplifying our mechanical design in high-load areas.

However, the NEMA 34's advantages come with several trade-offs. Its larger size (86x86x80mm) may pose challenges in our space-constrained design, potentially requiring significant modifications to our arm's structure to accommodate it. The higher current draw of 6.0A necessitates more robust and expensive motor drivers and power supplies, increasing the overall system cost and complexity.

Furthermore, the NEMA 34 is the most expensive option among the three, which could significantly impact our project budget, especially if used for multiple joints. While its high torque output is impressive, it may be overspecified for many of the joints in our robotic arm, leading to unnecessary costs and power consumption.

3.2.5 Selection Choice:

After careful consideration of our project's specific needs and constraints, we have selected the NEMA 23 stepper motor as the optimal choice for the majority of joints in our robotic arm. This decision is primarily driven by its superior balance of performance and cost-effectiveness. The NEMA 23's torque output of 2.4Nm meets our project's requirements for most joints while maintaining a compact form factor (57x57x82mm) that aligns well with our design constraints. Its 4A current draw is manageable with standard motor drivers, avoiding the need for specialized high-voltage components required by larger motors like the NEMA 34.

Furthermore, the NEMA 23 offers the best volume-to-torque and cost-to-torque ratios, critical factors in our space-constrained and budget-conscious design. While the NEMA 17 is more cost-effective, its insufficient torque output would compromise the arm's performance. Conversely, although the NEMA 34 provides higher torque, its increased size, cost, and power requirements outweigh the benefits for most of our applications.

For the most torque-intensive joint (likely the base rotation), we may consider using a single NEMA 34 motor to ensure adequate power without requiring complex gearing. This hybrid approach allows us to optimize performance where it's most needed while maintaining cost-effectiveness throughout the rest of the design.

In conclusion, the NEMA 23's balance of torque, size, and cost makes it the most suitable choice for the majority of our robotic arm's joints, allowing us to achieve our performance goals while adhering to our project's budgetary and design constraints. The selective use of a NEMA 34 motor for the highest-torque application provides a comprehensive solution that addresses all of our project's motor requirements.

Table 3.2.5 - NEMA Stepper Motor Comparison Table

Feature	NEMA 17	NEMA 23 (Chosen)	NEMA 34
Model	0.8Nm 2.3A	2.4Nm 4A	4.8Nm 6.0A
Dimensions (mm)	42x42x67	57x57x82	86x86x80

Torque (Nm)	0.8	2.4	4.8
Current (A)	2.3	4	6
Holding Torque (Nm)	0.8	2.4	4.8
Cost	Lowest	Medium	Highest
Torque-to-Cost Ratio	Low	High	Medium
Torque-to-Volume Ratio	Medium	High	Medium
Torque-to-Weight Ratio	Medium	High	Medium
Ease of Integration	High	Medium	Low
Gear Reduction Requirement	High	Low	Very Low
Overall Suitability for Project	Low	High	Medium

3.4 Motor Controller

The selection of a motor controller is critical to the performance, reliability, and ease of integration of the robotic arm system. Given the choice of stepper motors, the motor controller must efficiently drive the selected motors while providing precise control over their operation. Three main motor controllers were evaluated: the A4988, DRV8825, and TB6600.

3.4.1 Controller Types:

A4988 Stepper Driver:
The A4988 is a compact and affordable stepper motor driver commonly used in 3D printers and small robotics applications. It supports microstepping up to 1/16 and operates with a maximum current output of 1A per phase. While the A4988 is cost-effective and easy to use, its current-handling capacity falls short of the requirements for our selected NEMA 23 stepper motors, which draw up to 4A. This makes the A4988 unsuitable for our application, as it would lead to underpowered performance and potential overheating.

DRV8825 Stepper Driver:
The DRV8825 offers a higher current-handling capacity of up to 2.5A per phase, with microstepping support up to 1/32. It is widely used in applications requiring moderate torque and precision. While the DRV8825 outperforms the A4988 in terms of power output and precision, it still cannot fully meet the current demands of the NEMA 23 stepper motor without significant modifications or cooling enhancements. Additionally, its limited power-handling capacity excludes it from consideration for use with the NEMA 34 motor required for the base joint.

TB6600 Stepper Driver (Chosen):
The TB6600 stepper driver is a robust and versatile option designed for high-power stepper motors. It supports currents of up to 4.5A per phase, making it well-suited for the NEMA 23 and NEMA 34 motors used in our project. The TB6600 provides microstepping capabilities up to 1/32, ensuring precise control for smooth and accurate movements.

This motor controller also features built-in protection mechanisms, including overcurrent, overvoltage, and thermal protection, enhancing reliability during operation. The TB6600's ability to handle higher currents without overheating, combined with its compatibility with larger motors, makes it an ideal choice for our application. Although it is bulkier and more expensive than the A4988 and DRV8825, its performance and reliability outweigh these drawbacks.

3.4.2 Applications and Selection Criteria:

To meet the requirements of our robotic arm, the motor controller must:

1. Handle the current and voltage demands of the NEMA 23 and NEMA 34 stepper motors.
2. Support microstepping for precise movement control.
3. Provide robust protection features to ensure durability.
4. Be cost-effective relative to its performance and capability.

After careful evaluation, the TB6600 motor controller was selected for its superior power-handling capacity, reliability, and compatibility with our chosen stepper motors. Its ability to drive both the NEMA 23 and the torque-intensive NEMA 34 motor eliminates the need for multiple types of motor controllers, simplifying the overall system design.

3.4.3 Conclusion:

The TB6600 motor controller’s high current capacity, advanced features, and versatility make it the optimal choice for our robotic arm project. By ensuring precise and reliable operation of the NEMA 23 and NEMA 34 motors, the TB6600 contributes to the system's overall performance and robustness. Although it involves a higher initial cost than other controllers, its benefits in terms of capability and reliability make it a cost-effective solution for achieving the project’s objectives.

Table 3.4.3 - Motor Controller Comparison Table

Feature	A4988	DRV8825	TB6600
Max Current (A)	1.0	2.5	4.5
MicroStepping	1/16	1/32	1/32
Suitability for NEMA 23	Not	Limited	Yes

3.5 Sensors:

3.5.1 Position Sensor Comparison:

In selecting the appropriate position sensors for our robotic arm project, we focused on sensors that offer high precision, reliability, and compatibility with our control system. Our selection process involved a thorough evaluation of various types of position sensors, including magnetic encoders, optical encoders, and potentiometers. Each sensor type offers distinct advantages and trade-offs in terms of accuracy, resolution, cost, robustness, and overall suitability for our specific application.

The selection of appropriate position sensors is crucial for the performance, accuracy, and reliability of our robotic arm. Factors such as measurement accuracy, resolution, environmental robustness, and cost all play significant roles in determining the most suitable sensors for our needs. We conducted a comprehensive analysis of each sensor type, considering not only their individual specifications but also how they align with our project's unique requirements and constraints.

3.5.1-1 AS5600 Magnetic Encoder (Chosen):

The AS5600 is a contactless, on-axis magnetic encoder IC that provides precise angular position sensing. It operates using a simple two-wire interface (PWM or I2C), offering high resolution of 12 bits (4096 positions per revolution). Its contactless design ensures excellent durability and resistance to wear, making it ideal for long-term use in a robotic arm.

The AS5600 provides an absolute position output, eliminating the need for a homing sequence and ensuring position retention during power cycles—a critical feature for maintaining precise control and reducing setup time. With a price of approximately $3.50, the AS5600 is highly cost-effective, offering exceptional performance without straining the project budget.

Another notable feature is its compact size and ease of integration, which makes it suitable for space-constrained designs. Its robust performance under various environmental conditions further enhances its reliability in demanding applications. The AS5600 also includes user-programmable features such as output range and start/end positions, enabling customization for specific project requirements.

3.5.1-2 CUI AMT10:

The CUI AMT10 is a through-bore encoder offering a resolution of 8192 counts per revolution (CPR) and an accuracy of 0.0127 degrees. While it delivers high precision and is suitable for applications requiring extremely precise positioning, its mechanical complexity and higher cost ($24.00) make it less appealing for a cost-sensitive, space-constrained project like ours.

3.5.1-3 MagAlpha MA702:

The MagAlpha MA702 is an on-axis magnetic encoder offering a resolution of 4096 CPR and an accuracy of 0.03 degrees RMS. Its compact design and high-speed capability (up to 60,000 RPM) are advantageous in certain applications. However, at $5.10, it is slightly more expensive than the AS5600 while providing comparable resolution and accuracy, making it a less optimal choice given our project's constraints.

3.5.1-4 Selection Choice:

After evaluating the options, the **AS5600 magnetic encoder** was selected for our robotic arm project. This decision was based on its combination of high resolution, absolute position sensing, cost-effectiveness, and ease of integration.

The AS5600's absolute position output eliminates the need for a homing sequence, ensuring precise control without position loss during power cycles. Its 12-bit resolution meets our positioning requirements, and its robust, contactless design offers long-term durability with minimal maintenance. Furthermore, its low cost allows for greater budget allocation to other critical components of the project.

While the CUI AMT10 provides higher resolution, its cost and complexity make it less suitable for our application. Similarly, while the MagAlpha MA702 offers comparable performance, the AS5600's lower price and sufficient resolution make it the better choice.

In conclusion, the AS5600 strikes the best balance between performance, cost, and reliability, making it the optimal choice for our robotic arm's position sensing needs.

Table 3.5.1-4 - Position Sensor Comparison Table

Feature	**CUI - AMT10**	**REV Through Bore Encoder**	**AS5600**
Type	Through Bore	Through Bore	Through Bore
Resolution (CPR)	8192	8192	4096
Accuracy (deg)	0.0127	0.01	0.04
Max RPM	7500	10000	60000
Price	$24.00	$35.00	$5.10

3.5.2 Current Sensor Comparison:

A current sensor is a device designed to detect and convert electrical current into an easily measurable output voltage, which is proportional to the current flowing through the measured path. Historically, current sensors have played a crucial role in applications like robotic arms, where they ensured precise control, safety, and functionality. These sensors provided real-time data about the electric current passing through the motors, which helped engineers monitor and control the arm's movements effectively. In modern systems, a current sensor would continue to be vital for optimizing performance, preventing overloads, and detecting faults. Without such devices, ensuring the stability and efficiency of complex robotic mechanisms would be significantly more challenging.

3.5.2-1 Hall Current Sensors:

There are several types of current sensors that could be useful for a robotic arm, with one prominent type being Hall Effect sensors. Hall Effect sensors are specialized devices that utilize the Hall Effect principle to measure the electric current flowing through a conductor. These sensors provide a non-contact method of current measurement, which is especially advantageous in applications where isolation, safety, and precision are critical, such as in robotic systems. These sensors have played a crucial role in robotics by ensuring accurate current measurement and control. The Hall element is the core sensing component that generates the Hall voltage. The Hall element, the core sensing component, generates the Hall voltage and converts magnetic fields into measurable electrical signals. In robotic arms, Hall elements were pivotal for achieving precise motor control. For instance, Hall current sensors provided real-time feedback on the current supplied to the motors, allowing for precise adjustments to speed and torque. By monitoring current levels, these systems detected and responded to overcurrent conditions, protecting motors and associated electronics from potential damage. In modern implementations, Hall Effect sensors would continue to enhance safety and efficiency. For example, they would be essential in Joint Position Control, ensuring that motors receive the correct current for precise movement of the arm's joints. Additionally, control systems could dynamically adjust the current based on real-time feedback from Hall sensors, ensuring smooth, accurate movements. In scenarios where safety is paramount, sudden changes in current could indicate collisions or obstructions, prompting the system to stop the arm and prevent damage. The ability of Hall Effect sensors to deliver such critical functionality highlights their indispensable role in both historical and contemporary robotic applications.

There are several significant advantages to using Hall Effect current sensors. Firstly, Hall current sensors measure the magnetic field generated by current flow without making direct electrical contact with the conductor. This non-invasive method not only enhances safety but also reduces wear and tear, eliminates the risk of disrupting the circuit being measured, and ensures reliable performance. This capability played a crucial role in applications requiring high precision and minimal interference. Another key advantage is that these sensors provide electrical isolation between the input (current-carrying conductor) and the output (measurement signal). This feature would be invaluable in protecting sensitive electronics from high voltage spikes, surges, and noise, thereby enhancing safety and reliability. Furthermore, Hall Effect sensors can measure both AC and DC currents over a wide range. This versatility allowed their use in diverse applications, such as industrial automation and robotics, where varying current levels and waveforms were common.

In modern systems, Hall sensors would likely continue to offer this flexibility, enabling a single sensor type to handle multiple current monitoring tasks efficiently. Additionally, these sensors are typically encased in robust housings that protect them from environmental factors like dust, moisture, and mechanical stress. This durable construction has made them ideal for use in harsh environments, ensuring long-term reliability and requiring minimal maintenance. Overall, the unique combination of non-invasive operation, electrical isolation, and versatility underscores the indispensable role of Hall Effect sensors in both past and potential future applications

On the other hand, there are certain limitations to using Hall Effect current sensors. The performance of Hall Effect sensors can be influenced by temperature variations, which may require additional temperature compensation circuitry to maintain accuracy across a wide temperature range. Another limitation is that measurement inaccuracies could occur due to the sensor's zero-current output drifting over extended periods. This drift, if left unaddressed, might affect long-term precision in critical applications. Additionally, external magnetic fields can interfere with the sensor's measurements, necessitating careful placement and shielding to minimize interference from nearby magnetic sources. Lastly, Hall Effect sensors often have limited bandwidth, which affects their ability to measure high-frequency currents accurately. This limitation would be significant in applications requiring the monitoring of rapidly changing currents, such as in high-speed motor drives or power electronics. While these limitations exist, they do not overshadow the considerable benefits that Hall Effect sensors provide in various applications.

Hall Effect current sensors are essential components in modern robotic arms, serving multiple roles. One critical role is motor control and monitoring. Hall Effect sensors measure the current supplied to the motors that drive the robotic arm's joints and actuators. This capability is beneficial because it not only allows for precise control of motor torque and speed through accurate current measurement but also enables real-time feedback to the control system. Such feedback helps maintain optimal performance by allowing immediate adjustments during operation. Another key application for Hall Effect sensors in robotic arms is Joint Positioning and Movement. These sensors monitor the current supplied to each motor, enabling the control system to determine the position and movement of each joint. This functionality ensures that each joint moves to its correct position, which is vital for tasks requiring high precision.

Furthermore, Hall Effect sensors play a significant role in Force and Torque Sensing. By measuring the current, the torque produced by the motors can be inferred, as torque is directly proportional to current in DC motors. This capability is especially essential for applications requiring specific force control, such as gripping, lifting, or delicate manipulation of components and workpieces. This precise control helps prevent excessive force application, which could otherwise damage sensitive components or materials. Additionally, overcurrent protection is another role that the Hall sensors play when implemented in a robot arm. The sensors would detect abnormal current levels that would indicate potential issues, such as motor stalls or electrical faults. This is favorable because this would enable the control system to take protective actions, such as shutting down or reducing power to prevent damage, and assist in diagnosing issues, improving maintenance and reducing downtime. Finally, the sensors monitor the overall current consumption of the robotic arm to manage energy use effectively, which would help optimize energy usage, reducing operational costs. For battery-operated robotic arms, this is crucial for ensuring optimal battery usage and longevity.

Another critical component of Hall Effect current sensors is the magnetic core. The magnetic core serves several important functions, including magnetic field concentration and direction control, ensuring that magnetic field lines are directed perpendicularly to the Hall element for optimal sensing. Additionally, the reduction of external interference, achieved through the magnetic core's design, helps minimize noise and ensures accurate measurements. A feedback mechanism within the system maintains high accuracy and linearity by dynamically adjusting the sensing parameters. Finally, the signal conditioning circuitry within Hall Effect current sensors performs several crucial functions. For example, the raw Hall voltage generated by the Hall element is typically very small, necessitating amplification. Another vital function of the circuitry is filtering, which removes noise and enhances signal quality, ensuring the reliability of the output. In modern digital control systems, the analog signal produced by the Hall element must be converted into digital form. This process involves the use of analog-to-digital converters (ADCs), which digitize the amplified and conditioned signal.

3.5.2-2 Shunt Resistors:

Another type of current sensor that was considered was a shunt resistor current sensor. A shunt resistor current sensor is a simple and effective way to measure electrical current. It operates on Ohm's law, which states that the voltage drop across a resistor is directly proportional to the current flowing through it. There are two main components for a shunt resistor. One component would be a precision resistor with a very low resistance value (typically in the milliohm range). It is placed in series with the current path. Another component would be a differential amplifier or a voltage measurement device measuring the voltage drop across the shunt resistor.

There are several advantages when using shunt resistors. For instance, Shunt resistors offer high precision in current measurement. The voltage drop across the resistor is directly proportional to the current, providing an accurate representation of the current flow. Moreover, The design and implementation of shunt resistor-based current sensing circuits are straightforward. They require minimal components, making them easy to integrate into various systems. Finally, Shunt resistors are relatively inexpensive compared to other current sensing technologies like Hall-effect sensors or current transformers. There are several limitations to shunt resistors, however. Shunt resistors convert electrical energy into heat, which would lead to loss of power. This can be a concern in high-current applications where significant power dissipation might occur. Also.The voltage drop across the shunt resistor, although small, can affect the performance of low-voltage circuits. This is a critical consideration in sensitive electronic applications. Moreover, Excessive heat generated by the resistor can affect its accuracy and stability. Proper heat management, such as heat sinks or thermal management techniques, is necessary to maintain accuracy.

Robot arms, especially those used in industrial automation and precision tasks, require precise control and monitoring of their various components, particularly the motors that drive their movements. Shunt resistors would play a crucial role in the robotic arm by providing accurate current measurements. For starters, the torque produced by a motor is directly proportional to the current flowing through it. By using shunt resistors to measure the current, the control system can accurately determine and adjust the motor torque. An accurate current measurement would ensure precise control of the motor's speed and position, which is essential for tasks requiring high precision, such as assembly and machining. Moreover, Shunt resistors help detect overcurrent conditions that may indicate a mechanical jam or excessive load. The control system can then take protective actions, such as shutting down the motor or reducing its speed, to prevent damage. Continuous current monitoring would allow for early detection of faults and anomalies, enabling predictive maintenance and reducing downtime.

Shunt resistors would also assist in Joint Force Sensing. By measuring the current through the motor at each joint, the control system can estimate the force being applied. This is crucial for applications where the robot arm interacts with delicate objects or requires precise force application. Real-time current data allows the control system to adapt to varying loads and operational conditions, improving the efficiency and accuracy of the robot arm.

Finally, Shunt resistors would help with the Power Management of the robot arm.By measuring the current drawn by each motor, the system can monitor and optimize the overall power consumption of the robot arm. For battery-operated robotic systems, accurate current measurement helps in efficient battery usage and management, ensuring longer operational times. Moreover, Continuous current monitoring helps in managing the thermal profile of the motors and other components, preventing overheating and ensuring reliable operation.

3.5.2-3 Rogowski Coil:

Another option for a current sensor would be a Rogowski coil. Rogowski coils are a type of electrical device designed to measure alternating current (AC) or high-speed current pulses.They were developed to address limitations in conventional current transformers, offering a non-invasive and highly accurate method for current measurement. The primary component of a Rogowski coil is the helical winding of a conductive wire, typically made from copper due to its excellent electrical conductivity and low resistance. The number of turns in the coil and the spacing between them are carefully designed to ensure that the coil generates an accurate voltage signal proportional to the rate of change of the current. Another critical component of the Rogowski coil is the type of core it is implemented on. Unlike traditional current transformers that use a magnetic core, Rogowski coils are wound on a non-magnetic core, usually made from flexible materials such as plastic or rubber. This design prevents magnetic saturation and allows the coil to handle a wide range of current levels, making it ideal for high-speed and high-frequency applications. Moreover, the flexibility of the non-magnetic core allows Rogowski coils to be easily wrapped around conductors of various shapes and sizes, enhancing their versatility in diverse applications.

The shielding component of the Rogowski coil is crucial for ensuring accurate performance. To minimize interference from external electromagnetic fields, Rogowski coils often incorporate shielding, which historically proved essential in applications requiring high precision. This shielding typically consists of an additional conductive layer surrounding the winding, acting as a barrier against electromagnetic noise. The shielding layer is usually grounded to further reduce interference, ensuring precise current measurement even in electrically noisy environments. Furthermore, the entire winding and core assembly is encapsulated in a durable protective covering, typically made from insulating materials such as rubber or high-strength polymers. This encapsulation protects the coil from environmental factors like moisture, dust, and mechanical damage, which, if left unchecked, could compromise its performance.

Lastly, because the output of a Rogowski coil is proportional to the rate of change of current ($\frac{di}{dt}$), an integrator is required to convert this signal into a voltage that is proportional to the actual current. Historically, analog integrators were commonly used, employing operational amplifiers and passive components such as resistors and capacitors to perform the integration. In modern systems, integrators can be either analog or digital. A digital integrator typically uses microcontrollers or digital signal processors (DSPs) to process the signal with greater precision and flexibility. These digital systems often include calibration settings to adjust gain and offset, ensuring that the output accurately reflects the measured current. The inclusion of integrators is critical to the functionality of Rogowski coils, as they enable the conversion of dynamic signals into actionable data. Properly calibrated integrators allow these coils to maintain their reputation as reliable tools for high-precision current measurement across a wide range of applications, from industrial power monitoring to advanced robotics and renewable energy systems.

There are several notable advantages to using Rogowski coils. Firstly, Rogowski coils do not require breaking the circuit or making direct contact with the conductor, which reduces the risk of electrical hazards and simplifies installation. Their flexible design allows them to be easily wrapped around conductors of various shapes and sizes without requiring disassembly, making them particularly valuable in applications where space is limited or access is challenging. Moreover, Rogowski coils can measure a wide range of current levels, from just a few amperes to several kiloamperes, without experiencing saturation. This feature sets them apart from many other current sensing devices, which often struggle with high-current measurements. Additionally, they are capable of accurately detecting low currents, making them versatile across a broad spectrum of applications. Another key advantage of Rogowski coils is their wide bandwidth, which enables them to capture fast transient events and high-frequency signals. This capability makes them well-suited for dynamic and rapidly changing current measurements, such as those found in power electronics, industrial equipment, and renewable energy systems. By combining safety, flexibility, and accuracy, Rogowski coils remain an essential tool in both modern and legacy systems, with the potential to play an even greater role in future technological advancements.

Another significant advantage of Rogowski coils is that the absence of a magnetic core eliminates the risk of saturation, a common issue with traditional current transformers. This feature ensures linearity and accuracy even at very high current levels, making them highly reliable in demanding applications. The lightweight and flexible design of Rogowski coils also makes them easy to install in confined spaces and around irregularly shaped conductors. This design feature was especially beneficial in industrial environments where accessibility is limited. The simplicity of the design without a magnetic core also makes Rogowski coils more cost-effective to produce compared to traditional current transformers, especially for high-current applications. Moreover, for safety purposes, the coils produce a low voltage output, significantly reducing the risk of electric shock during installation and measurement. This characteristic not only enhances safety but also simplifies handling procedures for operators. Additionally, the materials used in Rogowski coils are typically stable across a wide range of temperatures, ensuring consistent performance in various environmental conditions.

However, using Rogowski coils also presents some limitations. The voltage output from a Rogowski coil is proportional to the derivative of the current ($\frac{di}{dt}$), which necessitates the use of an integrator to obtain the actual current measurement. This requirement adds complexity to the system and demands precise calibration for accurate results. Historically, achieving such precision posed challenges, especially in environments where signal integrity was critical.The accuracy of the current measurement largely depends on the quality of the integrator and its calibration. Over time, the calibration of the integrator may drift due to environmental factors or wear, necessitating periodic recalibration to maintain measurement accuracy. Moreover, Rogowski coils are primarily designed for AC current measurement and are not suitable for measuring steady-state DC currents. While they excel in detecting high-frequency components and transient events, they cannot measure the DC component of the current directly. This limitation could restrict their use in applications requiring continuous DC current monitoring. However, for scenarios involving high-frequency AC signals or rapidly changing currents, Rogowski coils remain an invaluable tool, with the potential to become even more versatile as future technologies address these limitations.

In a robotic arm, each joint is typically driven by an electric motor. Accurate current measurement is critical for precise control of these motors, as the current is directly proportional to the torque produced by the motor. Rogowski coils can measure the current supplied to each motor, enabling real-time adjustments to optimize performance. The data from Rogowski coils is often utilized in feedback loops to maintain desired motion profiles, ensuring that the robotic arm moves smoothly and accurately along its programmed trajectory. This is particularly important in tasks requiring high precision, such as medical robotics or assembly line automation. Another significant application of Rogowski coils is fault detection and protection. They can detect overcurrent conditions, which may indicate issues such as mechanical jams, excessive loads, or electrical faults. Continuous current monitoring also provides insights into the health of the robotic arm's components. For example, an increase in current might suggest that a motor is working harder than usual due to increased friction or other mechanical issues. This early warning system allows maintenance to be performed proactively, preventing costly failures or downtime.

Energy efficiency is another crucial application of Rogowski coils. These coils enable the precise measurement of power consumption for each motor in a robotic arm. This data can be analyzed to optimize the energy usage of the robotic arm, ensuring efficient operation and reducing overall energy costs. By analyzing current data, the control system can distribute loads more evenly across all motors, which enhances both the efficiency and lifespan of the robotic arm. This load balancing is particularly beneficial in preventing localized wear and overheating, ensuring consistent performance over time. Additionally, electric motors generate heat as a byproduct of their current draw, which is directly proportional to the load they carry. Rogowski coils provide real-time current data, allowing the control system to monitor and manage the thermal load on each motor effectively. This proactive thermal management helps prevent overheating, reducing the risk of motor failure and ensuring reliable operation.

3.5.2-4 Current Transformers:

Another option for a current sensor for the robotic arm is the Current Transformer (CT). Current Transformers (CTs) are devices used to measure alternating current (AC). They function by producing a reduced current proportional to the current in the circuit, which can then be safely monitored and measured by meters and control systems. Historically, CTs have been widely used in power systems for their accuracy and ability to isolate measurement devices from high-voltage circuits.

CTs consist of several essential components, one of which is the primary winding. The primary winding function is to carry the current that needs to be measured. It usually consists of a single turn, such as a cable or busbar, passing through the CT. The primary winding is connected in series with the load, so the current flowing through it is the same as the load current. Another crucial component of the CT is the secondary winding, where the measurement current is induced. The secondary winding is typically composed of many turns of wire wound around the core. The number of turns in the secondary winding determines the transformation ratio. For example, a CT with one primary turn and 100 secondary turns will produce a secondary current that is 1/100th of the primary current. This transformation enables accurate and safe monitoring of high currents in the primary circuit. The final critical component of the CT is the magnetic core. The core concentrates the magnetic field generated by the primary current and provides a path for the magnetic flux to induce a current in the secondary winding. The material and design of the core are vital for the efficiency and accuracy of the CT.

It is important to highlight the numerous advantages of current transformers (CTs).One of the most significant advantages of CTs is their ability to provide electrical isolation between the high-voltage primary circuit and the low-voltage secondary circuit. This ensures the safety of personnel and equipment by allowing current measurements to be taken without direct exposure to hazardous high voltages. Moreover, CTs offer exceptionally high accuracy in current measurement, making them critical for precise monitoring and control in various applications, such as industrial automation and power distribution systems. CTs can also be calibrated to ensure consistent and reliable performance over time, further enhancing their long-term utility. CTs also make it safer to measure and monitor high currents by reducing them to lower, manageable levels, allowing integration with standard monitoring devices. Furthermore, they play a vital role in monitoring power quality parameters, which is essential for energy management and efficiency improvements. Accurate current measurements provided by CTs enable operators to identify imbalances or inefficiencies in power distribution systems. Additionally, CTs assist in effective load management, ensuring that power is distributed evenly and efficiently across all connected systems. By combining safety, accuracy, and versatility, CTs have been indispensable in past industrial and electrical applications.

Current transformers (CTs), despite their many advantages, have several limitations that can impact their accuracy and reliability. One significant issue is unnecessary accuracy variations, which can occur if CTs exhibit nonlinear behavior under low current conditions or when there are changes in ambient temperature. Furthermore, if the core material becomes saturated due to excessively high primary current or exposure to external magnetic fields, the CT can no longer produce an accurate proportional current, leading to increased measurement errors. Another limitation is that CTs can produce erroneous readings when operating outside their specified range. This can affect system reliability, particularly in dynamic applications. Additionally, CTs may introduce a phase shift between the primary and secondary currents, impacting power measurement accuracy. This issue is particularly problematic in power factor correction and energy metering applications, where precision is crucial. Finally, standard CTs are designed for AC current measurement and cannot directly measure direct current (DC) without additional components or specialized designs. This limitation restricts their use in systems requiring DC measurement, such as solar power installations or electric vehicles. If future innovations address these constraints, CTs could become even more versatile and widely applicable across various industries.

Current transformers (CTs) offer several benefits when applied to a robotic arm. One key application is motor control and monitoring. CTs measure the current supplied to the electric motors driving the robotic arm, providing accurate current measurements that are critical for controlling the motor's speed, torque, and position. The current data provided by CTs is fed into the control system, which uses it to adjust motor inputs dynamically. This ensures the precise and smooth operation of the robotic arm, enabling it to follow programmed trajectories with high accuracy. Additionally, overload and fault protection are other crucial applications. CTs can detect overcurrent conditions that might damage motors or other components. When an overcurrent is detected, the control system can take protective actions such as reducing power, shutting down the motor, or triggering an alarm. Moreover, CTs can identify sudden current changes that indicate faults such as short circuits or mechanical jams. Their real-time monitoring capabilities allow the system to quickly detect and address these issues, minimizing downtime and ensuring operational efficiency. Lastly, CTs play a critical role in energy management by monitoring the robotic arm's power consumption. This data is essential for optimizing energy usage and improving overall efficiency. CTs also assist in managing the electrical load by providing accurate current measurements, enabling the system to distribute power efficiently among different components of the robotic arm. These capabilities, combined with potential future advancements, make CTs an invaluable tool for robotics and automation.

3.5.2-5 Current Sensor Technological Choice:

Selecting the most suitable current sensor for the robotic arm involved an extensive comparison of various characteristics. After careful analysis, it was concluded that the Hall Effect Current Sensor was the optimal choice. This decision was based on its ability to provide essential electrical isolation, measure both AC and DC currents, and offer a fast response time, which would be crucial for dynamic measurements in robotic applications. If other sensors had been selected, they might not have met the project's requirements as effectively.

Additionally, the non-invasive nature of Hall Effect sensors eliminated the need for direct electrical contact, reducing the risk of electrical hazards and ensuring system safety. Their robustness and ability to perform reliably in diverse conditions made them ideal for integration into the robotic arm. Had less durable or less precise sensors been chosen, the overall performance and safety of the robotic system could have been compromised.

The combination of their versatility, fast response, and reliability ensured that Hall Effect sensors were not only the best choice for this application but also provided a reliable foundation for the arm's operation. Looking ahead, their consistent performance and adaptability could continue to make them indispensable in future robotic advancements.

Table 3.5.2-5 - Current Sensors Technological Comparison Table

Feature	**Hall Effect (Chosen)**	**Shunt Resistors**	**Rogowski Coil**	**Current Transformer**

Output type	Analog/Digital	Analog (Voltage)	Analog (Voltage)	Analog (Voltage or Current)
Range	Wide	Narrow to Medium	Wide	Medium to High
Accuracy	Moderate to High	High	Moderate	High
Linearity	High	High	Moderate	High
Bandwidth	Low to Moderate	High	High	Moderate
Installation Complexity	Low to Moderate	Low	Moderate to High	Moderate to High
Response Time	Fast	Very Fast	Fast	Moderate
Power Consumption	Low to Moderate	Low	Very Low	Low
Cost	Moderate to High	Low	High	Moderate to High

3.5.2-6 Current Sensor Price Selection Choice:

After selecting the type of current sensor to use, the next step was to choose a specific brand that met the application requirements. Key characteristics considered during the decision-making process included the current range, accuracy, response time, and cost. Based on these criteria, the selection was narrowed down to three brands: Allegro ACS712, Texas Instruments TMCS1101, and LEM LA 25-NP. After thoroughly comparing and contrasting the features, the Allegro ACS712 was selected as the optimal choice. This decision was made due to its affordable price, competitive performance, and ability to measure current up to 5A, which was sufficient for the robot arm's needs. The ACS712 also offered an accuracy of 185 mV/A and a response time of 5 µs, which ensured reliable and real-time current measurements for the robotic arm's operations. Its surface-mount form factor made it easy to integrate into the system, further enhancing its practicality. If a faster response time or higher current range had been critical, the LEM LA 25-NP might have been chosen, but for this application, the ACS712 struck the ideal balance between performance and cost-effectiveness. Ultimately, the Allegro ACS712 was selected not just for its affordability but also for its compatibility with the robot arm's current range and operational requirements. This choice ensured efficient performance while maintaining the overall budget, making it the best option for this application.

Table 3.5.2-6 - Current Sensor Price Comparison

Feature	Allegro ACS712 (Chosen)	Texas Instruments TMCS1101	LEM LA 25-NP
Current Range	≤ 5 A	≤ 20 A	≤ 25 A
Accuracy	185mV/A	±1.5%	±0.5%
Response Time	5µs	6.5µs	1µs
Form Factor	Surface Mount	Surface Mount	Through-hole
Cost	$5.69	$6.59	$34.21
Seller	DigiKey	DigiKey	DigiKey

3.6 End Effectors:

The end effector is one of the defining features of a robotic arm and can determine the available practical applications of the arm itself. Each kind of end effector comes with its own litany of strengths and limitations as applied to our robot. End effectors are specialized parts of a manipulator type robot that allows it to interact with its environment in ways such as picking up or moving objects. Some examples of end effectors include claw grippers, suction cups, and magnets. Each end effector has its own implications for how the robot has to be designed and will change the overall capabilities of the robot. For this it's important to match our end effector with the intended behavior of the robot. We intend for the robot to be able to pick up an object and have it be able to determine whether or not it is holding an appropriate target based on feedback given by the current sensors. For this there are a few traits that we need to target in order to select an appropriate end effector.

The first criteria is that the hold the robot has on an object has to be fairly consistent across multiple trials in holding the same or similar objects. This kind of predictability is helpful in programming our intended behavior because it leads to less variation in the current drawn by the motors. Less variation in each trial lets us better create an expected range of currents that are deemed an “appropriate” object.

3.6.1 Vacuum Gripper:

One of the end effectors we considered is a vacuum end effector. This would include powered suction cups and similar hardwares. This would be best used for purposes such as picking up objects with broad, flat surfaces and smooth textures. A potential application of this would be something like a robot in a car factory handling car windows. A potential advantage is that because the robot isn’t gripping the object with a claw there’s less potential for an item to be crushed by the gripper. They can also quickly pick up and release objects by powering the motor. This allows certain objects to be picked up relatively simply in comparison to some other styles like the parallel or angular gripper. Vacuum grippers also can utilize the power of the motor, and the friction coefficient of the flange in contact with the surface of the object. A drawback is that since the suction power is limited, these types of end effectors are mostly limited to lightweight objects. They also are poor at handling objects with porous or uneven surfaces further limiting the kind of objects that can be handled by this type of manipulator.

3.6.2 Parallel Gripper:

A parallel gripper uses 2 or more powered grippers to hold and manipulate objects. This is in contrast to angular grippers in which the fingers move together apart on a more centralized joint on an angle. Parallel grippers are fairly simple in control and execution with a wide variety of styles to choose from so availability is less of an issue. These grippers are also broadly effective with a wide variety of object shapes.

One of the main advantages compared to other kinds of claw grippers is predictability with how the claw will interact with objects while still being open enough to accommodate a broad variety of shapes. However, very small objects or ones with more irregular shapes could be a potential weakness of this design. Flat objects for example are difficult for a parallel gripper to pick up unless the object is positioned upright. The flexibility is also quite limited, but a parallel gripper remains a good option among gripper style end effectors.

3.6.3 Angular Gripper:

Angular grippers operate utilizing jaws that open in an angular motion around a central pivot point. This can involve 2 or more jaws to create different kinds of grips. A larger number of fingers in the jaw also affects the way the manipulator can handle objects. A larger number of fingers favors smaller objects for example, but a two finger design is applicable in a wide variety of use cases as they are decent at handling objects that fit between the angle of their jaws. A few drawbacks of this design are that the orientation of the jaws affects the effectiveness of the grip on the object. Also the length of the jaws also has an impact on the force exerted by the gripper so the design of this manipulator can be deceptively simple, introducing unexpected issues.

3.6.4 Electromagnetic Gripper:

Electromagnetic grippers utilize magnetic fields generated by electromagnets to attract ferromagnetic materials and handle them. The electromagnetic field is controlled by the electromagnet so by powering and unpowering it, the attractive force can be turned on or off. This allows objects to be picked up or dropped quickly and simply. Because it relies on an attractive force and not a physical connection reliant on friction or orientation, a magnetic gripper can very easily pick up compatible objects regardless of how they are positioned. Since the gripper relies on magnetic attraction the types of objects that can be handled by the manipulator are limited to magnetic materials. Also the environments in which it can be operated are limited too as magnetic materials in the robot's environment could interfere with the operation of the manipulator. The electromagnet also requires constant power to remain active so it will be drawing power as long as it is holding an object would add up over time.

3.6.5 Gripper Comparison:

After comparing the advantages and limitations of the different styles of grippers the parallel gripper appears superior to the angular gripper for this application due to its predictability. The vacuum gripper is also advantageous compared to the electromagnetic gripper due to the greater diversity of objects that can be handled. The vacuum and electromagnetic grippers are more comparable due to their operation being an on and off procedure rather than a hold with a jaw unike the angular and parallel grippers.

With that, the top two styles of grippers in consideration are the vacuum and parallel gripper. Ultimately we elected to utilize the parallel gripper as it can be used in a much wider variety of applications. Since the vacuum gripper is still an attractive technology it has been included in the hardware comparison table alongside parallel grippers in consideration. Ultimately, we decided to pick the LG-NS Robot Gripper with its servo included since it is the simplest to implement and we don't need a rotating wrist.

Table 3.6.5-1 - Gripper Type Comparison Table

Feature	**Vacuum Gripper**	**Parallel Gripper**	**Angular Gripper**	**Electromagnetic Gripper**
Strengths	Best used with flat and smooth objects. Can pick up and release objects quickly.	Predictable grip and interaction with objects.	Can accommodate a greater range of sizes than a parallel gripper.	Can handle objects of appropriate weight regardless of shape. Can pick up and release objects quickly.
Weaknesses	Bad at handling objects with rough textures and/or irregular shapes. Needs to be powered the entire time it is handling an object.	Struggles with irregular shapes.	Less surface area in contact with the object leading to less stability.	Can only be used to handle magnetic materials. Will attract any magnetic object nearby.
Applications	Can be used to handle flat, smooth, and delicate objects that are otherwise difficult for other kinds of grippers to handle such as glass panes or	Can be used to handle objects with a diversity of shapes, textures and materials.	Can be used to pick up small objects that benefit from a pinch style grip. Multiple jaws can improve grip in exchange for ability to	Can be used to handle magnetic objects with a diversity of shapes and textures

	tiles.		handle larger objects.	
Complexity	High complexity due to the need for air supply and vacuum generation systems.	Low to moderate, just requires extra servo	Low to moderate, just requires extra servo	Moderate to high, will involve an electromagnet and power supply.

Table 3.6.5-2 - Hardware Comparison Table

Feature	**Robot Suction Cup Vacuum Pump Kit For 25T Servo MG996 MG995 DS3218**	**LG-NS Robot Gripper with Servo (Chosen)**	**Robot Gripper**
Short Description	A kit of parts coming with a suction cup, vacuum pump kit, and switch. Compatible with 25T servos	A simple parallel gripper that can be operated by a single servo.	A parallel gripper operated with 2 servos. 1 for opening the jaw, the other for rotating the wrist.
Other Specs	N/A	Max clamp width: 1.3" Size: 57x65x30 mm(2.24"x2.56"x1.18") Weight: 25g	Clamp range:0-54mm (0-2.13") Size:105x100x30mm (4.13x3.94x1.18")
Applications	Can be used in applications such as simple pick up put down behavior but is limited in the types of objects that can be handled.	A simply operated gripper with a fairly small opening but is operated simply.	A parallel gripper capable of rotating the claw to grip at more angles with a larger opening than the LG-NS. Only slightly more

			expensive
Price	$22.11	$34.99 (With servo included)	$19.50
Supplier	Ebay	DFROBOT	DFROBOT

3.7 Microcontroller:

The microcontroller chosen for this project is the ESP32-Wroom-32. These microcontrollers were selected due to their dual-core processing, flashing memory, flashing speed, as well as their compatibility with our chosen communication channel. The ESP32 is an Espressif device compatible with the Arduino IDE. This device is beginner-friendly, and compatible with most peripheral devices, making it a fantastic choice for our project, which requires many peripheral devices.

The ESP32 comes equipped with 28 GPIO pins, 5v and 3.3v OUT pins, several SPI pins, and RX/TX connections. With these connections we are able to power our communication bus (CAN) PCBs, joystick PCB, current sensors, and encoders. The ESP32 operates at a clock frequency of up to 240 MHz, and not only offers compatibility with SPI but CAN connections as well, a feature that very few other microcontrollers offer. Its processing speed provides the necessary computational power for real-time control tasks. This high clock frequency ensures that the microcontroller can manage the precise timing and synchronization required for motor control, data parsing, and feedback processing. The ESP32's dual-core Tensilica LX6 processors offer a balance of performance and efficiency, making it well-suited for handling the complex control algorithms and real-time data processing needed in this robotic arm project. Its ample GPIO pins and multiple I2C, SPI, and UART interfaces also facilitate easy integration with various sensors, motor drivers, and communication modules.

While we did not go with a wireless connection for our robotic arm, we did have the option using the ESP32s, which are capable of bluetooth and wireless connection. Due to time constraints and overall purpose of the project, we did not use these features.

Even with all of these powerful features, the ESP32 offers low-power modes that help optimize energy use, which is beneficial for battery-powered systems, or systems that may need some serious power management. This feature is great to keep in mind when working with our entire power distribution system.

These modules are also cost efficient and easy to buy in bulk, making the replacement of parts and the possibility of cheap backups possible. Each development board is easily purchased, which can be stripped for parts for all needed components at a much cheaper rate than simply buying the MCU.

In addition to the ESP32, several other microcontrollers could be considered for this project, each with its own set of features and benefits. The STM32F4 series, for example, offers high performance with its ARM Cortex-M4 core, which includes a Digital Signal Processing (DSP) unit and a Floating Point Unit (FPU). This makes it ideal for complex control algorithms and real-time processing. Operating at a clock speed of up to 168 MHz, the STM32F4 series provides extensive memory options and multiple communication interfaces, making it capable of handling detailed kinematic calculations and motor control tasks. Its multiple communication interfaces, including I2C, SPI, UART, CAN, and USB OTG, provide flexibility in connecting to various sensors and actuators.

Another strong candidate is the RP2040, a powerful and versatile microcontroller with dual-core processing, high clock speed, and extensive memory. Operating at up to 133 MHz, the RP2040 is particularly suitable for simple SPI applications and sensor-related operations.The wide range of peripherals and interfaces supported by the RP2040, including multiple I2C, SPI, and UART interfaces, support complex control tasks and allow for flexible sensor and actuator integration, making it an excellent choice for projects requiring quick connections and multiple peripherals that pass large amounts of data.

The Atmel SAM D21 offers a good balance of performance, power efficiency, and ease of use with its ARM Cortex-M0+ core, operating at up to 48 MHz. While not as powerful as some alternatives, it is sufficient for many control tasks and offers lower power consumption, which can be beneficial for battery-operated systems. The SAM D21's comprehensive set of interfaces and peripherals, including multiple I2C, SPI, UART, USB, and CAN, combined with its affordability and support within the Arduino ecosystem, make it a strong candidate for projects requiring efficient and straightforward microcontroller solutions.

The Texas Instruments MSP430 series is known for its ultra-low power consumption, making it ideal for applications where energy efficiency is critical. Operating at up to 25 MHz, its 16-bit RISC architecture provides sufficient processing power for many control applications. The MSP430's extensive peripheral set, including multiple I2C, SPI, UART, and USB interfaces, and low power modes make it a suitable choice for portable or battery-powered robotic systems.

Lastly, the Arduino Due, based on the Atmel SAM3X8E, offers a good mix of performance and ease of use. Operating at 84 MHz, its ARM Cortex-M3 core provides adequate processing power for control tasks, and its memory is sufficient for handling the necessary computations and data storage. The Arduino Due's compatibility with the extensive Arduino ecosystem and libraries simplifies development and prototyping, making it a user-friendly option for rapid development and testing.

Table 3.7 - Microcontroller Comparison Table

Microcontroller	Clock Speed	Cores	Memory	Interfaces
RP2040	133 MHz	Dual-core Arm Cortex-M0+	2 MB Flash, 264 KB SRAM	Multiple I2C, SPI, UART, PIO
STM32F4 Series	168 MHz	ARM Cortex-M4	1 MB Flash, 192 KB RAM	Multiple I2C, SPI, UART, CAN, USB OTG
ESP32 (Chosen)	240 MHz	Dual-core Tensilica LX6	520 KB SRAM 16 MB Flash	Wi-Fi, Bluetooth, multiple I2C, SPI, UART, CAN
Atmel SAM D21	48 MHz	ARM Cortex-M0+	256 KB Flash, 32 KB SRAM	Multiple I2C, SPI, UART, USB, CAN
MSP430	25 MHz	16-bit RISC	256 KB Flash, 16 KB SRAM	Multiple I2C, SPI, UART
Arduino Due	84 MHz	ARM Cortex-M3	512 KB Flash, 96 KB SRAM	Multiple I2C, SPI, UART, USB OTG, CAN

3.8 Communication Channels:

For this project, we require accurate real-time information that is transmitted via the hardware and received by our software. Our software will then take this information to simulate movement using a ROS environment and PyBullet simulator. This simulation data is then transmitted back to our microcontroller hardware in real-time, providing us with real-time movement adjustments. Our microcontrollers are responsible for driving our motors, and passing the appropriate data to our Raspberry Pi 5. To successfully transmit data between these two mediums, we required an embedded system to help both components send messages bi-directionally. This is done successfully by using one of two communication channels, a choice which dictated how fast, reliable, and error-free our data was transferred.

Computer buses are the basis for transferring real-time data between our embedded hardware and our software components. These buses transmit data over parallel wired connections or by sending a singular bit of data at a time over a serial connection. Industry standard often dictates that serial connections be used over longer distances, since these types of connections tend to maintain data integrity far better than parallel connections, which send data as a whole, all at once. While choosing between these two types of connections, we considered the fact that parallel connections are often quite costly and lack proper data synchronization when transmitting, even over shorter distances. These “shorter” distances are not much longer than the inside of a RAM stick, which greatly reduces the area we would have been able to work with, while still requiring a large amount of wiring when compared to a serial connection. In recent decades, serial connections have been replacing parallel ones, even for smaller spaces, due to these weaknesses.

Accuracy, reliability, and affordability were all absolute musts when we were choosing components for this project, as all three of these were the major pillars of our overall objectives and goals. Therefore, choosing the serial communication method was far more beneficial to us than a parallel one.

3.8.1 Serial Communication:

The type of communication channel we used was required to be reliable, resistant to electromagnetic interference, and compatible with our ESP32 microcontrollers. Serial communications send a single stream of data over a cable, which reliably and synchronously send singular bits at a time between two locations. While parallel connections send far more bits per second than a serial connection, it is possible to increase the data rate by increasing the clock cycle without sacrificing the integrity of the transmitted bitstreams. These types of connections also consume less space overall. They require considerably less wiring than parallel connections, often using a twisted-pair cable structure that requires only two interconnected wires to transfer data over.

While the quantity of wires required to create a serial communication bus is far less than a parallel one, the cables used for transferring data long-distances are often far thicker. Examples of such connections include wired USB computer mouses and HDMI cable connections. The specific connection we used connects two integrated circuits within the same housing, so the thickness of our wiring was accounted for when designing the architecture of our project.

Choosing a serial connection means that we were then able to choose from a plethora of communication buses, which were to be the circuit-to-computer translator during any movement from our robotic arm. There are several different types of wired serial buses that we were able to choose from, including an Inter-Integrated Circuit (I2C) bus, Serial Peripheral Interface (SPI) bus, Universal Asynchronous Receiver/Transmitter (UART) bus, and a Controller Area Network (CAN) bus.

Since there are many unique and useful options to choose from in this particular category of components, we narrowed down our choices of serial buses by looking at our design constraints. Our project constraints required that we look at how compatible the communication protocols are with our specific microcontrollers, as many buses require a special communication device to transmit signal data depending on how the bus processes it. Sorting them by their protocols, we found that every bus, excluding the CAN bus, sends signal data digitally. CAN buses are the only bus-type from this list that send signal data differentially. The differences between the two can be summarized by how they handle bits, including exactly how data is transferred and recovered. Digital signal transfer works by using a line of wires for separate purposes. The first wire transfers bits, while the second wire provides a clock signal, designed to synchronize each bit sent. There is also a "selection" line which determines the direction of travel for the bits, setting the flow of data-conversation between each device. Differential signal lines, however, only require a twisted pair of wires that send data equally. These wires also have the same ground potential without the need for an actual common ground line to be implemented.

The physical differences between these data protocols is immediately apparent, as the digital signal protocols require many more wire harnesses and connection points than a differential protocol. This can drive up the overall cost between components, as well as affect how quickly devices can send messages between each other.

Table 3.8.1 - Communication Channel Comparison Table

Feature	Serial Communication (Chosen)	Parallel Communication
Data Rate	Low Data Rate, Single-bit data transfer	High Data Rate, Multiple-bit data transfer
Data Synchronicity	Reliable, Sends data linearly	Unreliable, Sends data nonlinearly
Distance	Capable of very large distances (Fiber optic cables, Phone lines)	Capable of shorter distances (Microprocessor connections)
Cost	Cost-efficient, only one wire	Not cost-efficient, multiple wires needed
Bandwidth	Support for high bandwidth signals	Support for low bandwidth signals

3.8.2 Bus Protocols:

Communication protocols within serial and parallel channels dictate the format of messaging between two devices. These protocols define how data is transferred, how devices speak to each other, and determine how to recover corrupted bits.

The choice between a digital or a differential protocol is very dependent on the computing power of the technology that is being developed. Many microcontrollers support digital transfer over a differential one, and require an extra component for transferring data if you wish to use a differential signal protocol. However, cost alone is not the determining factor of which protocol to use. This choice may be more cost-effective in the long run, but it will also result in a higher rate of error over time. This error will cause more collisions and injury as readings become increasingly inaccurate due to a high sensitivity to electromagnetic interference.

If the designed device does not experience a high amount of electromagnetic noise, it may be useful to consider using a digital protocol, since these protocols are overwhelmingly compatible with various electrical components and microcontrollers, but do not handle electromagnetic noise as well as differential protocols do. This is due to the high complexity of the wiring in buses that utilize digital protocols. This complexity often comes from the requirement that a “master” device be used, which drives all “slave” devices, to send data over a clock signal. This clock wire is the digital bridge between analog devices and is sensitive to interference and noise that can modulate its waveforms. This modulation can hinder synchronicity between devices, causing slower or incorrect rates of bit transferal.

The communication protocol we used dictated how our data was synchronized, defined, and recovered when transferred. Error recovery was one of the most important portions of choosing our protocol. Safety and response time were large priorities of our project, which required a very small margin of error. Handling errors within our bitstreams was a crucial step, and there is a very large difference between the methods that each protocol uses to handle error-causing interference and noise.

Among protocols that use digital signaling are I2C, SPI, and UART. These protocols all use the Master-Slave device communication method. While I2C allows for multiple masters, it still depends on these master devices driving slave devices. I2C communication protocols are often used in the Internet of Things, are simple to implement, and compatible with various devices. SPI is very similar, however it does not use multiple masters. Both use a clock signal for synchronous data transfer, however SPI does not have built-in error handling like the I2C protocol does. UART does not have this synchronous clock signal, and instead uses bits to mark the data sent. This particular method is much older than the other two, and works similarly to a CAN bus, although much less reliably.

All of the digital signaling methods have their unique advantages, however none work as well as the CAN bus when it comes to automation. The CAN bus supports a level of complexity and communication between devices that the digital protocols are not able to achieve, simply because of their sensitivity to noise and method of data transfer.

Table 3.8.2 - Bus Comparison Table

Feature	CAN Bus (Chosen)	I2C Bus	SPI Bus
Data Rate	Up to 1 Mbps	Up to 5 Mbps	Up to 65 Mbps
(Built-in) Error Handling	Bit Stuffing, Error Flags, ACK/CRC	ACK/NACK	None
Protocol	Differential Voltage, CAN 2.0A, CAN 2.0B	Variable Voltage Levels	Variable Voltage Levels
Clocking	Asynchronous	Synchronous Master CLK	Synchronous Master CLK
Use Cases	Automation (Automotive and Industrial)	LED Displays, Sensors	LCD, Sensors, Memory Devices

3.8.3 CAN Buses:

Controller Area Network (CAN) buses were developed in the 1980s and were designed specifically for automotive electronics. Today, these buses are used in both the automotive industry, as well as most industrial automation. Automotive and automated devices require a plethora of separate nodes and devices that are all interconnected, yet no nodes should be the main driver, i.e. no one node should be more powerful than another. This reasoning is why CAN buses work as a decentralized framework, ensuring all nodes attached to the bus channels are equal.

While many buses make use of the master-slave method of organizing devices, in CAN buses there are only masters. This means that while the CAN bus is capable of master-slave device control, it is more likely to be used between all devices equally. Compared to other buses, the data rate of a CAN bus is much slower, however they have much more advanced and reliable built-in techniques for error handling. Our project contained only two major devices that our CAN bus will send data to and from, eliminating the issue of speed in most instances. Since data is broadcasted over a CAN bus whenever the line is deemed free (no data is being transferred). This bus has the capability of talking to all nodes at once, allowing each device to accept or ignore any data transferred over. Were we to consider each motor a device, then this bus would be able to communicate to each one simultaneously.

The data that a CAN bus transmits is carried in "frames". This means that all data is sent over as a block of data, with extra bits inserted for error checking and synchronization purposes. The handling of errors is done through processes such as bit-stuffing, flags, and ACK/CRC, all of which are methods that use extra bits to verify data integrity. During bit-stuffing, a certain amount of 0's or 1's are placed throughout the transmitted data, in a patterned and known way, to properly check if the sent data and the expected data match. If the system detects a mismatch, then an error has been found and the system will resend the frame that contains the corrupted bits to correct the error. These extra bits allow other devices to recognize the data being sent, which will trigger them to accept or ignore a given frame. Each frame has a set amount of bits at the beginning and end which sandwich the real data being sent. These bits are the identification bits and any error flags that have been inserted during the error handling steps. With these methods, the possibility of error correction is greatly increased, making for a much more reliable system overall. This reliability greatly reduces any risk of injury when dealing with real-time environments, and in the case of our project, will allow our arm to operate mostly error-free. Most bus-types only provide an "acknowledgement check" (ACK/NACK) built into the architecture, which detects the dominant bit sent over the transmission wires, if they provide any built-in error detection at all. There will always be some amount of error that we cannot prevent, however by using a CAN bus architecture, we greatly reduce the risk of electromagnetic noise errors, as well as software malfunctions.

The space that a CAN bus takes up is minimal and designed to be efficient. There are only two wires required, one for CANH (CAN-High voltage) and one for CANL (CAN-Low voltage). These two lines are complimentary, meaning that they both receive equal and opposite current. While there are certain bus-types that have a far faster data rate, these buses are not compatible with the type of system we wished to build. Additionally, the speeds they provide were very likely not to make a large impact on our project's overall functionality. In real practice, they did not, and performed wonderfully in real time with all of our peripherals attached. The rate of the ISO 11898-2 standard (high-speed CAN bus) is 1 Mbps, which is more than enough for a real-time environment. Even if this data rate were to slow message transfer for our robotic arm down a noticeable amount, the error handling and various protocols and capabilities of a CAN bus far exceed any other bus type and were all necessary to meet the safety and response goals of our project.

There was a large pool of options while we were choosing CAN bus types, including separate protocols, which affect the number of bits sent per frame. The protocols within a CAN bus range from the Standard protocol, which uses an 11-bit identifier, to CAN 2.0B (Extended CAN), which uses a 29-bit identifier. This longer identifier allows the CAN bus devices to exert finer control over messages sent, but greatly reduces the compatibility between nodes if they do not use the exact same protocols. A good workaround for this issue is the CAN FD, which allows for a flexible data-rate, data field-size, and greater compatibility.

While the CAN bus is not natively supported by the Raspberry Pi, there are additions we can add onto our microcontrollers and microprocessor to make it compatible with the hardware. Among these options include rewiring arduino connections and HAT integration.

The PICAN-FD is a CAN FD bus compatible with the Raspberry Pi that also includes a real-time clock. This component has a data rate of up to 8Mbps, 120 Ohm terminator, Python compatibility, and battery back-up. However, the cost of such a component is around $85 USD on average, with a low end of $75 USD. The Waveshare RS485 is not as expensive, but only has support for 3.3V input. It is not compatible with a CAN FD interface like the PiCAN is, and instead uses an SPI interface and Standard CAN protocol. This is a module that attaches on top of the hardware, known as HAT, which Raspberry Pi models are designed to be compatible with.

Another CAN bus compatible with the Raspberry Pi as a Hardware Attached on Top is the 2 Channel CAN FD Shield, which supports using a SPI interface as well as CAN FD. While this component is more expensive than the Waveshare bus, it offers us the correct voltage capabilities, and the more efficient CAN protocol. It is not priced too expensively, but still supported all of our project's needs with a simple enough attachment to our MCU that it would allow us to complete our project's basic goals. There is ample documentation and support for this hardware, making it a good fit for our project's finished design.

The CANable.io USB adapter is an open-source USB attachable, which plugs directly into our Raspberry Pi 5 and creates a new CAN node just by existing within the system. It is an open-source project, and was economically advantageous. It was a simple plug-and-play solution which made data transfer from our CAN PCBs to our Raspberry Pi 5 much easier and streamlined.

Due to how expensive these parts can be, we decided to use one of these module boards as a reference to then build our own. The best CAN controller to use is the MCP2515, and the best transceiver to use is the MCP2551. We ordered both of these parts from Digikey and created our own communication boards, then used the CANable.io USB adapter to make our custom CAN PCB boards compatible with our computer.

Table 3.8.3 - CAN Bus Comparison Table

Feature	2 Channel CAN FD Shield for Raspberry Pi	PiCAN FD	CANable.io USB adapter (Chosen)	Waveshare RS485/CAN
Protocol	Supports CAN FD, SPI Interface	CAN2.0B and CAN-FD	Supports CAN, CAN2.0B, CAN-FD,	SPI Interface
Voltage Input	5V	3.3V, 5V	3.3V-5V	3.3V
CAN Controller/ Transceiver	MCP2517FD/MCP2557FD	MCP2517FD/MCP2562FD	Uses its own interface to create candle-interface CAN module	MCP2512/SIT65HVD230DR
Cost	$29.90 USD	$74.75 USD	$29.99 USD	$13.99 USD

3.8.4 USB Communication Technology:

The joystick controller also needed a communication link between itself and the central Raspberry Pi. USB (Universal Serial Bus) technology is a widely adopted standard for connecting and transferring data between devices. It offers a versatile and robust solution for many applications, including our project, where we needed reliable and fast communication between the joystick controller and the main processing unit running our ROS/PyBullet simulation. USB provides several advantages that make it suitable for our needs, such as ease of integration, high data transfer rates, and support for both power and data transmission over a single cable.

USB technology was designed to standardize the connection of peripherals to computers, facilitating both communication and power supply. There are several versions of USB, including USB 1.0, 2.0, 3.0, and the most recent USB-C with many generations in between each. Each iteration has improved data transfer rates and power delivery capabilities. For instance, USB 2.0 can transfer data at up to 480 Mbps, while USB 3.0 can achieve speeds of up to 5 Gbps, making them suitable for the high-speed data communication required in our project.

One of the primary advantages of using USB technology in our project was its ease of integration. USB interfaces are ubiquitous and supported by a wide range of microcontrollers, including the Arduino Leonardo, which we used for our joystick controller. The Arduino Leonardo features a built-in USB controller, allowing it to emulate a USB Human Interface Device (HID) such as a keyboard or joystick. This simplified the development process, as we were able to leverage existing libraries and tools to implement USB communication without requiring additional hardware.

USB provided a reliable and high-speed communication channel between the joystick controller and the main processing unit. This was crucial for transmitting real-time data from the joysticks and buttons to the software, which then processed the input and sent back control signals to the robotic arm. The bi-directional nature of USB communication ensured that data flowed seamlessly between the hardware and software components, maintaining the responsiveness and accuracy needed for precise control while additionally, delivering power to the joystick controller, reducing the need for separate power supplies and simplifying the overall system design. The decision to use USB technology aligned with our project's goals of accuracy, reliability, and affordability. USB's high data transfer rates ensured that input from the joystick controller was communicated swiftly and accurately, while its ease of integration with the Arduino Leonardo simplified development and reduced costs. After comparing various USB technologies, we chose USB 2.0 because it was the only option with a microUSB to USB connection that works with the Arduino Leonardo, had speed capabilities that far exceeded what we required at a very affordable price, and we just so happened to have our own on-hand. Below is a comparison of the USB technologies we considered.

Table 3.8.4 - USB Comparison Table

Feature	**USB 1.0**	**USB 2.0 (Chosen)**	**USB 3.0**	**USB 3.1**	**USB 3.2**
Date Transfer Rate	Up to 12 Mbps	Up to 480 Mbps	Up to 5 Gbps	Up to 10 Gbps	Up to 20 Gbps
Compatible with Arduino Leonardo	N/A	Yes	No	No	No
Release Year	1996	2000	2008	2013	2017
Cost Per Unit	No longer sold	$1.15	$6.99	$11.30	$15.99

3.9 Software Comparison:

The purpose of our chosen software was to connect our teleoperation device input to our robotic arm, as well as simulating all movements and motor positions to properly correct the robotic arm through its given path of movement. The operating system and related software that we chose, such as the simulator used, was required to be capable of handling a real-time environment, which included speedy adjustments based on output sent from the current sensor hardware in the case of a collision. For our software to properly communicate with our hardware, we required detailed and dedicated libraries aimed at robotic development with the capability of working with a physics simulator. The simulator had to be capable of allowing us to fully and faithfully simulate our robotic arm at the time of operation. There are many simulation softwares that use their own modifications on the ODE physics engine to provide realistic physical reactions within a simulated environment to demonstrate the way our code and our physical hardware interact with each other, including but not limited to Matlab, Webots, and Gazebo.

The actual programming language that we used had to be dynamic and have a high degree of modifiability. Our robotic arm was designed in a way that gave us full control over the movement at the time of operation, and our programming language should reflect this control. During the process of comparing languages, we realized that using more user-friendly languages, such as Python, would loosen our constraint of time, but may not have given us the full amount of control we desired. In the end, this language is also far more compatible with software than other languages, and does not require as fine-tuned control as a C library would.

Choosing each component of our software was dependent on how quickly our team would be able to learn and utilize it while not sacrificing the needed higher-level features. The operating system of choice acted as a toolbox which provided to us all of the needed physical capabilities that our robotic arm required to be dynamic, dextrous, and reactive.

3.9.1 Operating System Software:

To properly communicate between our robotic arm, our encoders, and the operator, we decided to use the Robot Operating System (ROS) as the basis for our software-hardware connection. The most developed version at this point in time is the ROS 2: Humble, which has since allowed ROS 1 to be deprecated.

This system hosts a large set of libraries that we used for talking between our robotic components and our simulation software. The operating system is highly specified for a broad range of robotic development, which we used to facilitate the creation of a simulated environment that modeled our robotic arm and allowed for small adjustments via the encoders.

To properly host the ROS software, the robotic arm required some amount of programmable board embedded within it. The goal of using ROS was to avoid major software development like defining our own libraries, since that was not the focus of our project scope. With this criteria in mind, the official ROS documentation recommended installing binary packages, such as the Debian package, which allowed us to use an already-built installation of ROS 2 and begin working with it directly after download. It also took care of any dependencies and extra packages we may have needed, rather than sorting through and creating libraries from scratch or altering the ROS base files. These packages are most compatible with an Ubuntu Linux system, however, there are also binary package installs available for Window devices, excluding Debian packages.

The ROS 2: Humble documentation has many snippets of code that allowed us to build a first-time environment for our robot. The environment was built and set up according to the documentation and required no real unique input. Since we used the Raspian environment, the setup was simple, but many peripheral packages were needed to run a proper simulator.

As well as environment setup, the documentation provides several tutorials designed to hone developer’s skills in utilizing the ROS system. Most of the code used in our project on the ROS side did not need to be unique, but rather pieced together correctly in a way that fulfilled the requirements of our electrical components. By taking input from our physical components, it was fairly simple to feed this input into our software components and correctly calculate the needed distance for a particular movement.

ROS uses a system that assigns the roles of “subscriber” or “publisher” to separate devices, which allows them to receive and output data respectively. In our case, our tele-operational device was a “publisher”, outputting data and actions for our “subscriber” arm to consume and use. These definitions, along with three separate interfaces, allowed us to establish lines of data communication between devices to assist in our 3D simulation. Our project used the “topics” and “actions” interface in particular, since the “services” interface requires the addition of physical sensor data, which our base project did not include besides the positional sensor, which was used directly to provide feedback for our 3D model. The “topics” interface was used to allow both devices to speak consistently with each other through a shared and defined name that both devices could find easily. This also allowed our devices to have a steady stream of data as the robot continued to function, which included any positional errors and blockages the robotic arm encounters, indicated by the motor accuracy, which always tried to correct itself to its desired position. This position was below a defined limit, which the simulator did not control, but with proper implementation could be human-operated to replace the teleo-perative device. The “actions” interface allowed our tele-operation device to actually send desired movements to our robotic arm, as well as provide our robotic arm with telegraphed and recorded movements. Actions can be programmed in by the operator at the time of operation, or before operation as static, preset movements via the joystick, whose inputs are directly responsible for all input taken in and handled by the ROS interface.

There are numerous packages designed to make robotic functions and algorithms easier to program such as the subscriber and publisher roles. These packages also make communications between a simulator and our robotic arm possible beyond simply coding the movements for our robotic arm. ROS 2 can be used alongside several different physics simulators to realistically simulate the movements of our robotic arm, as the operating system acts as an interface between the simulator and the robotic device.

In particular, the ROS 2 packages are compatible with the Webots, Pybullet, and Gazebo simulators. The Webots simulator can be found on github for free or on the Cyberbotics website. It is by far the easier tool to use for those who are inexperienced in programming robotic devices or working with simulators.. However, Gazebo is better for a professional environment due to its broader reach in terms of features and high-end utility. It is far better for more advanced projects, which would occur frequently in industrial settings that require heavier and more complex machinery than what we intend to create.

Pybullet ended up being our ultimate choice, along with the ROS2: Humble distribution package.

Table 3.9.1 - Software Comparison Table

Feature	Robotic Operating System 2 (ROS 2: Humble)	Robotic Operating System 1 (ROS 1)
Testing and Support	Ubuntu, Windows 10, Linux, OS X El Captian, Community	Ubuntu, Community
Programming Language	C++11, C++17, Python 3.15	C++03, Python
Environment Setup	Separated into package-builds, Isolated	Requires referenced source files, Non-isolated
Support for Real-time	Real-time support using RTOS	No real-time support
Available tools, packages, services	Launch files in Python, Indexed resource look-up, ABI compatibility	Launch files in XML, Crawling resource look-up, Assumes ABI incompatibility

3.9.2 Simulators:

The modeling of our robotic arm was done using Onshape for CAD design. Once the 3D model was complete, it was converted into a URDF file compatible with PyBullet, our chosen simulator. Using PyBullet instead of Gazebo allows us to better meet our project requirements and streamline the simulation process. This approach minimizes errors and reduces the time spent on rebuilding or reworking the project in major ways. Time is a critical constraint in our project, as we only have a few months to research, understand, and build a functioning prototype. By reducing uncertainties, we loosen this constraint overall.

PyBullet will be used not only for debugging and simulation during development but also as the driving force for the robotic arm's "decision-making" during practical application. Compared to Gazebo, PyBullet offers a more straightforward and lightweight environment that suits our need for precise kinematic simulation. While Gazebo provides advanced features such as multiple physics engines, our focus on efficient debugging and accurate kinematic control makes PyBullet a better choice for this project.

The simulation environment is crucial for debugging, as this is a component we cannot fully test until the hardware is completed. Onshape enabled us to create a high-level prototype of our robotic arm with greater accuracy and ease of testing than alternatives such as Blender or Webots. While Gazebo offers extensive compatibility and a variety of physics engines, PyBullet's streamlined approach aligns better with our focus on project constraints, particularly regarding hardware compatibility and computational efficiency.

Our hardware platform, the Raspberry Pi 5, uses an ARM processor, which influences our choice of simulation software. PyBullet is fully compatible with ARM architectures, eliminating potential compatibility issues and allowing us to seamlessly integrate the simulator with our ROS 2-based system. This decision ensures that our simulation environment remains efficient and adaptable to our computational resources.

In summary, while other simulators like Gazebo or Simulink offer advanced capabilities, PyBullet's user-friendly environment and compatibility with our system make it the optimal choice for this project. Combined with Onshape for CAD and ROS 2 for integration, this workflow enables precise control and simulation of our robotic arm, ensuring the success of our prototype within the given time constraints.

Table 3.9.2 - Simulator Comparison Table

Feature	**Gazebo**	**Matlab**	**PyBullet (Chosen)**	**Webots**
Physics Engine	ODE (Default), Bullet, Simbody, DART (Compiled Separately)	Unreal Engine, Simscape	Bullet Physics Library	Custom built fork of ODE
3D Modeling	Yes, natively	Yes, with Simulink	Yes	Yes, natively
User Friendliness	Moderate learning curve	Not user friendly	Very user friendly	Very user friendly
Compatibility with ARM	Compatible	Compatible	Compatible	Not compatible

Environment				
Compatibility with ROS 2	Compatible by default	Compatible as a separate interface	Compatible	Compatible by default
Cost	Free	Free with an Education License	Free	Free

3.9.3 Programming Languages:

Our microcontrollers and our computer utilize two different programming languages. This is due to their capabilities and necessities for our project. Our microcontrollers, for example, need fine-tuned control over all SPI connections, and need a more robust library capable of giving the operator full control over the program. The Raspberry Pi on the other hand, needed to be capable of multithreading, complex computational processes and processing of large amounts of data. This is due to the need for simultaneous running of a real-time simulator, and scaling of joystick data, including recording and replaying keyframe movements from our joysticks.

With all of this in mind, we chose the C/C++ language for our microcontrollers. The ESP32 microcontrollers are already very robust systems, being powerful processors from the get-go. It was possible to use micro-python for these systems, however this would have been too limiting for our needed purposes. The languages between our computing device and microcontrollers did not need to talk, so each language was determined by the specific device's usage of it. With the use of C and the Arduino IDE, we had full control over all pins, algorithms, and functions we needed to run through our ESP32s to drive our motors, parse encoder and current sensor data, and pass information across our CAN bus.

While we are using the ROS as an interface to communicate between our code, robot, and simulators, we also had to choose a language to program our simulator. The language we used for this simulator was the Python language, as it offers fantastic user support, open-source packages, and a community-driven selection of libraries. Packages exist to handle CAN data, timestamping, simulating, and threading in concise and easy to handle ways.

Our chosen simulator, PyBullet, is an existing python interface with a 3D simulator capable of communicating with our ROS dependencies, which contributed to our choice to use python for all ROS and code on the Raspberry Pi.

Table 3.9.3 - Programming Language Comparison Table

Feature	**Python (Chosen: Raspberry Pi)**	**Mathworks**	**C (Chosen: ESP32s)**
User Friendliness	Very user friendly and syntax is easy to learn and use, but takes away some fine control from the programmer	Terminal is very specific when encountering errors, easy to debug	Not user friendly, requires great amount of familiarity and coding skill, but allows the coder to have finer control over code
Third-party Packages	Many available to download for specific needs	Many available to download for specific needs	Does not have many available
Built-in Functions	Many built-in functions designed to remove tedious work of many common algorithms	Many built-in functions designed to remove tedious work of many common algorithms	User will have to build functions from scratch
Cost	Free	$980 USD per year	Free
Open Source or Proprietary?	Open-source	Proprietary	Open-source, dependent on IDE

3.9.4 Control Algorithms

The choice of control algorithm is crucial for achieving precise and efficient operation of the robotic arm. Several control strategies were considered, each with its own set of advantages and limitations. The main options evaluated were Open Loop Control, Simple Position or Velocity PID Control, Field-Oriented Control (FOC) of torque, position, and velocity, and Bang-Bang Control.

Open Loop Control is the simplest form of control, where the system operates without feedback. In this method, the control action is predetermined and does not adjust based on the system's output. It is simple to implement and requires minimal computational resources, with no sensors needed, thus reducing system complexity and cost. It also offers quick response time as there's no feedback processing. However, open loop control lacks accuracy and precision, especially under varying conditions, and is unable to compensate for disturbances or changes in the system. It has no self-correction capability, leading to cumulative errors over time.

PID (Proportional-Integral-Derivative) control is a feedback control method that calculates an error value as the difference between a desired setpoint and a measured process variable, then applies a correction based on proportional, integral, and derivative terms. It offers improved accuracy compared to open loop control and can handle minor disturbances and changes in load. It is widely used in industry, with abundant resources and implementation examples. However, PID control requires tuning for optimal performance, which can be time-consuming. It may exhibit overshoot and oscillations if poorly tuned, and its performance can degrade in non-linear systems or when operating conditions change significantly.

Field-Oriented Control (FOC), also known as vector control, is an advanced motor control technique that provides precise control over torque, position, and velocity by manipulating the magnetic fields in the motor. FOC offers high precision and efficiency across all speed ranges, excellent torque control even at low speeds, smooth operation, and fast dynamic response. It is effective in handling non-linear systems and varying loads. However, FOC is more complex to implement, requiring advanced knowledge of motor theory. It demands more computational power, potentially requiring more powerful microcontrollers, and needs accurate rotor position feedback, necessitating high-resolution sensors.

Bang-Bang Control, also known as on-off control or hysteresis control, is a simple control strategy where the system switches abruptly between two states. It is very simple to implement with minimal computational requirements and offers fast response to large errors. It can be effective in systems with significant delay. However, bang-bang control suffers from poor precision, especially for small errors, and can cause wear on mechanical components due to frequent switching. It may also lead to limit cycling (oscillation around the setpoint).

Based on the comparison, We chose Bang-Bang Control for our robotic arm due to its simplicity and adequacy for our project's needs. Bang-Bang Control, also known as on-off control, operates by switching abruptly between two states. This simplicity minimizes computational requirements and eliminates the need for complex tuning or advanced feedback systems. While it lacks the precision of more advanced methods like PID or Field-Oriented Control, Bang-Bang Control is well-suited for applications where large corrections are sufficient, and precise adjustments are less critical. Given our project's requirements and constraints, this method effectively balances implementation simplicity with functional adequacy.

Table 3.9.4 - Control Algorithm Comparison Table

Feature	**Open Loop**	**Simple PID**	**FOC**	**Bang-Bang (Chosen)**
Precision	Low	Medium	High	Very Low
Complexity	Very Low	Medium	High	Very Low
Computational Requirements	Very Low	Low	High	Very Low
Torque Control	Poor	Medium	Excellent	Poor
Adaptability to Non-linear Systems	Poor	Medium	Excellent	Poor
Sensor Requirements	N/A	Medium	High	Low
Implementation Difficulty	Very Easy	Moderate	Difficult	Easy
Suitability for Precision Robotics	Low	Medium	High	Very Low

3.10 Power Supply:

The primary function of a power supply is to convert electric current from a source into the correct voltage, current, and frequency required to power the components of the robot arm and ensure proper operation. When designing the system, a decision must be made between using a linear or a switching power supply. A linear power supply operates by stepping down the AC voltage to a lower AC level through a transformer, which is then filtered and converted to DC. The resulting DC voltage is regulated using a series or shunt regulator to maintain a consistent output. This type of power supply is typically used in audio equipment, low-noise instrumentation, and other applications where minimal noise is critical. However, due to the excess voltage being dissipated as heat in the regulating transistor, a linear power supply is less efficient, with efficiency ranging from 30% to 60%. Additionally, the design tends to be bulky and heavy because of the large transformer and heat sinks required for heat dissipation. While linear supplies were historically favored in noise-sensitive applications, modern switching supplies often provide a more efficient and compact alternative without compromising performance.

A switching power supply would convert AC to DC using a rectifier. The DC voltage would then be switched on and off at high frequencies using a transistor. This pulsed voltage is passed through a high-frequency transformer or inductor and then filtered and regulated to provide a stable DC output. This kind of power supply would be used in computers, telecommunications, industrial equipment, and any application where efficiency and compact size are more critical than low noise. Because they convert energy more effectively and waste less power as heat, this kind of power supply is much more efficient, more specifically 80%-90%. Furthermore, switching power supplies were smaller and lighter, as the high-frequency transformer used in their design was much smaller than the low-frequency transformers required in linear power supplies. The reduced size and weight made switching power supplies more cost-effective, particularly for high-power applications, as they required less extensive heat dissipation mechanisms.

Despite these advantages, switching power supplies introduced certain challenges. The high-frequency switching produced electrical noise and ripple, which could interfere with sensitive electronics if not properly managed. Additionally, their design was more complex due to the need for multiple components, including control circuits, high-frequency transistors, and advanced filtering systems. This complexity required careful engineering to ensure reliability and performance, particularly in applications like robotics, where precision and stability were critical.

After comparing both options, we decided to use a switching power supply instead of a linear power supply for our robotic arm. This decision was made because the switching power supply was more efficient, physically smaller and lighter, generated less heat, and was more affordable than the linear alternative. Furthermore, the reduced heat generation minimized the need for bulky heat sinks, which was particularly beneficial for compact and portable designs like robotic arms. This made it an ideal choice for ensuring consistent performance while keeping costs and size manageable.

3.10.1 Battery Powered Supplies:

Several types of power supplies were considered for the robot arm, one of which was the battery power supply. This type was crucial for providing portable and reliable energy to the device. Battery power supplies comprise several essential components that ensure the efficient storage, management, and delivery of electrical energy. A key component of this type of power supply is the battery cells, which convert chemical energy into electrical energy. These cells can come in two main types: primary cells, which are non-rechargeable (e.g., alkaline, lithium), and secondary cells, which are rechargeable (e.g., NiCd, NiMH, Li-ion, Li-Po, lead-acid). In the past, secondary cells were particularly advantageous for robotic applications due to their ability to be recharged, making them more cost-effective and sustainable over time. If designers prioritized long-term efficiency and reduced environmental impact, secondary cells would likely have been the ideal choice. Among these, lithium-ion and lithium-polymer batteries stand out because of their high energy density, long lifespan, and lightweight construction. These attributes make them particularly suitable for compact and mobile robotic systems. Furthermore, their ability to deliver consistent power over extended periods has been a key factor in advancing the performance of portable robotic devices.

Another essential component of the system is the Battery Management System (BMS), which monitors and manages the battery pack to ensure safe and efficient operation. If a battery pack were to operate without a BMS, it could be prone to overcharge, over-discharge, or thermal damage, which would significantly reduce its lifespan and reliability. Several critical features have been integrated within the BMS: voltage and current monitors measure the voltage and current of each cell to prevent dangerous conditions such as overcharging, over-discharging, or overcurrent events. Temperature monitoring ensures that the cells operate within safe temperature ranges, protecting them from thermal damage or thermal runaway, which could lead to catastrophic failure. In the past, balancing circuits were introduced to equalize the charge across all cells, maximizing the battery pack's lifespan and overall performance. These balancing circuits, whether passive or active, manage individual cell voltages to ensure consistent operation. The BMS also includes a microcontroller that processes data and controls other components, such as sensors for measuring voltage, current, and temperature. Furthermore, a communication interface allows the BMS to exchange data with external monitoring and control systems, making it an intelligent and indispensable component for modern battery-powered devices. By incorporating such features, the BMS not only improves safety but also optimizes performance, ensuring that the battery system remains efficient and reliable over time.

A charging circuit is another critical component within the battery system. This circuit ensures that the battery pack is charged safely and efficiently. It controls the charging process to prevent overcharging and optimize battery life by employing constant current (CC) and constant voltage (CV) stages. If the charging circuit were not included, the battery could degrade over time, reducing both its performance and lifespan. Key components within the charging circuit include a charge controller, which manages the charging process, and an adapter, which supplies the necessary voltage and current for charging. The inclusion of these elements ensures that the battery can be recharged repeatedly without compromising its performance or safety. Furthermore, housing and connectors play a vital role in supporting the battery's functionality. The housing protects the battery cells and associated electronics from physical damage and environmental factors such as moisture, dust, and vibrations. Robust connectors ensure reliable electrical connections and efficient power delivery to and from the battery pack. In the case of the robot arm, durable wires and connectors are used to transfer power seamlessly between the battery, motor controllers, and other components.

There are several advantages to using batteries as power supplies. Battery power supplies are portable, allowing devices to operate independently of fixed power sources. This portability makes them essential for applications such as mobile phones, laptops, portable medical devices, and wearable technology, where mobility and autonomy are critical. If a device were dependent on a fixed power source, its functionality would be significantly restricted, especially in applications that demand flexibility.

One notable advantage is that many modern batteries, particularly lithium-based ones, offer high energy density, enabling significant energy storage in a compact and lightweight form. Furthermore, batteries provide immediate power without requiring startup time, making them invaluable for backup systems. This feature ensures seamless operation during power outages or emergencies, which is vital for critical applications such as hospitals, data centers, and safety systems. Finally, advancements in battery technology have allowed for customization to meet specific needs, such as fast-charging capabilities, long shelf life, or operation in extreme temperatures. For instance, if a system required rapid energy replenishment, fast-charging batteries could be utilized to reduce downtime significantly.

However, there are several limitations when batteries are used. Despite advancements in battery technology, batteries still have a lower energy density compared to some fossil fuels. This limitation reduces the range and endurance of battery-powered devices, making them less suitable for applications requiring long operational times or extensive mobility, such as electric vehicles or large-scale industrial systems. Another disadvantage is that batteries have a finite lifespan and degrade over time. In the past, this degradation often led to a gradual reduction in capacity and efficiency, necessitating periodic replacement and recycling. Moreover, recycling processes for certain types of batteries, such as lithium-ion, are complex and not always universally available, posing significant challenges for sustainable usage. Finally, batteries add significant weight to devices, especially in applications requiring large energy storage. This increased weight can negatively impact the overall design and efficiency of devices, particularly in portable systems like drones, robots, or medical equipment, where minimizing weight is crucial for performance. For example, in robotics, the added weight from batteries could reduce the payload capacity or mobility of a robot arm, thereby affecting its operational efficiency.

Battery power supplies play a crucial role in enabling the mobility, flexibility, and efficiency of robotic arms across various applications. For autonomous mobile robots (AMRs), robotic arms are often mounted on mobile platforms to perform tasks such as material handling, inspection, and pick-and-place operations. In the past, batteries provided the necessary power for both the movement of the mobile platform and the operation of the robotic arm, allowing it to operate without being tethered to a power outlet. This enhanced the range and flexibility of robotic systems. If batteries had not been implemented in such applications, the mobility of robotic arms would have been significantly restricted. Moreover, batteries enable continuous operation during power outages or in areas lacking proper power infrastructure. When robotic arms are designed to work alongside humans in shared workspaces for tasks such as assembly, packaging, and quality control, batteries power these devices, allowing them to function in environments where a fixed power supply is not feasible. Battery-powered collaborative robots, or cobots, can easily be moved and deployed to different workstations, providing unmatched flexibility in dynamic work environments. This eliminates the need for extensive wiring, reducing clutter and improving safety in the workspace.

For robotic arms used in manufacturing lines for welding, painting, assembly, and material handling, batteries provide backup power to ensure uninterrupted operation during power failures and support remote or flexible installations. This ensures that critical manufacturing processes are not disrupted during outages. Additionally, batteries allow robotic arms to be placed in locations without direct access to power sources. Finally, battery power is highly beneficial for robots used in healthcare, hospitality, retail, and domestic environments for tasks like assistance, delivery, and customer interaction. This type of power supply enables robots to operate independently, providing services without constant human intervention and navigating complex tasks in various environments. These capabilities significantly enhance the utility and versatility of robotic systems across industries.

3.10.2 Switching Power Supplies:

Another type of power supply that could be used in a robotic arm is a switching power supply. This type of power source converts alternating current (AC) from a wall outlet into direct current (DC), which is used by most electronic devices.Switching power supplies are particularly advantageous because they efficiently convert high-voltage AC into the lower, stable DC voltage required for robotic systems. Additionally, their compact size and high energy efficiency make them ideal for modern robotic applications. This type of power supply ensures consistent performance, reduces energy losses, and provides a reliable power source, which is critical for robotic arms' precise and continuous operation.

There are two types of AC-DC power supplies: a linear or switching power supply. A linear power supply uses a transformer to step down the AC voltage to a lower AC voltage, which is then filtered and converted to DC. The resulting DC voltage is regulated using a series or shunt regulator to maintain a constant output voltage. This kind of power supply is used in audio equipment, low-noise instrumentation, and other applications where low noise is critical. This linear power supply is less efficient (typically 30-60%) because the excess voltage is dissipated as heat in the regulating transistor. For sensitive analog and audio applications, low electrical noise and ripples would be produced. Physically, the linear supply would be bulky and heavy due to the large transformer and heat sinks required for dissipating heat. The linear power also has faster response time to changes in load, providing stable output quickly.

A switching power supply would convert AC to DC using a rectifier. The DC voltage is then switched on and off at high frequencies using a transistor. This pulsed voltage is passed through a high-frequency transformer or inductor and then filtered and regulated to provide a stable DC output. This kind of power supply would be used in computers, telecommunications, industrial equipment, and any application where efficiency and compact size are more critical than low noise. This kind of power supply is much more efficient (typically 80-90% or higher) because they convert energy more effectively and waste less power as heat. Furthermore, the switching power supply is smaller and lighter because the high-frequency transformer is much smaller than the low-frequency transformers used in linear supplies. Because of the size of the device and less need for extensive heat dissipation, it is generally cheaper for high power outputs. However, due to the high-frequency switching, high electrical noise and ripple would be produced. Moreover, the design is more complex due to high amounts of components.

These power supplies are composed of various components, each playing a critical role in converting and regulating electrical power. One of the key components are transformers. Their purpose is to convert AC voltage from one level to another. This is done by the primary coil, which receives the input voltage, the secondary coil, which delivers the transformed voltage. And the core, which provides a path for magnetic flux. Another component would be rectifiers. Rectifiers convert AC to DC by the use of diodes. The diodes would allow current to flow in one direction, converting AC to pulsating DC. Another key part of the power supply would be filters, which smooths out the pulsating DC from the rectifier. Within these filters consists of capacitors, which stores and releases charge, reducing ripple voltage and inductors, which provide high impedance to AC components, further smoothing the DC output. The final part of a power supply would be the voltage regulator. Their purpose is to maintain a constant output voltage despite variations in input voltage or load conditions. A pass transistor controlled by a feedback loop would be involved to maintain a constant output voltage.

When it comes to using the AC-DC Power supply, there are several benefits. AC-DC power supplies can be used in various applications, from small electronic devices to large industrial machinery, making them versatile solutions for different power needs. They can also adapt to various input voltages and frequencies, making them suitable for global use where electrical standards vary. Modern AC-DC power supplies, particularly switching power supplies, also offer high efficiency (typically 80-95%). They convert most of the input power into usable output power with minimal energy loss. High efficiency leads to lower energy consumption and reduced operational costs, which is beneficial for both consumers and industries. Another advantage is that AC-DC power supplies provide a stable DC output voltage, which is crucial for sensitive electronic devices and systems. They maintain a constant output voltage despite variations in input voltage or load conditions, ensuring reliable performance. Their smaller size allows easier integration into various electronic devices and systems. Finally, the overvoltage and overcurrent Protection feature would protect both the power supply and the connected devices from damage due to excessive voltage or current. Also, the short-circuit protection feature Prevents damage and enhances safety by disconnecting the power supply during a short circuit.

Using an AC-DC power supply would also involve some limitations. Switching power supplies can generate electromagnetic interference, which can affect the performance of nearby electronic devices. Additional EMI filtering and shielding are often necessary to minimize interference, adding to the design complexity and cost. Moreover, switching power supplies have more components and complex circuitry compared to linear power supplies, which can increase the risk of failure and require more sophisticated design and manufacturing processes. Furthermore, In applications with highly variable loads, maintaining stable output voltage can be challenging and may require advanced regulation techniques. Sudden changes in load can cause temporary fluctuations in output voltage, which may affect sensitive electronic devices. Finally, high-quality AC-DC power supplies with advanced features and high efficiency can be expensive compared to simpler, less efficient models, which may require regular maintenance and have higher repair costs if they fail.

AC-DC power supplies are critical components in the design and operation of robotic arms. They provide the necessary direct current (DC) power required for the various subsystems within a robotic arm. One of those subsystems is the control system.The components within the system include microcontrollers, microprocessors, and sensors. These include encoders, gyros, and proximity sensors. There are also the communication modules like Wi-Fi, Bluetooth, and the CAN buses. The control system typically requires a stable DC voltage (e.g., 5V, 3.3V). Thus, leading to low electrical noise, which is crucial to avoid interference with sensitive microcontrollers and sensors. The roles of AC-DC Power Supply within the control system include providing a stable and regulated DC voltage to ensure reliable operation of microcontrollers and sensors and ensuring minimal electrical noise, which is essential for the accurate functioning of the control system. Another subsystem within the robot arm is the actuator. The actuator system contains various electric motors (e.g., brushed DC motors, brushless DC motors, stepper motors) and each of them drives the movement of the robotic arm's joints and end effectors. Another part of the actuator are servos, which provide precise control of position and speed, often used in applications requiring high accuracy. Since significant current is required, especially during startup and heavy load conditions, the power supply provides sufficient current and power to drive motors and servos. High-efficiency power supplies reduce heat generation and improve overall system performance, as well as overcurrent and short-circuit protection features.

3.10.3 Regulated Power Supplies:

The final type of power supply that would be favorable when designing a robotic arm is the regulated power supply. A regulated power supply is a device that provides a stable and consistent voltage output, regardless of variations in input voltage or load conditions. This reliability is essential in many electronic applications, including robotics, to ensure the predictable and safe operation of sensitive electronic components. If an unregulated power supply were used instead, fluctuations in voltage could have caused erratic behavior or damage to critical components in the robotic arm. There are two types of regulated power supplies that could be used: linear and switching power supplies. Linear power supplies are known for their simplicity and low noise but are less efficient and bulkier compared to switching power supplies. On the other hand, switching power supplies offer high efficiency, compact size, and better energy management, making them ideal for most robotic applications.

Linear regulated power supplies are a type of power supply that provides a constant output voltage by using the basic principle of using a voltage regulator dissipating excess power as heat. There are two types of linear regulators. In a series regulator, a variable element (usually a transistor) is placed in series with the load. The regulator adjusts the resistance of the transistor to maintain a constant output voltage. Another type of linear regulator is shunt regulator. In a shunt regulator, a parallel component (often a Zener diode) is used to regulate the voltage by shunting excess current away from the load. The Zener diode maintains a constant voltage across the load by conducting current when the input voltage exceeds the desired output voltage during the operation. Excess current is then diverted through the diode, keeping the output voltage stable.

Switching regulated power supplies, also known as switch-mode power supplies (SMPS), are highly efficient power converters that use high-frequency switching techniques to convert electrical energy. The fundamental principle of a switching regulated power supply would involve switching an electronic switch (such as a transistor) on and off rapidly to convert electrical energy from one form to another. Energy storage elements, such as inductors and capacitors, would be used to smooth the output voltage and filter out noise. There were four types of switching regulators that were researched: Buck Converters (Step-Down Converter) step down the input voltage to a lower output voltage. The switch (typically a MOSFET) would turn on and off at high frequency. When the switch is on, energy would be stored in the inductor and when it is off, the energy stored in the inductor would be transferred to the output load through a diode. Another type would be a Boost Converter (Step-Up Converter). This device would step up the input voltage to a higher output voltage. When the switch turns on, the current flows through the inductor, storing energy. When the switch turns off, the inductor releases its energy through a diode to the output capacitor and load.

A Buck-Boost Converter would step up or step down the input voltage depending on the requirements, combining the principles of both buck and boost converters. Thus, output voltage could either be higher or lower than the input voltage. Finally, a Flyback Converter would then provide electrical isolation between input and output while stepping up or stepping down the voltage. How it would operate is that it uses a transformer to store energy when the switch is on and transfer energy to the output when the switch is off. The secondary winding of the transformer would deliver power to the output through a diode.

Regulated power supplies came with several benefits over other types of power sources. The regulator provided a constant voltage, regardless of variations in input voltage or load conditions, ensuring the reliable operation of electronic devices. This was critical for sensitive electronics like microcontrollers, sensors, and communication devices. Furthermore, regulators included protection features such as overcurrent, overvoltage, and short-circuit protection. These safeguards would have protected both the power supply and the connected devices, thereby enhancing longevity and safety. Another benefit to using regulators could have been the production of low output noise and ripple. In particular, linear regulated power supplies produced an output that was smooth with minimal noise and ripple. This feature would have been ideal for applications requiring clean power, such as audio equipment, medical devices, and precision instruments, where stability and noise reduction were essential for optimal performance.

Regulated power supplies also had some disadvantages when used in devices. Linear regulated power supplies, in particular, were less efficient as they dissipated excess input power as heat. This inefficiency could have been especially problematic when the input voltage was significantly higher than the output voltage, negatively affecting the device's performance. Another disadvantage of the regulator was that the heat generated by less efficient linear regulators necessitated additional cooling solutions, such as heat sinks or fans, which increased the size, weight, and cost of the power supply. This could have limited their use in compact or portable devices where space and weight were at a premium. Furthermore, regulated power supplies were often limited in their maximum output power, making them unsuitable for high-power applications. Finally, switch-mode power supplies (SMPSs) were more complex to design and build compared to linear regulators, as they involved more components and sophisticated control mechanisms. This complexity would have translated to higher initial costs and potentially higher maintenance costs over the device's lifetime. Moreover, SMPSs could generate electromagnetic interference due to their high-frequency switching operation, which might have affected nearby sensitive electronic equipment. This would have required additional filtering and shielding, further adding to the cost.

Regulated power supplies play a crucial role in the operation of robot arms, ensuring that all components receive stable and reliable power. For example, if a robotic arm is tasked with industrial automation, the regulator ensures consistent performance and precision in repetitive tasks by providing stable power to motors and control systems.This is important because this would reduce downtime and maintenance by protecting against voltage spikes and ensuring reliable operation in high-demand environments. If a robot arm was used for medical purposes, the regulator provides the high precision and stability required for delicate and intricate procedures. Regulated power is critical for the safety and accuracy of medical interventions. This type of power would protect sensitive medical equipment from power fluctuations, ensuring patient safety and procedural success. Finally, if a robot arm would be needed for research and development purposes, the regulator supplies stable power for experimental setups, ensuring that test results are reliable and repeatable. This would enable accurate evaluation of new designs and technologies by eliminating power-related variables.

3.10.3 Power Distribution Table:

A power distribution table served as a critical tool in the design and creation of a robot arm, as it provided a detailed overview of how electrical power was allocated across various components. Its primary purpose was to ensure that each subsystem, such as motors, sensors, microcontrollers, and actuators, received the appropriate voltage and current for proper operation. By creating a power distribution table, engineers were able to identify the total power requirements of the robot arm, which was essential for selecting an adequate power supply. The table outlined the voltage levels required by each component, the corresponding current draw, and the total power consumption for the entire system. This information helped prevent power imbalances or inefficiencies, which could have led to overheating, underperformance, or potential damage to sensitive components. By systematically organizing power requirements, the table also aided in troubleshooting and future upgrades, making it a vital element in the successful design and operation of the robot arm.

Table 3.10.3 - Power Distribution Table

Description	Current (A)	Voltage (V)	Power (W)	Quantity	Max Total Power (W)
Stepper Motor Controller	4	24	96	3	384
End Effector Servo	160m	6	0.96	1	0.96
ESP32 WRoom MCU	1100m	3.6	3.96	3	11.88
Position Sensor Encoder	12m	3.3	0.0396	3	0.1188
Current Sensor	14m	5	0.07	3	0.21
CAN Bus Controller	10m	5.5	0.055	3	0.22
CAN Bus Transceiver	200m	4.5	0.9	3	2.7
3.3V-5V Boost Regulator	2.5	5.5	13.75	1	30.25

24V-3.3V Buck Regulator	3	24	72	1	72
24V-5V Buck Regulator	3	24	72	1	72
Logic Level Driver	25m	0.5	0.0125	1	0.0125
Total Power			574 Watts		

3.10.4 Power Supply Technological Choice:

When choosing the best power supply for the robotic arm, we analyzed the advantages and disadvantages of each type and concluded that the switching power supply was the optimal option. Since the robotic arm was stationary, it would have most likely required a stable and reliable source of AC power. The AC-DC power supply was deemed the best choice due to its consistent power delivery, minimal maintenance requirements, and enhanced safety features, which ensured reliable operation over extended periods. Additionally, it eliminated the need for frequent replacements or recharging, making it a cost-effective and practical solution. Therefore, this type of power supply would have been the most suitable option for a stationary robotic arm, providing both efficiency and reliability in its operation.

Table 3.10.4 - Power Supplies Technological Comparison Table

Feature	Batteries	AC-DC Power Supplies (Chosen)	Regulated Power Supplies
Power Source	Chemical Energy	Alternating Current (AC)	Can be AC or DC
Output Type	Direct Current (DC)	Direct Current (DC)	Highly Stable Direct Current
Portability	High(Portable & mobile)	Low (requires an AC outlet)	Medium (depends on the power source)
Runtime	Limited by battery capacity	Continuous (as long as connected to AC)	Continuous (as long as power source is stable)
Efficiency	Varies (depends on type and capacity)	Moderate to high	High (due to regulation)

Feature	Batteries	AC-DC Power Supplies (Chosen)	Regulated Power Supplies
Maintenance	Moderate (requires recharging/replacement)	Low	Low
Cost	Variable	Moderate to high	High (due to regulation & precision)

3.10.5 Power Supply Part Choice:

The brand of the power supply would have needed to be chosen after deciding on the type of power supply, which in our case was the AC-DC power supply. First, we calculated the total power that the robotic arm would require to operate efficiently. After determining these requirements, we compared various brands of power supplies based on their features and specifications. The primary features we focused on were efficiency and price, as these were critical for ensuring both cost-effectiveness and performance. Other important considerations included the size and reliability of the power supply to ensure it could seamlessly integrate into the system and provide consistent power delivery.

Eventually, the decision was narrowed down to three brands: Mean Well, Delta Electronics, and TDK-Lambda. After thoroughly comparing and contrasting their characteristics, it was determined that the Mean Well LRS-600-24 would have been the best power supply to use. This decision was based on its superior balance of performance, reliability, and cost-effectiveness, making it the ideal choice for the stationary robotic arm. By selecting this model, we ensured that the power supply would meet all operational and budgetary requirements without compromising on quality.

Table 3.10.5 - Power Supply Price Comparison

Feature	Mean Well LRS-600-24 (Chosen)	Delta Electronics PJT-24V600WBAA	TDK-Lambda RWS600B-24
Power	600W	600W	600W5
Efficiency	91%	89%	88%

Feature	Mean Well LRS-600-24 (Chosen)	Delta Electronics PJT-24V600WBAA	TDK-Lambda RWS600B-24
Size	Compact	Compact	Compact
Reliability	Very High	Very High	Very High
Cost	$29.99	$175.46	$195.24

Chapter 4: Standards and Design Constraints

Standards can be defined both by official rulebooks and regulations, as well as by marketplace trends. The standards we follow for our project design are aimed at conformity, adaptability, and ethical practice. We have chosen components that follow standards which conform to size and/or design restrictions, giving us the advantage of interchangeability between parts both during and after the design process. Our standards also serve to provide strict guidelines on ethical practices during the process of design and development of our project.

Constraints define the limitations and restrictions that are imposed upon our project via physics, time, budgeting, and technological capability. Even after an ultimate choice for components and digital interfaces is made, we must keep their various constraints in mind continuously, working with and around them throughout the project design and development process. These restrictions include time of development, cost of the component, compatibility with other chosen components, and safety.

In this chapter, we will describe the specific design standards and constraints for our particular hardware and software components.

4.1 Hardware Standards and Constraints:

4.1.1 Joystick Standards:

4.1.1-1 ISO 9241-410:

ISO 9241-410 is a standard focusing on the ergonomics of human-system interaction, specifically providing criteria for the design of physical input devices such as keyboards, mice, trackpads, and in our project's case, joysticks. This standard ensures that input devices are designed to accommodate the capabilities and limitations of users, promoting effectiveness, efficiency, and satisfaction in use. ISO 9241-410 outlines several key ergonomic design criteria that must be met for joysticks to ensure optimal usability. One of the primary criteria is appropriateness for the intended tasks and use environment. This involves designing the joystick to achieve the required effectiveness and efficiency for precise control tasks. The joystick must be easy to operate, with intuitive controls that do not cause user fatigue or discomfort during extended use. This is critical in our project as the joystick will be used for controlling a robotic arm, requiring precise and sustained input from the user.

The standard also emphasizes the importance of reliable and responsive control to prevent unintended movements. The joystick's movement should be predictable and consistent, ensuring stable operation. This helps in minimizing the biomechanical load on the user, allowing for a natural range of motion that aligns with ergonomic needs. By minimizing the risk of repetitive strain injuries, the design enhances user comfort and efficiency over prolonged periods. Ergonomics for hand-held or hand-sized joysticks differ from thumbsticks, with thumbsticks requiring less force and providing finer control, making them suitable for our project's precision needs.

By adhering to ISO 9241-410 and addressing these constraints, the joystick design can meet the ergonomic and functional requirements necessary for effective and efficient control, while also maintaining durability and reliability in various operating conditions. This comprehensive approach to design ensures that the joystick will provide a high-quality user experience and meet the demanding needs of its application.

4.1.1-2 ISO 10218-1:

ISO 10218-1 focuses on the safety requirements for industrial robots and robotic devices. This standard is crucial for ensuring the safe design, protective measures, and information use of industrial robots, which include joysticks used for controlling these robots. Although primarily aimed at industrial settings, the safety principles established by ISO 10218-1 can also be applied to other robotic applications, making it relevant to our project.

ISO 10218-1 outlines several key safety design criteria that must be met for joysticks used in robotic control systems. One of the primary criteria is ensuring that the joystick can achieve the required effectiveness and efficiency for precise control tasks while maintaining safety. This involves designing the joystick to prevent accidental or unintended movements that could lead to hazardous situations. The joystick must be reliable, with consistent performance under various operational conditions, including extreme environments.

The standard also emphasizes the importance of protective measures, such as incorporating fail-safes and redundancy in the joystick’s design. These measures ensure that in the event of a component failure, the joystick can still operate safely or shut down in a controlled manner to prevent accidents. This is particularly important in applications where the joystick is used to control heavy or potentially dangerous machinery. In the case of our project this will come in the form of an emergency off button.

ISO 10218-1 also specifies the need for clear and comprehensive information for use. This includes providing detailed instructions for the operation, maintenance, and troubleshooting of the joystick device. Proper documentation helps ensure that users can operate the joystick safely and effectively, reducing the risk of misuse or accidents.

By adhering to ISO 10218-1 and addressing these constraints, the joystick design can meet the stringent safety and reliability requirements necessary for effective control in industrial and robotic applications. This comprehensive approach to design ensures that the joystick will provide precise and reliable control while enhancing the overall safety of the machinery it operates.

4.1.1-3 Design Constraints:

In designing the joystick according to ISO 9241-410 and ISO 10218-1 standards, several design constraints must be considered to ensure compliance and optimal performance.

Space Constraints:

The physical space available for integrating the joystick into the control unit is limited. This necessitates a compact design for all components, including the joystick, buttons, and microcontroller. The joystick must fit within the allotted space without compromising its functionality or ergonomic design. This constraint influences the choice of joystick model and the arrangement of other components within the control unit.

Power Consumption:

Power efficiency is critical for ensuring the joystick operates effectively without draining the power supply. Components must be selected for their low power consumption, particularly the microprocessor, which should manage inputs and communication efficiently to conserve energy. This constraint ensures that the system can run for extended periods without frequent recharging or power issues, enhancing the usability and reliability of the joystick.

Environmental Conditions:

The joystick must be designed to operate under various, reasonable environmental conditions, such as temperature fluctuations, humidity, and exposure to dust or moisture. This requires selecting materials and components that meet relevant environmental standards and testing them to ensure they can withstand these conditions. The joystick must remain functional and reliable in diverse environments, ensuring consistent performance regardless of external factors.

4.1.2 Button Standards:

4.1.2-1 IEC 60947-5-1:

IEC 60947-5-1 is a standard that specifies the requirements for the control circuit devices and switching elements, particularly for push buttons used in industrial environments. This standard ensures the safety, reliability, and performance of push buttons, which are critical components in control systems for machinery and equipment. The standard covers various aspects, including the mechanical and electrical endurance of the push buttons, their resistance to environmental factors, and their ability to provide clear and unambiguous operation for the user.

IEC 60947-5-1 outlines the design criteria for push buttons to ensure they can withstand the rigors of industrial use. This includes specifications for the materials used, the mechanical strength, and the operational life of the buttons. Push buttons designed according to this standard must endure a specified number of operating cycles without failure and must resist dust, moisture, and other environmental factors that could impair their function. This ensures that push buttons remain reliable over long periods, even in harsh conditions.

The standard also emphasizes the importance of ensuring that push buttons provide clear tactile feedback to the user. This feedback helps operators confirm that their input has been registered, which is crucial in environments where quick and precise control is necessary. Additionally, IEC 60947-5-1 specifies the electrical characteristics of push buttons, including their voltage and current ratings, to ensure safe operation within control circuits.

4.1.2-2 Design Constraints:

Space Constraints:

Push buttons must be compact to fit into limited spaces within control units. This necessitates careful selection of button models that can be integrated seamlessly into the overall design without sacrificing functionality. The small size of the buttons must not compromise their accessibility or ease of use, which is critical in maintaining efficient control operations.

Power Consumption:

The push buttons need to be energy-efficient to ensure they do not significantly contribute to the overall power consumption of the control system. Low-power components are preferred to extend the operational life of battery-powered devices and reduce the frequency of maintenance. This is particularly important in systems that rely on minimal power sources and require long-term reliability.

Environmental Conditions:

Push buttons must be robust enough to operate reliably under various environmental conditions, including exposure to dust, moisture, and temperature variations. IEC 60947-5-1 specifies that buttons must be constructed from materials that can withstand these conditions. This includes using corrosion-resistant materials and ensuring the buttons are sealed against contaminants. Compliance with these environmental standards ensures that the push buttons can maintain their performance and safety in diverse settings.

By adhering to IEC 60947-5-1 and addressing these constraints, the push button design can meet the safety, reliability, and performance requirements necessary for effective control in industrial and electronic applications. This comprehensive approach to design ensures that the push buttons will provide a high-quality user experience and meet the demanding needs of our application.

4.1.3 CAN Bus:

4.1.3-1 Standards:

CAN buses are always built with error-handling techniques embedded within the technology. One of these techniques built into the communication bus is bit-stuffing, which is a common standard among CAN buses. These buses also follow set sizes of frames and data sent dependent on the protocol the CAN bus is built to follow. These protocols can cause a lack of backwards compatibility in the case of different protocols being used across communication devices. If the same protocols are used, then these standards allow the buses to communicate cleanly and handle errors efficiently.

4.1.3-2 Constraints:

Specific protocols may introduce compatibility constraints with certain devices. Using the incorrect CAN protocol can result in data errors and render the receivers of CAN buses incapable of handling messages sent. This occurs when the protocol used for the CAN bus for one node sends a higher bit identifier than the node receiving the message, making messages between buses incompatible. This is why we used the same standard CAN protocol for all devices.

The cost of CAN bus modules can also be quite costly, and a single CAN bus device was needed for each microcontroller used within our project. This made our options extremely limited, since it could have easily caused the cost of our project to skyrocket. We chose instead integrated chips within the same family that were compatible and easy to purchase and replace upon damage.

4.1.4 USB Cable Standards:

In our robotic arm project, adhering to USB standards is essential for ensuring reliable communication between various components and the central control system. USB (Universal Serial Bus) technology offers a versatile and robust solution for many applications, providing high data transfer rates and support for both power and data transmission over a single cable. The following outlines the key aspects of USB standards relevant to our project:

4.1.4-1 USB 2.0:

USB 2.0 offers data transfer rates of up to 480 Mbps, making it suitable for peripherals such as keyboards, mice, and certain sensors. This standard is widely compatible and provides adequate speed for low to moderate data transfer requirements. USB 2.0 typically uses Type-A and Type-B connectors, which are prevalent and well-supported across many devices. Additionally, USB 2.0 cables can deliver power up to 2.5 watts (5V, 0.5A), making them useful for powering smaller components.

4.1.4-2 USB 3.0 and 3.1:

USB 3.0 and 3.1 significantly enhance data transfer capabilities, with USB 3.0 supporting up to 5 Gbps and USB 3.1 reaching up to 10 Gbps. These versions are ideal for high-speed data transfer applications, such as connecting cameras or data-intensive sensors. USB 3.0 and 3.1 are backward compatible with USB 2.0, ensuring versatility and ease of integration. They include Type-A, Type-B, and the versatile Type-C connectors. Additionally, these standards support higher power delivery, with USB 3.0 and 3.1 capable of delivering up to 4.5 watts (5V, 0.9A).

4.1.4-3 USB 4.0:

The latest USB standard, USB 4.0, supports data transfer rates of up to 40 Gbps, making it the best choice for extremely high-speed data transfer needs, such as real-time video streaming or high-bandwidth sensors. USB 4.0 uses the Type-C connector exclusively and supports USB Power Delivery (USB PD), offering power delivery up to 100 watts (20V, 5A). This capability makes USB 4.0 highly suitable for power-hungry applications and ensures efficient power management in our robotic arm project.

4.1.4-4 Key Features:

Power Delivery:

USB cables not only transmit data but also deliver power to connected devices. The power delivery capabilities vary across versions, with USB 2.0 providing up to 2.5 watts, USB 3.0 and 3.1 up to 4.5 watts, and USB 4.0 supporting up to 100 watts through USB Power Delivery (USB PD).

Backward Compatibility:

USB standards are designed to be backward compatible, ensuring older devices can still connect to newer ports. This feature is crucial for maintaining the longevity and versatility of our robotic arm's components.

Plug-and-Play Capability:

USB's plug-and-play feature simplifies the connection process, allowing devices to be connected and recognized without the need for additional drivers. This ensures ease of setup and operation for the robotic arm's components.

Data Integrity and Error Handling:

USB standards incorporate robust data integrity checks and error handling mechanisms, which are critical for the reliable operation of the robotic arm. This ensures accurate data transmission between sensors, controllers, and actuators.

By leveraging USB standards, specifically that of USB 2.0, we can achieve a reliable, high-speed communication network that enhances the overall functionality and safety of our robotic arm system. USB technology will be used to connect high-speed sensors, transmit data between the joystick controller and the main processing unit, and ensure efficient power delivery to various components. Adhering to these standards ensures that our project meets industry benchmarks for performance, compatibility, and safety.

4.1.5 NEMA Stepper Motor Standards:

The National Electrical Manufacturers Association (NEMA) has established standardized sizes for stepper motors, which significantly simplified our selection process for the robotic arm project. These standards define the physical dimensions of the motor's faceplate, ensuring consistent mounting options across different manufacturers. NEMA sizes for stepper motors are denoted by a number that corresponds to the width of the motor's faceplate in tenths of an inch. For instance, NEMA 17 motors are 1.7 inches (43.2 mm) wide, NEMA 23 motors are 2.3 inches (58.4 mm) wide, and NEMA 34 motors are 3.4 inches (86.4 mm) wide.

Importantly, NEMA standards also specify safety and tolerance ratings for these motors. NEMA 17, 23, and 34 motors are typically rated for industrial use, with operating temperatures ranging from -20°C to 40°C (-4°F to 104°F). They are designed to meet IP40 protection standards as a minimum, offering protection against solid objects larger than 1mm but no specific protection against water ingress. For more demanding environments, some manufacturers offer variants with higher IP ratings, such as IP65, which provides dust-tight sealing and protection against low-pressure water jets.

In terms of tolerances, NEMA standards specify tight manufacturing tolerances for motor dimensions. For instance, the faceplate width tolerance is typically ±0.5mm (±0.02 inches), ensuring consistent fit across different manufacturers. Shaft diameters and lengths also have specified tolerances, usually within ±0.013mm (±0.0005 inches), which is crucial for precise positioning in robotic applications like ours.

These standardized sizes and specifications played a crucial role in narrowing down our search for the appropriate stepper motor. The NEMA standards provided us with exact measurements and tolerance ranges for each motor size category, allowing us to quickly assess which motors would fit within our design's physical constraints and meet our safety requirements. This was particularly important for our compact robotic arm design, where precision and reliability are paramount.

Additionally, NEMA standards ensure consistent mounting hole patterns for each size category, giving us confidence that we could easily integrate the chosen motor into our design and find compatible mounting hardware. The interchangeability offered by these standardized dimensions meant that we could consider motors from multiple manufacturers within the same NEMA size category, knowing they would be physically interchangeable and meet the same basic safety and tolerance specifications.

By leveraging these NEMA standards, including their safety ratings and tolerance specifications, we were able to quickly focus our search on specific size categories that met our physical design requirements and operational needs. This standardization significantly streamlined our component selection process, allowing us to more efficiently compare options within relevant size categories and ensure compatibility with our overall system design. The NEMA standards thus proved invaluable in guiding our motor selection, helping us balance performance requirements with physical constraints and safety considerations in our robotic arm project.

4.1.6 Power Supply:

Safety standards for power supplies are critical to ensure that these devices operate safely and reliably, protecting users and equipment from hazards such as electric shock, fire, and electromagnetic interference. Here are detailed explanations of the key safety standards and the organizations that develop and enforce them:

4.1.6-1 Standards:

UL 60950-1: This is the standard for information technology equipment safety. It covers various aspects, including electrical insulation, protection against electric shock, fire enclosures, and temperature limits.

UL 62368-1: This standard replaces UL 60950-1 and UL 60065. It is a hazard-based standard applicable to audio, video, information, and communication technology equipment. It focuses on identifying and mitigating potential hazards to users and equipment.

IEC 60950-1: Similar to UL 60950-1, it is an international standard for safety in information technology equipment.

IEC 62368-1: The international counterpart to UL 62368-1, it covers a broad range of products and focuses on hazard-based safety engineering principles.

The CE Mark indicates compliance with EU safety, health, and environmental requirements. For power supplies, compliance typically involves meeting the requirements of the Low Voltage Directive (LVD) 2014/35/EU, which ensures that electrical equipment operates safely within its specified voltage range.

The ATX standard specifies the physical dimensions, electrical outputs, and connector types for desktop power supplies.This includes voltage outputs (+3.3V, +5V, +12V, -12V, and +5VSB), current capacities, and load regulation.Standardized connectors such as the 24-pin motherboard connector, 4/8-pin CPU power connector, and PCIe connectors. Physical dimensions and mounting points to ensure compatibility with ATX chassis.

The 80 PLUS certification program rates power supplies based on their efficiency at various loads (20%, 50%, and 100% of rated capacity). Higher efficiency levels correspond to less energy waste and heat generation.

4.1.6-2 Constraints:

Constraints for power supplies encompass a range of electrical, thermal, and environmental considerations. These constraints ensure that power supplies operate reliably within their specified parameters and protect both the power supply and the devices they power. There were a number of electrical constraints. Some of these electrical constraints include voltage regulation, current capacity, noise, and efficiency. For voltage regulation, line and load regulation would need to be involved. Line regulation is the ability of the power supply to maintain a constant output voltage despite variations in the input voltage. Load regulation is the ability to maintain a constant output voltage despite changes in the load. Moreover,the power supply would need to provide the highest current to the load without exceeding its design limits.Some power supplies require a minimum load to maintain stable operation. Operating below this threshold could lead to instability or improper functioning. When it comes to the constraints of noise, minimizing noise is crucial for ensuring the proper operation of sensitive electronic devices. Finally, when making the robot arm, it would need to be highly efficient. High efficiency means less energy is wasted as heat, which is crucial for reducing energy consumption and thermal stress.

There were also some thermal constraints that needed to be considered. The power supply must effectively dissipate heat to prevent overheating and ensure reliable operation. Heat dissipation can be managed by the use of heat sinks, cooling fans, and proper ventilation. Moreover, the power supply must operate reliably across the temperature range encountered in the robot arm's environment. This can be solved by selecting components rated for the expected temperature range and incorporating thermal management solutions.

Considerations of environmental restrictions needed to be regarded as well. For instance, The power supply must be protected against dust, moisture, and other environmental contaminants. The solution is to use enclosures with appropriate IP ratings and sealing techniques that can protect against environmental factors. Furthermore, The power supply components must be resistant to humidity and corrosion to ensure long-term reliability. Conformal coatings and corrosion-resistant materials help with enhancing durability in humid environments.

When using switching power supplies, physical and mechanical constraints had to also be considered. Firstly, the power supply had to be compact and lightweight to fit within the limited space and weight budget of the robot arm. This was resolved by using high-density power supply designs and selecting compact components that can help manage size and weight constraints. Moreover, the power supply must be securely mounted and capable of withstanding vibrations and shocks experienced during the robot arm's operation. Robust mounting solutions had to be implemented and components that were highly rated for high vibrations and shock environments in order for durability of the power source.

Extra constraints when using a switching power supply included budget limitations. The cost of the power supply had to fit within the overall budget for the robot arm project. Balancing performance requirements with cost and exploring cost-effective alternatives without compromising critical features can help manage that restriction.

4.1.7 Current Sensors:

Current sensors for a robot arm are crucial for monitoring and controlling the electrical current passing through the motors and other components. The standards for these sensors ensure accuracy, reliability, and safety. These are the list of standards for the current sensors

4.1.7-1 Standards:

IEC 60947-2: This standard covers low-voltage switchgear and controlgear, including requirements for current sensors.

IEC 61557: Specifies the performance requirements for current sensors used in electrical installations.

ANSI C37.90: Relates to the protective relays and related devices, including current sensors used in protection systems.

IEEE 1584: Although primarily focused on arc flash calculations, it includes guidelines on current sensing for protection and safety.

IEEE C37.20.1: Standard for metal-enclosed low-voltage power circuit breaker switchgear, relevant to current sensors in such environments.

4.1.7-2 Constraints:

Current sensors in a robot arm application face several constraints that need to be carefully managed to ensure optimal performance and reliability. First, there can be some physical constraints when it comes to using current sensors. The current sensor must be compact and lightweight to fit within the limited space of the robot arm without adding significant weight, which could affect the arm's agility and precision. Moreover, The sensor must be easily mountable and integrable into the existing mechanical and electronic design of the robot arm. There would also be some electrical constraints as well when implementing current sensors. The sensor must cover the full range of currents expected during all operating conditions, including startup surges, normal operation, and peak loads. Moreover, high accuracy and precision are essential for precise control of the robot arm’s movements and for protection against overcurrent conditions. Using high-quality sensors with low error margins and regular calibration can help maintain accuracy. Furthermore, the sensor’s power consumption should be minimal to avoid excessive power draw from the robot arm’s power supply, so using low power sensors would be the answer.

There were also mechanical constraints when using current sensors for the robot arm. Firstly, The sensor had to be compact and fit within the limited space available in the robot arm. That is why small form factor sensors had to be selected so they can be easily integrated into the design. Furthermore, proper mounting and alignment are essential for accurate current measurement. It was important to implement precise mounting techniques and ensure that the sensor is correctly aligned with the current-carrying conductor.

Moreover, there were performance restrictions that needed to be thought about as well. For starters, The sensor had to be sensitive enough to detect small current changes accurately. Choosing sensors with high sensitivity and ensuring proper calibration were two critical steps to solve the problem. Additionally, the sensor should have minimal offset and drift over time and temperature variations. Because of this, high-quality sensors with low offset and drift characteristics were chosen and compensation algorithms had to be implemented.

4.2 Software Standards and Constraints:

Considering the abundance and versatility of the choices between languages, libraries, and interfaces, software may not appear to have many constraints. However, time to learn and debug a program, processing speed of the hardware used, and platform requirements all restricted our capabilities for developing our needed programs. For our project, this meant being cautious of languages that could not properly run our chosen simulator, both due to a compatibility issue with the ROS platform that the processor runs, and physical space for memory.

To combat some of these constraints, there are wider-reaching standards in place that help programmers to reach unique solutions within the bounds of limited hardware. Many of these are user-driven, rather than set by any specific protocols or rules. These standards range from syntax, to document formatting, to interfacing protocols.

4.2.1 Programming Languages:

The simulator we used does not have many standards, and is compatible with very few languages. Namely, the C++ language and some of its derivatives. This constraint prevented us from using a program such as Matlab or Python without the use of an interface to talk with the simulator. To combat this constraint, our project uses ROS (Robotic Operating System), which is standardized to work with most programming languages. This means that with a few libraries or packages downloaded, we could bridge the gap between our simulator software, Gazebo, and our desired programming language. With the use of PyBullet, we were then further able to introduce a user interface into our simulated software.

Not every programming language is created equal either, as there are many which are not open-source. This limits the programmer from full control over their program, and can present problems which the programmer will not be able to solve. Functions built within proprietary sources may limit the overall capabilities of the language or software used.

Python is an open-source platform with many libraries and packages available to the user to make programming easier. The primary standards it follows are in regards to its syntax, but is otherwise uniformly compatible with most object-oriented processes, as it was built from the structure of the Java programming language. It has support for many standard data-structures and algorithms, making it accessible and easy to learn.

4.2.2 Compatibility:

All of our software had to be compatible with each other, and in this way, software standards strive to make most programs and interfaces compatible. Common standards that facilitate this compatibility are document formatting, such as .doc, .txt, and .pdf, that most programs will have the capability to export to, making file transferral possible between programs.

4.2.3 Memory and Program Size:

Memory in particular affected the speed at which data is transferred between our embedded hardware and our software, as well as restricted the size of the program we used. We used a highly capable simulator to simulate and communicate motion between our robotic arm and it, which required a large amount of memory to run. Memory size and availability will affect our processing speed, and without a sufficiently large amount of space for continuous updates and real-time computing, we would not have had the ability to run our needed programs.

4.2.4 Processing Power:

Our hardware dictated how fast and reliably we would be able to process our program. All software is constrained by the technology that is used to run it. The Raspberry Pi has a standard of manufacturing that dictates all of their models have a memory of at least 8GB, which for a processor that is dedicated to one program, was enough to bypass the constraint of processing speed.

Chapter 5: Comparison of ChatGPT with Search Engines

5.1 ChatGPT

ChatGPT, a language model developed by OpenAI, presents several advantages over traditional search engines. One of the most significant benefits is its ability to mimic an understanding of and generate conversational text based on the context provided. This capability allows for more interactive and conversational experiences, making it ideal for tasks that require nuanced understanding and detailed responses.

Firstly, ChatGPT excels in contextual understanding. Unlike search engines that provide links to webpages, ChatGPT can retain the context of a conversation, enabling it to generate more relevant and coherent responses building upon previous ones. This makes it particularly useful for complex queries or follow-up questions where a continuous dialogue is necessary. For instance, in a research setting, ChatGPT can synthesize information from multiple sources and provide summaries that are easier to comprehend. Those same resources can be later referenced and worked with again at any point in the conversation. Additionally, ChatGPT offers an interactive and conversational experience. It can engage users in a back-and-forth dialogue, clarifying doubts and offering explanations in real-time. This interactive capability is particularly beneficial in educational contexts, customer support, and virtual assistance, where users seek more than just static information.

Another advantage is ChatGPT's ability to provide comprehensive answers. Instead of merely pointing users to various sources, ChatGPT can integrate information from different sources into a cohesive response. This is advantageous for users looking for synthesized information without having to sift through multiple webpages. Furthermore, ChatGPT is highly versatile, assisting with a wide range of tasks beyond information retrieval, such as generating creative content, providing language translations, solving coding problems, and aiding in writing and editing tasks.

Despite its strengths, ChatGPT has several limitations. One significant drawback is the accuracy and reliability of its responses. Since ChatGPT generates text based on patterns in the data it was trained on, it can sometimes produce incorrect or misleading information. Unlike search engines that often link to authoritative sources, or at the very least allow the user to choose which source it finds to be authoritative, ChatGPT's responses may lack verifiability.

Moreover, ChatGPT has a static knowledge base. Its knowledge is based on the data available up to its last update, and it cannot provide real-time information or updates. This makes it less useful for queries requiring the latest data, such as current news or live events. Additionally, ChatGPT often does not provide source referencing in its responses.

This can be a drawback for academic or professional use where source verification is crucial.

Another limitation is its performance with complex queries. For highly specialized or complex queries, ChatGPT may struggle to provide accurate and detailed responses. Its performance can be inconsistent for topics requiring deep domain-specific knowledge. Lastly, ChatGPT's responses can reflect biases present in the training data, resulting in outputs that may be culturally biased or inappropriate, which is a significant consideration for applications in sensitive areas.

5.2 Search Engines

Search engines like Google and Bing are fundamental tools for information retrieval on the internet, offering several distinct advantages. One of the primary benefits is their access to vast amounts of information. Search engines index billions of web pages, providing access to an extensive range of information on virtually any topic. This vast database allows users to find specific and detailed information quickly. Moreover, search engines provide real-time updates. They continuously crawl and index new content, ensuring that users have access to the latest information. This is particularly valuable for time-sensitive queries, such as current news, stock prices, or live event updates.

Another advantage is the ability to retrieve information from authoritative sources. Search engines rank results based on various factors, including relevance and authority, helping users find credible and reliable information. This ranking system enhances the reliability of the retrieved data. Additionally, search engines support diverse formats, retrieving information in various formats such as text, images, videos, and news articles. This diversity allows users to access information in the format that best suits their needs.

Search engines also offer advanced search features, enabling users to refine their searches with filters by date, location, file type, and more. These features help users find more specific information efficiently. Despite these advantages, search engines have limitations that can impact their usability.

One major limitation is information overload. The vast amount of information available through search engines can be overwhelming, making it difficult for users to sift through numerous results to find specific information. Additionally, not all sources indexed by search engines are of high quality or reliable. Users must critically evaluate the credibility of the sources, which can be time-consuming and challenging, and ultimately a skill that takes developing.

Another drawback is the presence of ads and sponsored content. Search engines often display ads and sponsored content at the top of search results, which can detract from the user experience and make it harder to find organic, non-promoted content. Privacy concerns also arise with search engines, as they track user behavior to improve search results and target ads. This data collection raises concerns about the use of users' search history and personal information for advertising purposes. Furthermore, companies like Google have been criticized for providing irrelevant or superfluous information in search results. Recently, Google's AI Gemini has been noted for answering questions immediately at the top of the screen, but many instances show that it provides incorrect or low-quality information [8][9].

Lastly, for complex queries, search engines may not always provide the most relevant results. Users may need to refine their searches multiple times or visit several web pages to gather comprehensive information. Ultimately search engines provide the user with near-infinite access to information, but the user must be skilled enough to retrieve that information and implement it effectively for it to be of any use.

5.3 Practical Applications and Experiences

In the context of a Senior Design project, both ChatGPT and search engines have played significant roles in enhancing the learning experience. For instance, when researching components for our project, ChatGPT provided synthesized information and explanations about different technologies, helping us make informed decisions without going through multiple sources. However, for real-time updates on component availability and prices, search engines proved more effective. Another example is troubleshooting technical issues. ChatGPT could generate step-by-step solutions and explanations, offering a quick understanding of problems. In contrast, search engines directed us to forums and documentation where we found detailed discussions and community support.

Moreover, while writing and editing reports, ChatGPT assisted in formatting references and refining drafts, making the process more efficient. On the other hand, search engines were invaluable for sourcing references and verifying information, ensuring the accuracy and credibility of our work. By comparing the advantages and limitations of ChatGPT and search engines, it becomes clear that each has unique strengths and weaknesses. ChatGPT excels in interactive, context-aware conversations, while search engines provide access to a vast and continuously updated repository of information. The choice between using ChatGPT or a search engine depends on the specific needs of the user and the nature of the query, but a combined use of both would greatly benefit any and all applications.

5.4 Pros and Cons:

Platforms such as ChatGPT, Google, and Bing make research for any project much faster. They provide results based on our specified needs, which include educational articles, theses from graduates with PhDs, schematics, and textbooks. While most of these could be found on paper at a local university's library, these platforms save us the time of searching by hand for the specific bits of information that we need.

This information is also up-to-date, providing us with the newest information and discoveries. This may mean the newest models for components we are searching for, as well as new scientific discoveries that we may be able to apply to our project to make development easier.

While in most cases, using these methods is far faster and more reliable than textbooks and paper documents printed decades ago, the biggest drawback is ensuring that the information gathered from these sources is factual. Where printed books and non-internet sources have the advantage is often in peer-review. Anyone can create a website, a forum, a page, or a social media post that ChatGPT or a search engine may pull information from. While it is possible that the sources found via these methods are factual and from trusted sources, it is much harder to verify this. Therefore, when using these methods, it requires the added step of finding at least three additional sources that confirm the information found.

Using search engines are far more reliable than ChatGPT, and a big drawback of the ChatGPT, and any generative AI platform, is the program's capability to lie. The program does not know what is true and what is false, and is making statistical "guesses" with the information it knows, which can often lead you to "learning" incorrect details about a subject. These "lies" may be peppered in between real and verifiable information, which can make parsing the true facts a bit tricky. Any information received from ChatGPT needs to be verified with either a paper source, or one found from other trusted sources on the internet, such as a government or scientific website that processes articles and papers that it publishes. Despite this major drawback, ChatGPT is an amazing tool to use in terms of efficiency, as it beats out all other methods by saving the user an immense amount of time. It can help to direct the user to the correct terminology or search they should use to find additional information from textbooks, scientific journals, and publishers online.

5.5 Examples:

An example where we used ChatGPT to help with research is for the gate driver section. We asked the AI for advice on how to best select a gate driver for our design. We started by asking basic overarching questions about the definition of and purpose of gate drivers. The answers served as decent summaries and were essentially true. A concise summary like this is useful because it helps with knowledge retention and helps streamline the learning process, quickly targeting what information needs to be researched next. After getting a summarization of the function of the gate driver, we were able to start a more direct line of questioning to help determine which specifications needed to be researched for the gate driver selection. To determine what kind of amplification we needed we asked the language model what logic-level voltage our preferred microcontroller used. The AI reported it as 3.3 V, and after checking with the product page this was confirmed as true. Using that as a jumping-point, we asked what other specifications we would need to select our gate driver and found a formula for drive-current based on the gate charge and switching time. We then checked the datasheet of the MOSFET, which gave us the value of the gate charge, but not the switching time. When we asked ChatGPT how to find switching time, it said that switching time is the summation of the rising time, falling time, turn-on delay, and turn-off delay times. All of these times were in the datasheet, so we were able to use them to calculate a drive current. The drive voltage was already in the datasheet so we now had the logic level, drive current, and drive voltage we needed to select a compatible gate driver for our desired parts. When asked about some of the amplification behavior, the AI volunteered information about how the voltage can be changed, but more interestingly it offered suggestions of parts that matched our specifications. This, however, revealed a limitation in the AI's ability to understand a complex and multi-part task, such as this one, as when checked on these parts on the product page they were not actually compatible with the 3.3 V logic level.

Another application in the gate driver section was helping us to interpret the datasheet of our preferred gate driver and better understand its function. It concisely explained how the gate driver is able to output a specific voltage based on a voltage supplied to it. Further reading of the datasheet confirmed this summary was accurate.

Demonstrated in Appendix A [20] is a conversation between ChatGPT and one of our group members, asking ChatGPT to explain the differences between different components we had looked at using for our project.

In this example ChatGPT assumed CUI - AMT10 and the rev through bore encoder are the same product and just made up info even though they are separate products. This example shows how unreliable ChatGPT is during the research phase, yet helpful when trying to find which direction we need to go for our project.

Chapter 6: Hardware Design

6.1 Joystick Subsystem PCB

6.1.1 Joystick Modules:

The joystick module is designed to provide control inputs for the robotic arm through a custom PCB. The schematic below shows two analog joysticks connected to the PCB, interfacing with the microcontroller unit (MCU) pins. The power connections are essential for the operation of the joysticks: both joysticks are connected to a +5V power supply, ensuring they have the necessary voltage to function. The ground (GND) connections are similarly made to establish a common reference point for the circuit.

Analog input connections are set up as follows: Joystick U2's vertical Y-axis is connected to the V1, V2, and V3 pins, while the horizontal X-axis is connected to the H1, H2, and H3 pins. Specifically, V1 (V+) and H1 (H+) are connected to the +5V power supply, V3 (V-) and H3 (H-) are grounded, and V2 (V) and H2 (H) provide the analog input to the respective MCU pins. For Joystick U2, V2 is connected to A1 and H2 is connected to A0. Joystick U3 follows the same connections as U2, with the only difference being V2 is connected to A3 and H2 is connected to A2. These connections allow the custom PCB to accurately read the position of each joystick axis.

All V- and H- connections are grounded together, ensuring a common ground reference for the signals. Similarly, all V+ and H+ connections are powered together by the +5V node, provided by the micro usb, which we will discuss later. This setup ensures that both joysticks receive a stable power supply and share a common ground, which is crucial for maintaining signal integrity and accurate readings.

The schematic also includes SEL+ and SEL- lines for the joystick buttons which we will not be using or connecting in our project. Also included are SHIELD lines that help protect the analog signals from electromagnetic interference, ensuring reliable readings.

By setting up these connections, the joystick module is integrated into the control system, allowing the custom PCB to process input from the joysticks and translate it into control commands for the robotic arm.

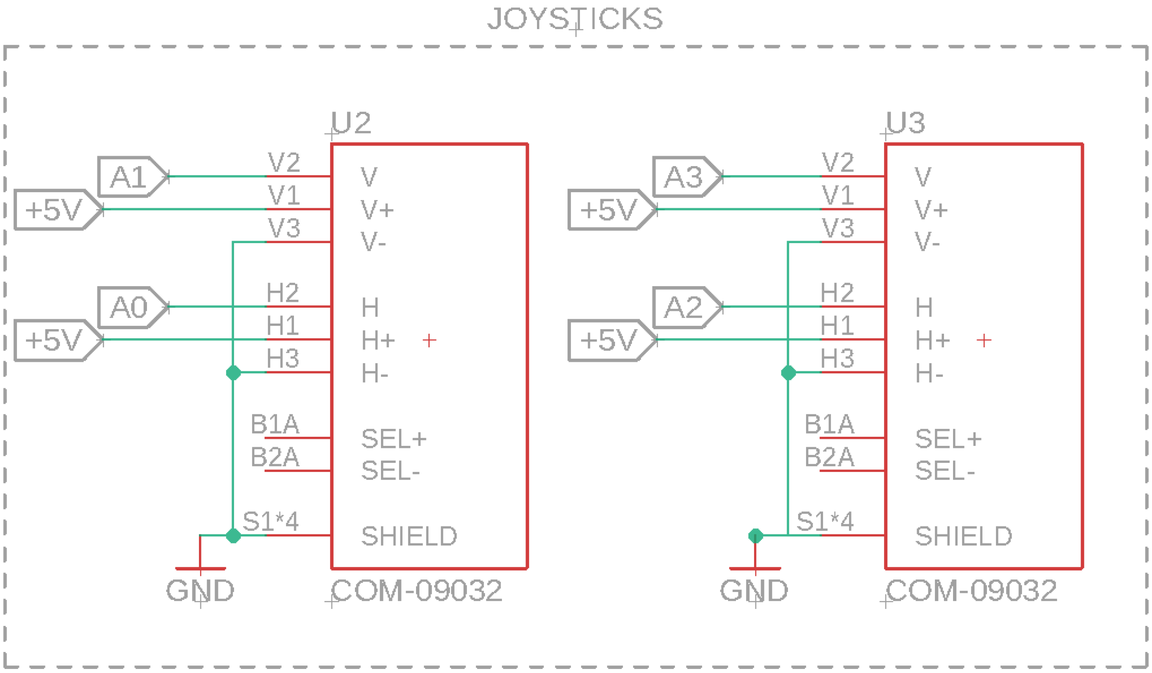


Figure 6.1.1 - Joystick Module Schematic on Fusion 360

6.1.2 Mechanical Buttons:

In the following section, we will discuss the schematic for the general-purpose buttons and reset buttons incorporated into our joystick PCB subsystem. These buttons play a crucial role in providing additional control functionalities, including emergency stops and system resets. The design and connections of these buttons will be detailed to illustrate their integration into the overall control system.

6.1.2-1 General Purpose Buttons:

The schematic shown below illustrates the connections for the general-purpose buttons incorporated into the joystick PCB subsystem. These buttons are 4-pin tactile switches, although only two pins are active at any given time, with the other two pins providing mechanical stability on the PCB. Each button is configured in a pull-down orientation:

one end of the button is connected to the +5V supply line, while the other end is connected to digital input pins D0, D1, and D2 on the MCU, respectively. Additionally, each button's connection to the ground is facilitated through a 10kΩ pull-down resistor, ensuring the input pin reads low (0) when the button is not pressed out of preference.

These general-purpose buttons are versatile and can be used for various control functions. One will act as an emergency stop button, another as a record and replay button, and the third as an end effector activator button. This setup ensures that functions can be easily and reliably controlled by the user through simple, robust, and effective button presses.

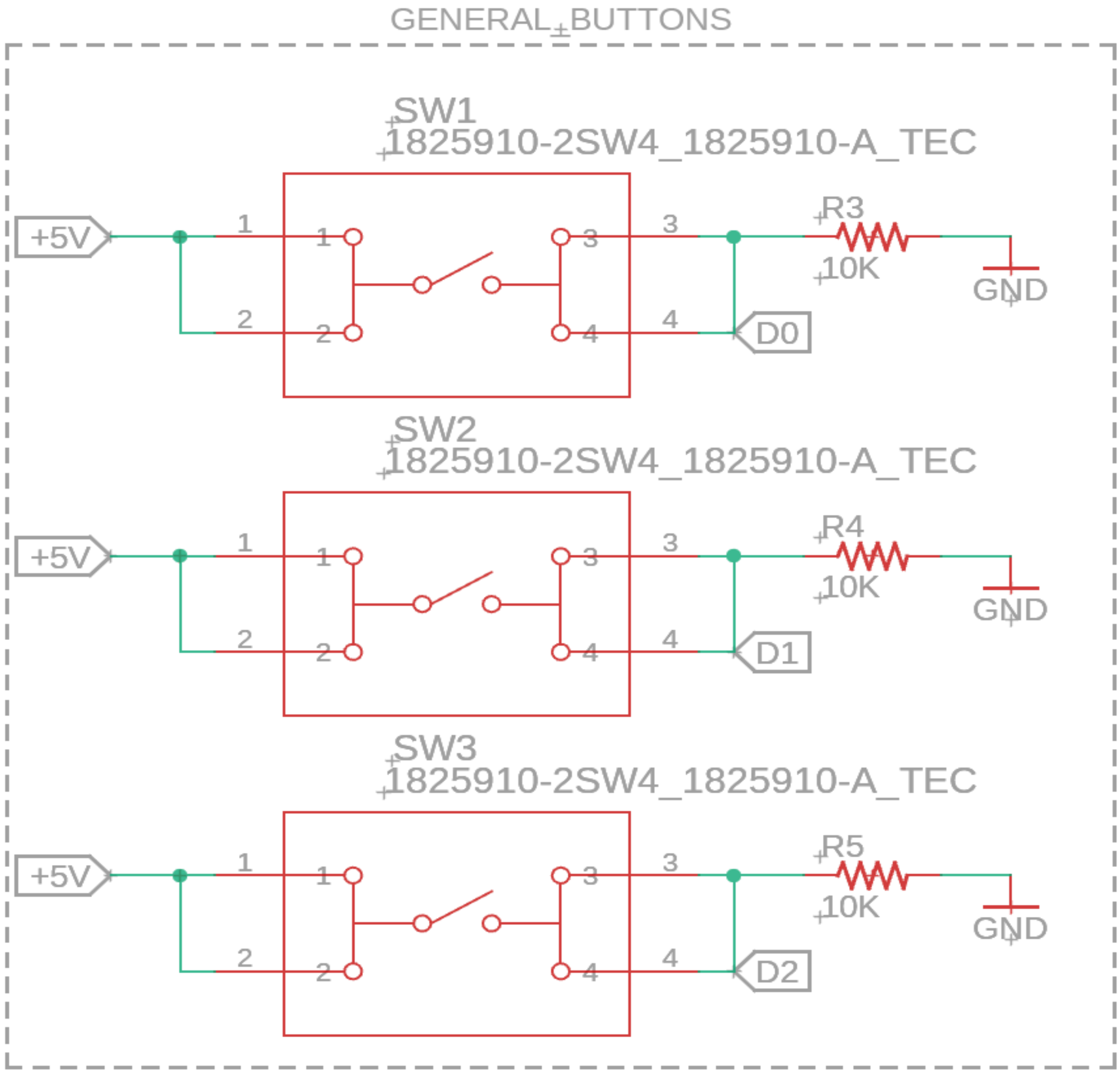

GENERAL_BUTTONS
SW1
1825910-2SW4_1825910-A_TEC
SW2
1825910-2SW4_1825910-A_TEC
SW3
1825910-2SW4_1825910-A_TEC
+5V
+5V
+5V
1
2
3
4
R3
R4
R5
10K
10K
10K
GND
GND
GND
D0
D1
D2

Figure 6.1.2-1 - General Input Buttons Schematic on Fusion 360

6.1.2-2 Reset Button:

The schematic below illustrates the reset button circuitry integrated into the joystick PCB subsystem. Unlike the general-purpose buttons, the reset button is configured as an active low input. This means that when the button is pressed, it creates a low signal (ground) that triggers the reset function on the MCU. One end of the button is connected to the ground, while the other end is connected directly to the reset pin on the microcontroller unit (MCU). This configuration allows the button to reset the MCU when pressed, ensuring the system can be quickly and efficiently restarted if necessary.

In addition to the direct connection to the reset pin, the reset button circuitry includes a connection to the +5V supply line. This connection features a 1N4148W diode and a 10kΩ resistor in parallel between the reset node and the +5V supply. The diode helps to protect the circuit by preventing reverse voltage, while the resistor ensures a stable voltage supply, reducing the risk of accidental resets due to voltage fluctuations. Furthermore, two 0.1µF decoupling capacitors are placed in parallel between the +5V supply and ground. These capacitors help to filter out any noise or transient voltages, providing additional protection to the circuit and ensuring consistent operation. The reset button is a crucial component for maintaining the stability and reliability of the joystick PCB subsystem. By allowing the system to be reset quickly and safely, this button helps to prevent and mitigate potential issues that could arise during operation.

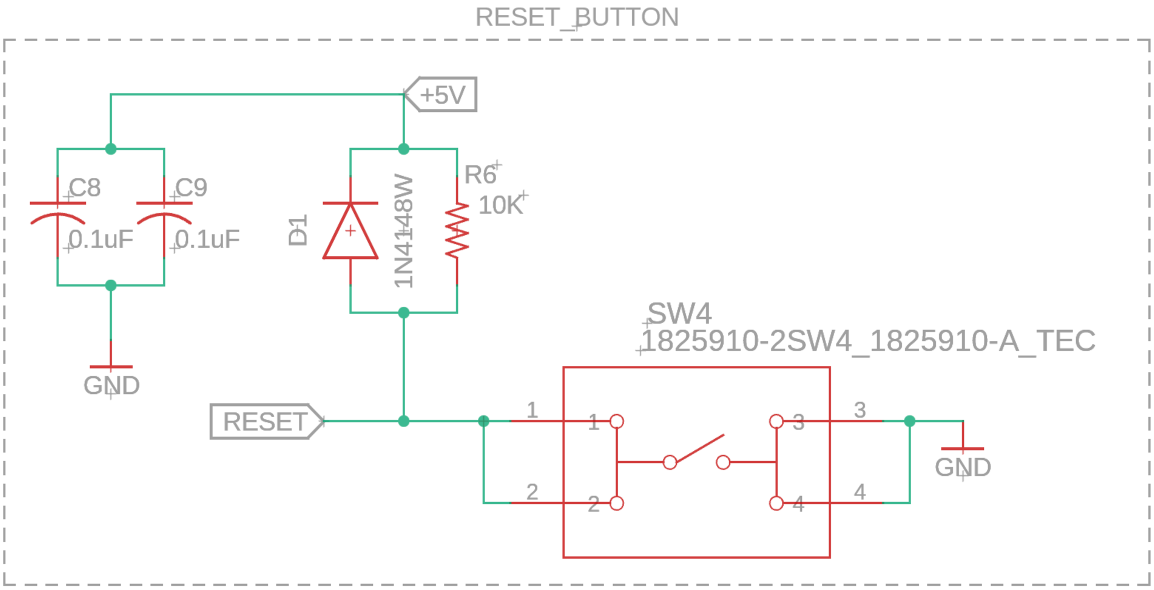


Figure 6.1.2-2 - Reset Button Schematic on Fusion 360

6.1.3 Micro Usb Connection:

The Micro USB connection is a crucial component of our joystick PCB subsystem, facilitating both power delivery and data communication. The Micro USB connector has five pins, each serving a specific function in the overall design.

Pin 1, labeled VUSB, is responsible for providing power to the entire PCB. This pin is connected in series with a fuse (F1), which offers protection against overcurrent scenarios with a maximum of 15V and 500 amps. The +5V node on the other side of the fuse distributes power to all necessary components within the joystick subsystem PCB. This connection ensures that the PCB receives a stable and protected power supply, crucial for reliable operation.

Pin 2, labeled D-, serves as the negative data line. According to the MCU datasheet, this pin is to be connected in series with a 22Ω resistor (R1). The node on the other side of the resistor is labeled RD- to differentiate it as the data line after the resistor, which is directly connected to the D- pin on the MCU. This configuration helps manage data transmission integrity by matching the impedance and reducing potential signal reflections. Pin 3, labeled D+, functions similarly to D- but serves as the positive data line. It is connected in series with a 22Ω resistor (R2), and the node on the other side is labeled RD+ to indicate the data line after the resistor, which is directly connected to the D+ pin on the MCU. This parallel configuration ensures that both data lines are properly managed for optimal data transmission.

Pin 4, labeled USBID, is not utilized or connected in our design. This pin is typically used for identifying the connected device type in some USB configurations, but it is not necessary for our joystick subsystem's functionality.

Pin 5, labeled UGND, is connected to the UGND pin of the MCU. Additionally, it is connected to the overall ground of the system. All grounds being connected is not typical when working with many analog and digital components together, but for our simple use case it was perfect and simpler.

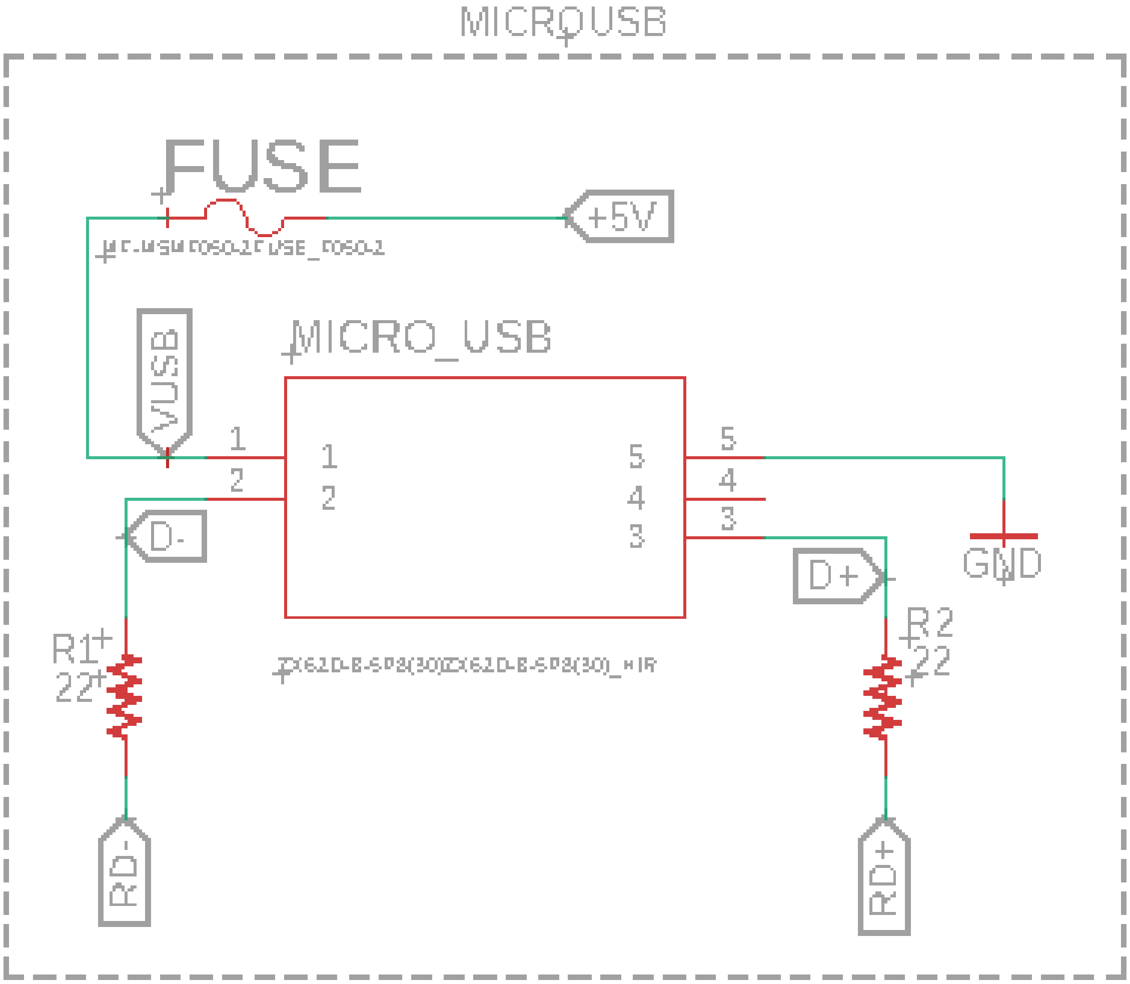


Figure 6.1.3-1 - Micro Usb 5-pin Schematic on Fusion 360

6.1.4 16MHz Crystal Oscillator:

An included 16MHz crystal oscillator plays a critical role in providing a stable clock signal for the ATMEGA32U4 microcontroller. This clock signal is essential for the timing and synchronization of the microcontroller's operations, including the execution of instructions and communication protocols. The schematic shows the crystal oscillator connected between the XTAL1 and XTAL2 pins of the microcontroller.

A 1MΩ resistor (R8) is placed in parallel with the crystal oscillator. This resistor helps start the oscillation by providing a small bias current, which ensures that the crystal can begin vibrating at its specified frequency of 16MHz. Additionally, two 20pF capacitors (C6 and C7) are connected from each end of the crystal to ground. These capacitors are specified by the crystal's datasheet and are crucial for the oscillator's proper operation, as they stabilize the oscillation by filtering out any noise and ensuring a clean and stable frequency output.

By connecting the crystal oscillator in this manner, we ensure that the microcontroller has a reliable and precise clock source, which is necessary for the accurate timing of its internal processes and external communications. This setup allows the microcontroller to perform consistently at its intended speed, enhancing the overall performance and reliability of our joystick controller system.

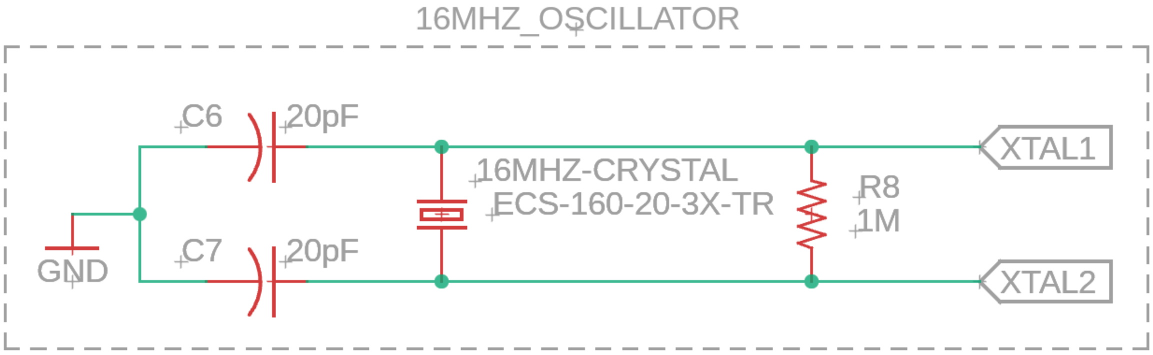


Figure 6.1.4 - 16MHz Crystal Oscillator Schematic on Fusion 360

6.1.5 Decoupling Capacitors and LEDs:

The power decoupling capacitors are critical components for ensuring the stable operation of the ATMEGA32U4 microcontroller. They help filter out noise and provide a clean and stable voltage supply, which is essential for the proper functioning of the microcontroller and other sensitive components. In the power decoupling schematic below, we have several decoupling capacitors strategically placed to achieve this purpose. Between the +5V and ground, a 10µF capacitor (C1) and a 0.1µF capacitor (C2) are placed in parallel. The 10µF capacitor helps to smooth out low-frequency noise and provide bulk capacitance, while the 0.1µF capacitor is effective at filtering out high-frequency noise. Together, they ensure a stable supply voltage close to the microusb, which is crucial for its reliable power delivery.

A 0.1µF capacitor (C3) is placed between VCC and ground, ensuring stability for the microcontroller's main power supply. Additionally, a 0.1µF capacitor (C4) is placed between UCAP and UGND. UCAP is the internal USB regulator capacitor, and having a 0.1µF capacitor here ensures that the USB transceiver receives a stable voltage, which is vital for maintaining reliable USB communication. The UGND is the ground specific to the USB circuitry, and this capacitor helps to maintain a clean power supply for the USB functions.

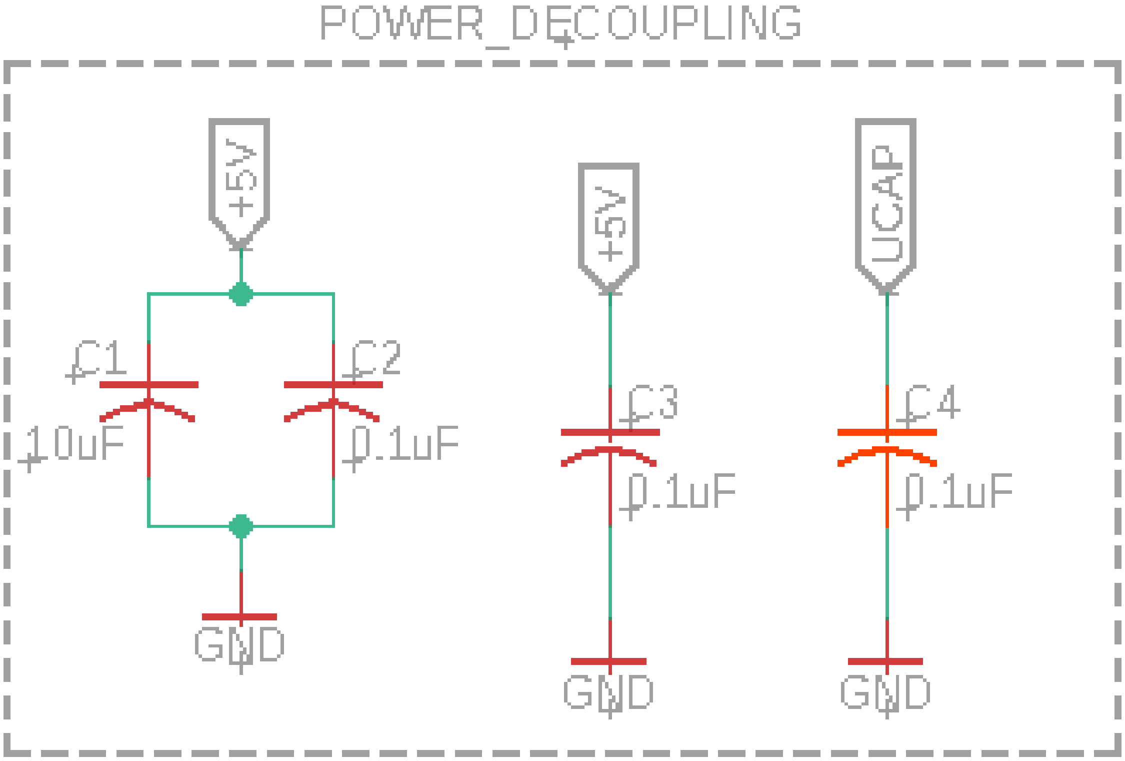


Figure 6.1.5-1 - Power Decoupling Capacitors Schematic on Fusion 360

Additionally, for convenience purposes, we included a power-on and a GPIO LED seen in the schematic shown below connected in series with a 1kΩ resistor (R9 & R10) between the +5V power supply, after the Fuse, and ground. These LEDs serve as an indicator to verify that the microcontroller is receiving power that isn't over the Fuses limit. When the system is powered on, the power LED will light up blue, providing a visual confirmation that the microcontroller and other components are properly energized, and then the GPIO LED can be programmed to ensure the MCU is working.

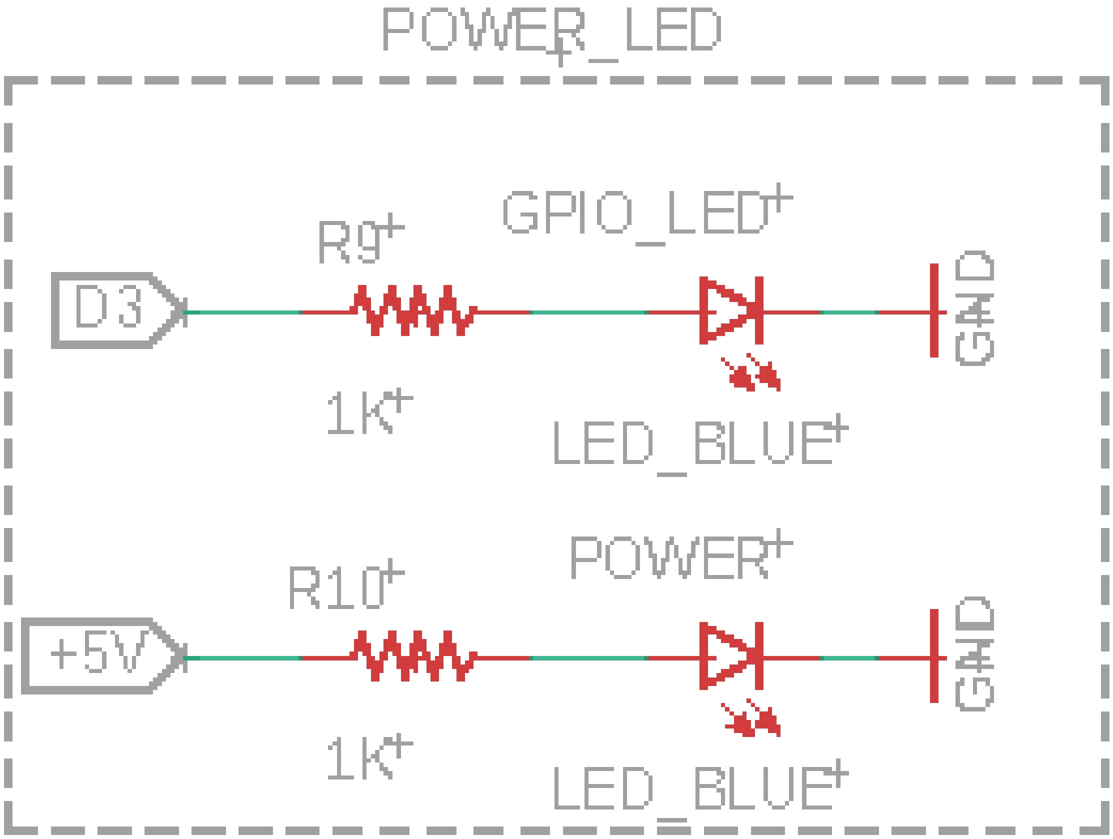


Figure 6.1.5-2 - LED Schematic on Fusion 360

6.1.6 MCU Connections:

In the first part of the ATMEGA32U4 microcontroller schematic, several important connections are made to ensure the proper functionality of the joystick PCB subsystem. The analog pins A0 to A3 are connected to the middle pins of their respective joystick potentiometers, allowing for the reading of analog voltage values corresponding to the joystick positions. The digital inputs D0 to D2 are connected to their respective general-purpose buttons, enabling digital read functionality for these input buttons.

The reset pin is connected to the reset node on its button, providing the ability to reset the microcontroller as needed. The D- and D+ pins of the microcontroller are connected to RD- and RD+ from the micro USB, facilitating USB communication for data transfer and power supply. The XTAL1 and XTAL2 pins are connected to the 16MHz crystal oscillator that was discussed earlier, to provide the necessary clock signal for the microcontroller's operation.

The AREF pin is connected to a decoupling capacitor to ground, providing a stable analog reference voltage for the ADC (Analog-to-Digital Converter) to ensure accurate analog readings. Additionally, the HWB pin is connected to a 10k resistor in series with ground, enabling the USB bootloader functionality when the reset button is pressed, allowing for easy firmware updates and programming via USB. Additionally, the microcontroller schematic includes connections for the SPI interface with labeled pins MISO, MOSI, and SCK. These connections facilitate serial communication, allowing for efficient data exchange with other peripherals or devices as needed. We will discuss this more in the ICSP header section.

In summary, this section of the schematic ensures that the microcontroller has all the necessary connections for analog and digital input readings, USB and ICSP communication, reset functionality, and a stable clock signal, enabling it to effectively manage the joystick and button inputs for the joystick PCB subsystem.

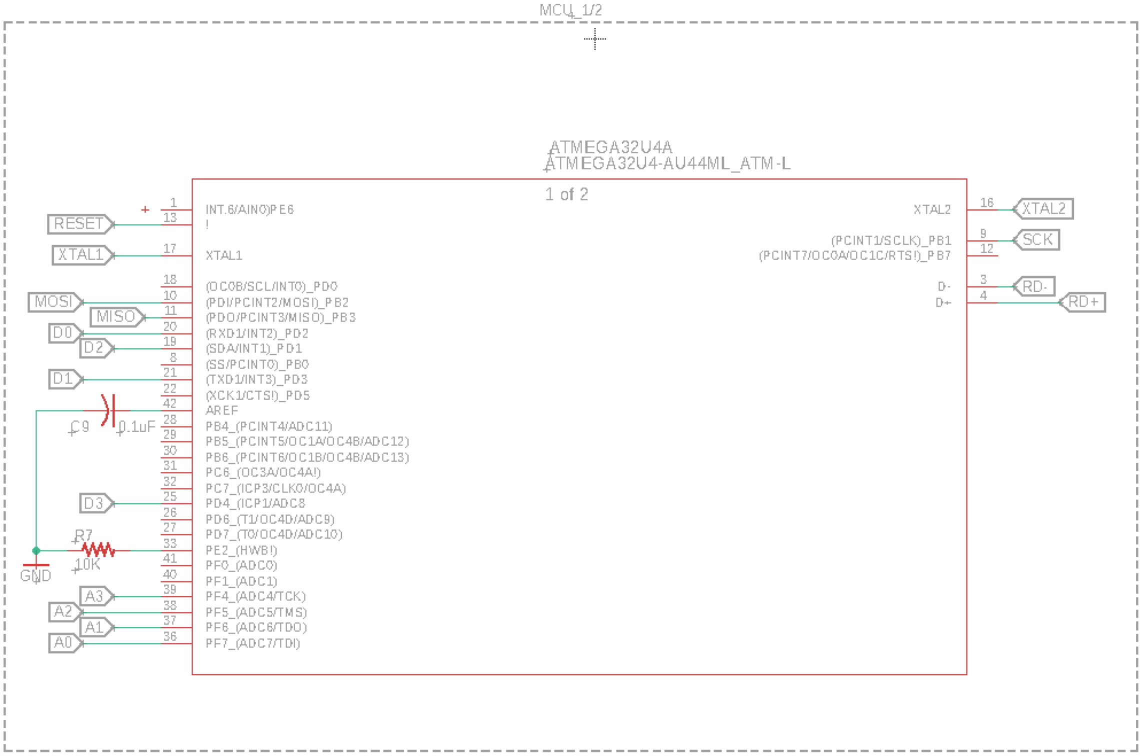


Figure 6.1.6-1 - ATMEGA32U4 Schematic 1 of 2 on Fusion 360

In the second part of the ATMEGA32U4 microcontroller schematic, the focus is on the power and ground connections essential for the microcontroller's operation. All VCC, VCC_2, VBUS, and UVCC pins are interconnected and linked to the +5V supply provided by the micro USB, ensuring a consistent power supply to the microcontroller. This connection is crucial for the stable operation of the microcontroller and all its internal components. Similarly, all ground pins (GND, GND_2, GND_3, GND_4) are interconnected to the same ground to maintain a common reference point for the circuit, which is vital for preventing ground loops and ensuring accurate signal readings. The UGND pin, which is specific to the USB ground, is separately connected to the UGND on the micro USB to maintain the integrity of the USB communication. Overall, these connections ensure that the microcontroller receives a stable and noise-free power supply, critical for the reliable performance of both its digital and analog functionalities.

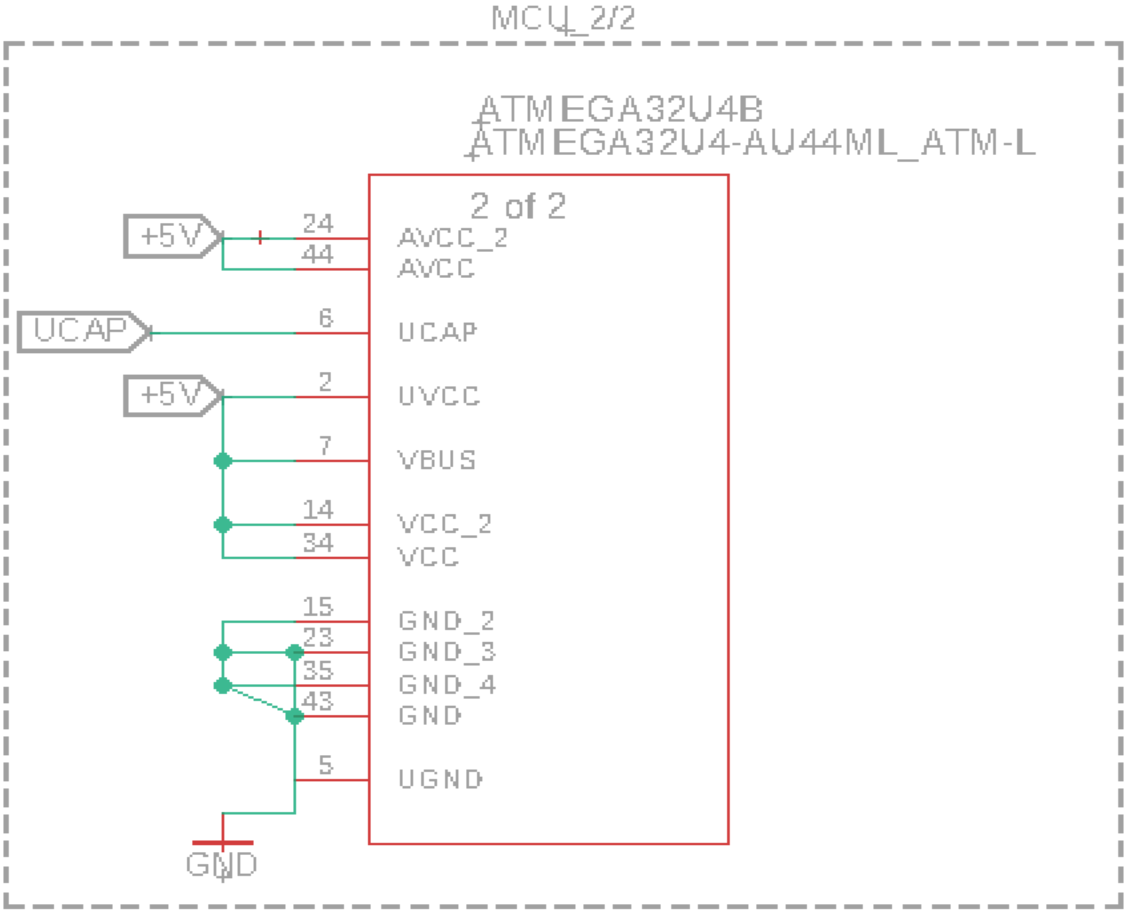


Figure 6.1.6-2 - ATMEGA32U4 Schematic 2 of 2 on Fusion 360

Finally, the ICSP (In-Circuit Serial Programming) header schematic shown below includes a 6-pin header that provides a connection for programming and debugging the microcontroller. The pins are connected as follows: pin 1 to MISO, pin 2 to +5V, pin 3 to SCK, pin 4 to MOSI, pin 5 to RESET, and pin 6 to GND. This header is initially used to burn the bootloader on the device using an Arduino, which is a necessary step for enabling subsequent programming via the micro USB port. Once the bootloader is burned, the board can be conveniently programmed through the micro USB connection. The ICSP header is essential for initial development with the custom PCB but will not be needed for normal operation.

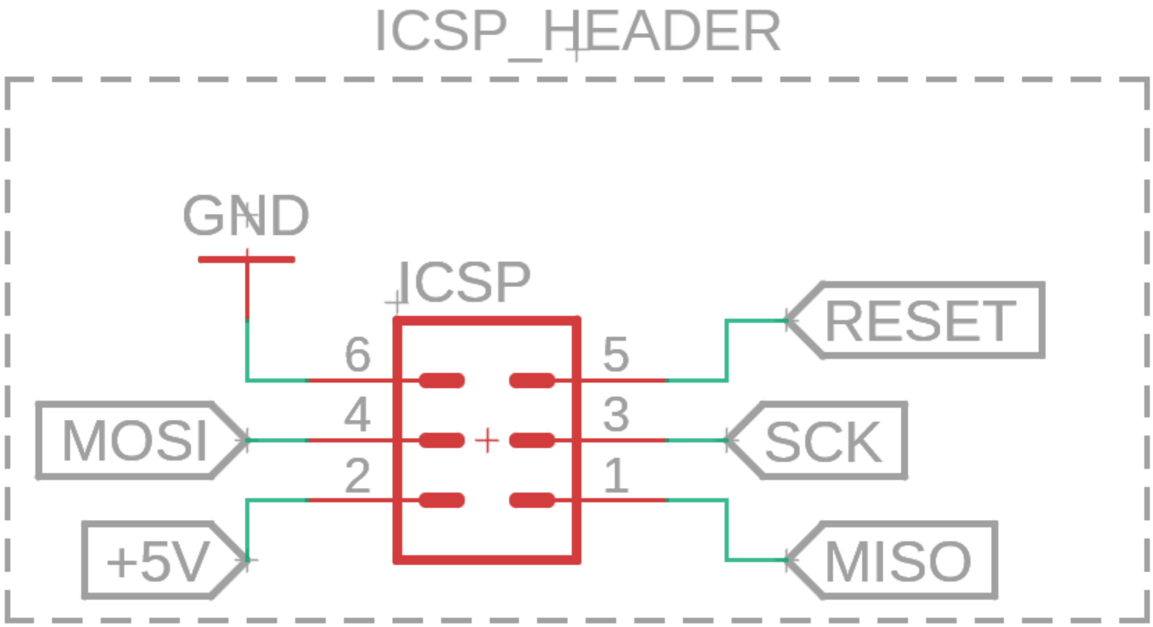


Figure 6.1.6-3 - ICSP Header Schematic on Fusion 360

6.2 Motor Subsystem PCB

The motor subsystem is one of the most complex parts of our hardware section. It is best understood when divided into multiple components. The motor control PCB includes several key schematics: Dual H-Bridge schematic, gate driver schematic, position sensor schematic, current sensor schematic, CAN bus schematic, and the RP2040 MCU schematic. Each of these components plays a crucial role in ensuring precise and efficient motor control.

6.2.1 Position Sensor

Position sensors allow for accurate detection of the motor shaft's angle, which is essential for implementing closed-loop control systems and enhancing the accuracy of the motor's positioning capabilities. It provides the microcontroller direct measurements of angle and allows the derivation of speed and provides a way to measure error in the feedback loop.

Our selected position sensor communicates over SPI. The connections made are as follows: Pin 5 (MOSI) is connected to GPIO1F1. Pin 6 (MISO) is connected to GPIO4F1. Pin 7 (CS) is connected to GPIO4F1. Pin 8 (GND) is connected to ground (GND). Pin 9 (PWM) is connected to GPIO2F1. Pin 12 (SCLK) is connected to GPIO2F1. Pin 13 (VDD) is connected to a 3.3V power supply. This position sensor will be placed on the motor PCB alongside other essential components, including the current sensor, gate driver, and MCU. The integration of these components on a single PCB ensures efficient communication and coordination between them, enhancing the overall functionality and reliability of the motor control system. The compact and efficient layout of the motor PCB will facilitate easy assembly and maintenance, contributing to the robustness of the system.

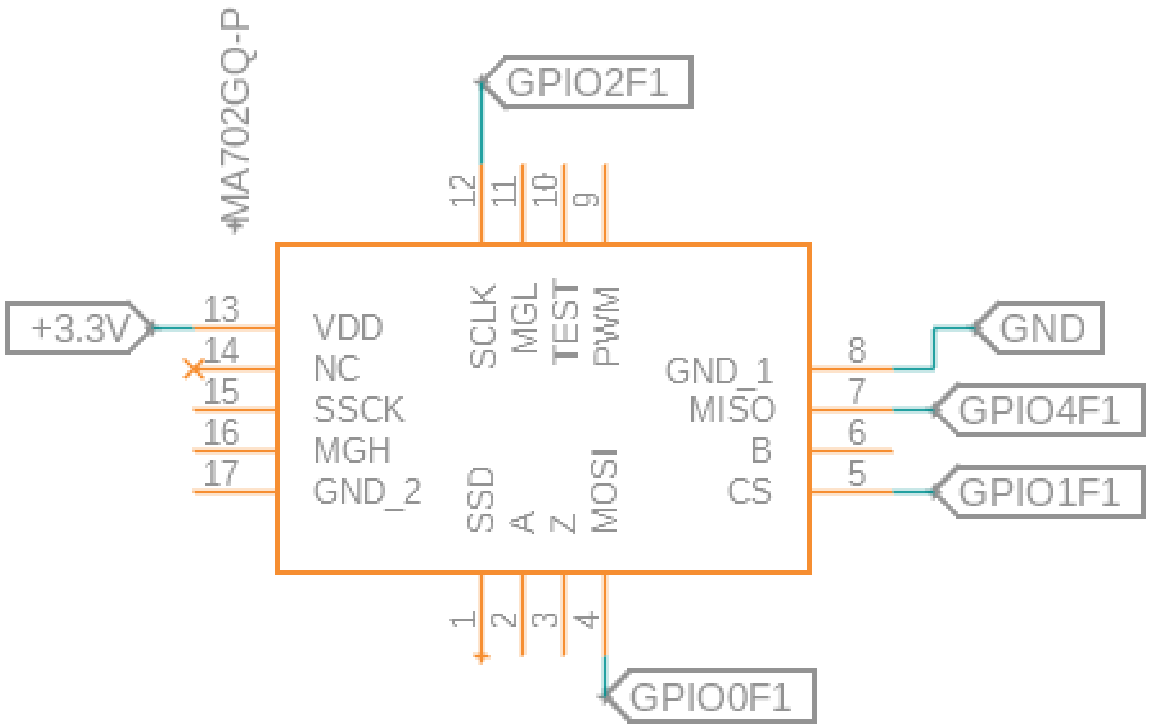


Figure 6.2.1 - Position Sensor (Rotary Encoder) Schematic on Fusion 360

6.2.2 Current Sensor

The current sensor is used to monitor the current running through the four phases of each motor, outputting an analog signal to the microcontroller. The four phases of the motor are connected to pins one, two, three, and four of the current sensor. Pin five is connected to the system ground. Pin 6 is a bandwidth selector pin and will adapt to either 20 kHz or 80 kHz based on the device connected to it. Pin 7 is the analog output signal that contains the current information the sensor was designed to measure. This pin will be connected back to the microcontroller. The final pin, pin 8 serves as a power input pin that powers the sensor. In this case it's connected to a 5 V source from our power distribution network. Once implemented this allows the microcontroller to send information about the current back to the main computer. This data can be used to make torque calculations in real time which is essential for our robot being able to make decisions based on the torque required to lift an object.

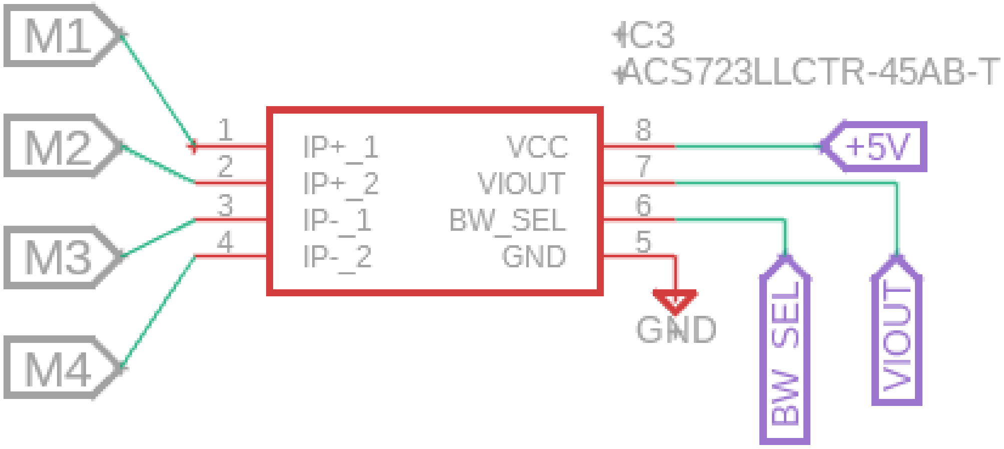


Figure 6.2.2 - Current Sensor Schematic on Fusion 360

6.2.3 CAN Bus

The CAN bus connections were constructed using our own board, built with MCP2551 transceiver and MCP2515 controllers. These chips required a quartz clock of at least 8,000 MHz, and a board for pins. We used a 16MHz external parallel crystal oscillator for all MCP2515 chips, to increase the speed of data sent. These boards ended up being similar to the amazon development boards we used for testing, which use all the same components except for the transceiver and clock speed. The development boards were built with a 5V arduino in mind, meanwhile we tweaked this design to be more compatible with a 3.3V device, the ESP32. The microprocessor was more tolerant of the voltage levels produced with these components in mind, especially once we separated the voltage lines between the 3.3V and 5V voltage-in.

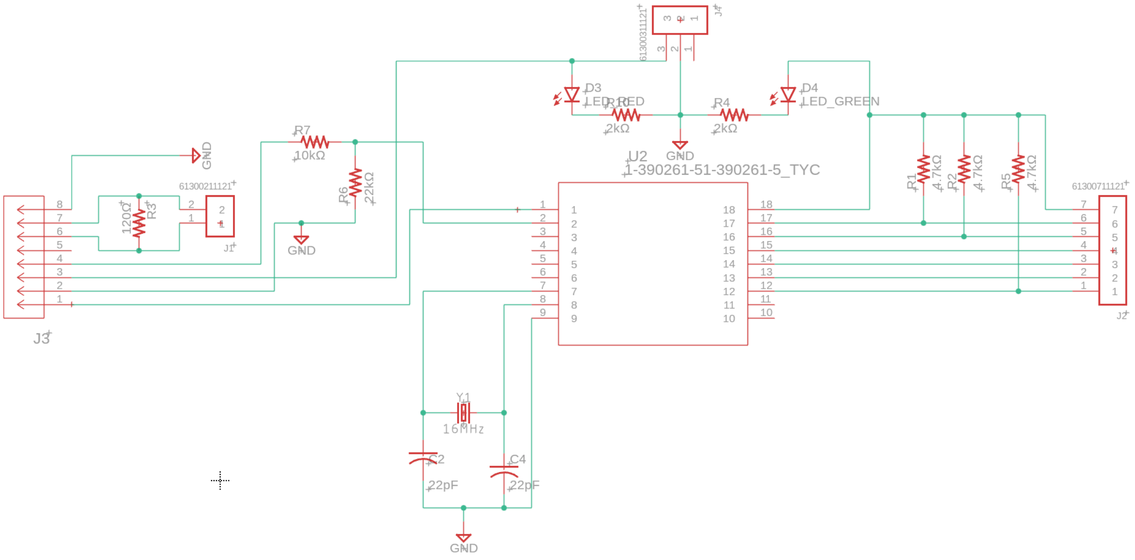


Figure 6.2.3-1 CAN Bus Schematic on Fusion 360

The CAN pins labeled SO/SI stand for the MISO/MOSI connections within the embedded system. The microcontrollers were connected to all of the SPI connections, which included SO/SI, the SCK (Clock) signal, the CS (Chip Select), and INT (Interrupt) pin.. The rest of the pins were fairly intuitive, which corresponded to the VOUT and ground of both CAN module and microcontrollers. Each microcontroller was connected to an individual CAN node, creating the network of CAN communication devices that we will use to talk not only with the motors, but with each microcontroller in real time.

The Raspberry Pi acted as the host that ran communications between tele-operative input and ESP32s, printing out and reading the CAN communications. It also handled the simulation and mapping of encoder and current sensor data used by both the motors on the ESP32 side, and the movement of our simulator. It was in charge of initial calculations, as well as sending our first message to the CAN bus nodes so that our motors could function.

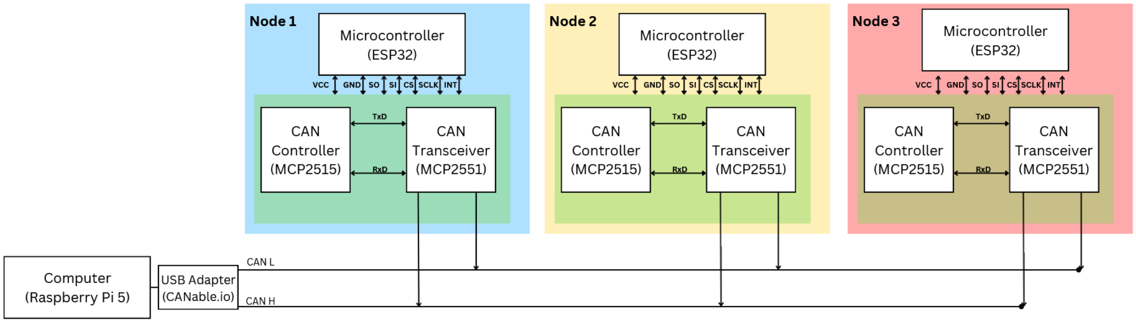


Figure 6.2.3-2 CAN Bus Node Connections

6.2.4 Microcontroller Connections

The ESP32 microcontrollers rely on being connected to our CAN boards in order to send information between each other. However, these microcontrollers are not 5V tolerant, so we used our custom MCU PCB board, coupled with a logic-level shifter, to shift the voltage from 5V to 3.3V, preventing our microcontroller from experiencing a burnout. Each microcontroller was connected to their own CAN bus to create three nodes that the signals passed between. These connections were attached to our motors, getting signals from the position encoders to determine how much movement was needed to drive each axis.

Each microcontroller belonged to a single node, connected centrally to the Raspberry Pi host. The CAN controller and transceiver components created the CAN architecture, allowing us to send messages to and from other CAN Tx/Rx connections. The microcontrollers were then connected to our motor drivers, as they were integral for shifting the motor position. The microcontrollers passed on variables to adjust the forward and backward movement of our motors, working in a closed-loop system that was capable of adjusting our motors back to an initialized "zero" state.

Below is a schematic of the microcontrollers and its connections as they fit with each peripheral device.

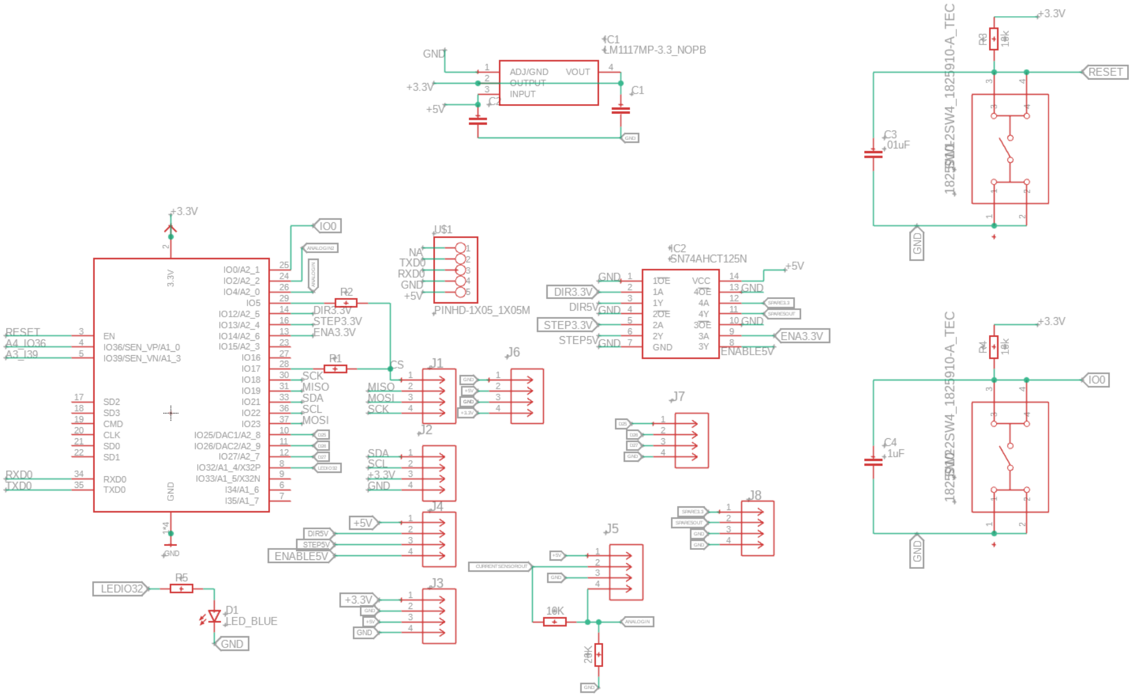


Figure 6.2.4 - Microcontroller Connections Schematic on Fusion 360

The current sensor is connected to the microcontroller via its bandwidth selector connecting to GPIO6 and VIOUT connecting to GPIO7. The bandwidth sensor will allow the current sensor to sync up its bandwidth to more closely match the microcontroller. VIOUT is the analog output of the current sensor that will allow the robot to gain real time information about the torque of the motor and allow it to make decisions based on that information.

The rotary encoder has connections at pins GPIO0, GPIO1, GPIO2, GPIO4, as well as the 3.3 V source and the ground. These connections allow for the rotary encoder to receive the power it needs to operate as a position sensor and communicate that information back to the microcontroller. That information can then be further passed through the CAN bus to the robot to allow the robot to make decisions based on the rotational position of the stepper motor. This is crucial for the actual movements of the arm as this will make the kinematics equations that help the robot figure out how to move from one position to another possible.

6.3 Raspberry Pi/Computer Subsystem

The Raspberry Pi used for our system was a Raspberry Pi 5 microprocessor, compatible with the ESP32 microcontrollers. These were connected to our CAN bus system to talk with microcontrollers, which were connected to each other via the CAN-L and CAN-H pins. A CAN-to-USB adapter was used to connect all lines, and make our CAN architecture compatible with our Raspberry Pi. When slotted in, it automatically showed up as a CAN network, allowing us to use all nodes in the system simultaneously and without strain.

The power system was also connected to the Raspberry Pi system via a microUSB to produce at least 5W of power. Additionally, the Raspberry Pi served as the central hub for processing data and coordinating communication between various subsystems, ensuring seamless operation and control of the entire system.

The Raspberry Pi was the central host for all communication between our main components, including our joystick subsystem. It was the main driver for simulating initial movement, as well as taking feedback to readjust any movement during processing an operation.

The schematic for this part can be found along with the pinout in the official documentation. We did not build this part, and only used it for simulations and powering our CAN network.

There was no pin connection to any device, as the CAN-to-USB device made it possible to create our own wired CAN-H and CAN-L connections, which sent the proper data across all CAN nodes seamlessly. This setup ensured reliable communication and power delivery for the entire control system.

6.4 Power Distribution Subsystem

Since the robot arm was powered by the switching power supply, the components within the system required proper voltage regulation and distribution. The output voltage of the power supply ranged from 21.6V to 28.8V, so 24V was chosen as the standard input voltage for most parts of the robot arm. The NEMA23 stepper motors, which operated at a supply voltage of 24V, were directly connected to the 24V output of the power supply. This connection was implemented using appropriately rated wires and connectors to handle the current draw of each motor safely and efficiently. However, several components of the arm required different voltage levels to function properly. To address this, three voltage converters were used to adjust the voltage, either by stepping it up or down to the required levels for each component. These converters utilized a combination of resistors, capacitors, inductors, and integrated Pulse Width Modulation (PWM) circuits to ensure accurate and stable voltage regulation.

To operate the AS5600 magnetic encoder and CAN Bus controller within the robot arm, a supply voltage of 3.3V was required. To achieve this, a 24V to 3.3V DC-DC buck regulator was utilized. As seen in the figure below, the buck regulator selected was the TPS54308DDCR switching buck regulator The input of the buck regulator was connected to the 24V output of the power supply. The output of the DC-DC converter provided a stable 3.3V, ensuring reliable operation of the encoders and CAN Bus controllers. To accommodate the current draw of these components, properly rated wires were used to maintain efficiency and safety in the connections. The schematic for the boost regulator can be seen in the figure below.

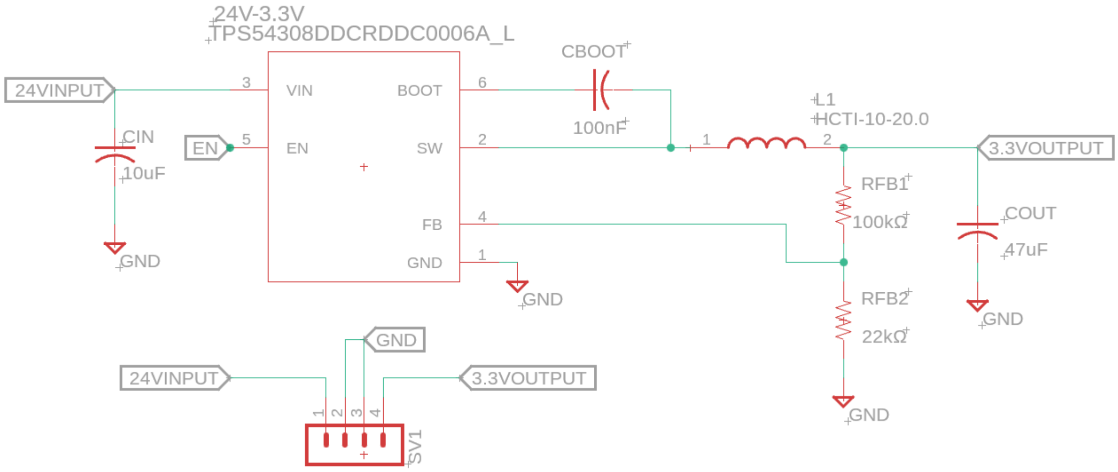


Figure 6.4-1 - 24V to 3.3V Buck DC-DC Converter Schematic on Fusion 360

Next, the ACS712 current sensors, CAN Bus transceivers, and microcontrollers required a supply voltage of 5V. To meet this requirement, a 3.3V to 5V DC-DC boost regulator was employed. The selected converter was the TPS613222ADBVR switching boost regulator, known for its efficiency and reliability. The input of the DC-DC converter was connected to the output of the 24V to 3.3V buck regulator to ensure proper voltage regulation. The output of the DC-DC converter provided a stable 5V supply, which was critical for powering the current sensors, transceivers, and MCUs. Appropriately rated wires were used to handle the current draw safely and efficiently, ensuring the system operated reliably under varying load conditions.The schematic for the boost regulator can be seen in the figure below.

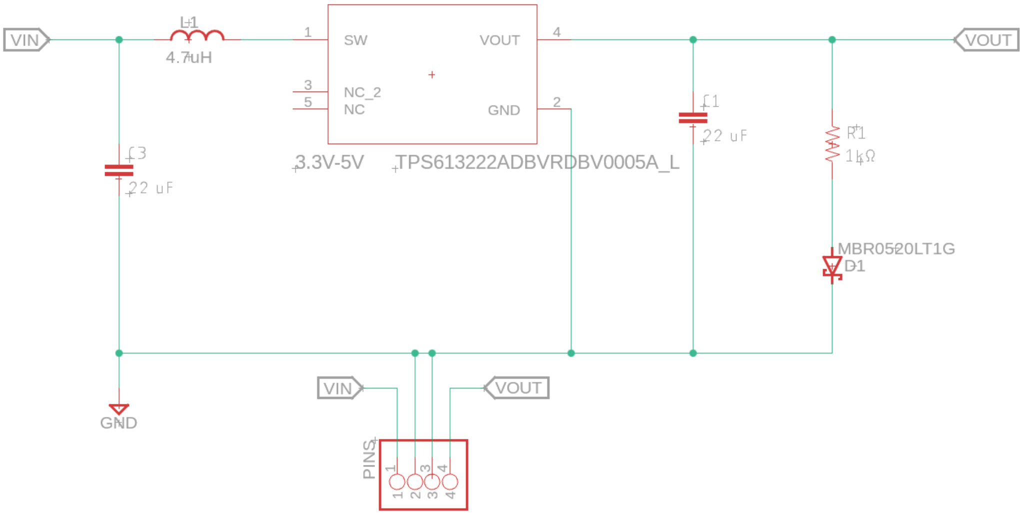


Figure 6.4-2 - 3.3V to 5V DC-DC Boost Converter Schematic on Fusion 360

Finally, the servo end effector required a supply voltage of 5V. Since the end effector demanded a high current to operate effectively, a 24V to 5V DC-DC buck regulator was employed. The selected buck regulator, the LM2576, was chosen for its ability to deliver a maximum output current of 3A, ensuring reliable performance under load. To establish the connection, the input of the DC-DC converter was wired to the 24V output of the power supply. The converter's output provided a stable 5V to power the servo parallel gripper, ensuring smooth and consistent operation. Properly rated wires were used to handle the high current draw safely and efficiently, contributing to the overall reliability of the system. The schematic of this type of voltage converter can be seen in the figure below.

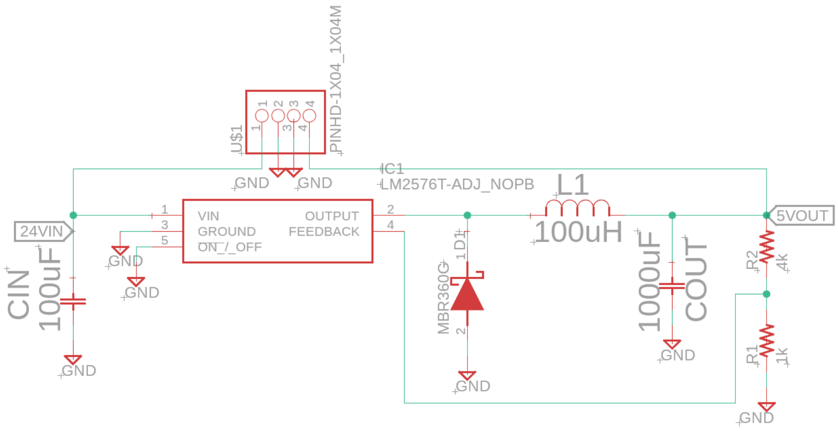


Figure 6.4-3 - 24V to 5V DC-DC Buck Converter Schematic on Fusion 360

6.5 Mechanical Components

6.5.1 Arm:

Our project utilized a multi-input multi-output mechanical differential to control the 3 axis of our robot arm. It allows us to vector power/torque from all motors to any of the axes that need it and lets us make a robotic arm with higher torque than direct on-axis configuration, meaning that we would not have to displace the motor mass to move.

The overall outside of our mechanical arm is not as important as the inside electrical components that drive it, however, below is a diagram of where each component of our arm is, and what the end result modeled. It consisted of three motors, all of them present in the shoulder area, constructed to move joints based on rotational axes.

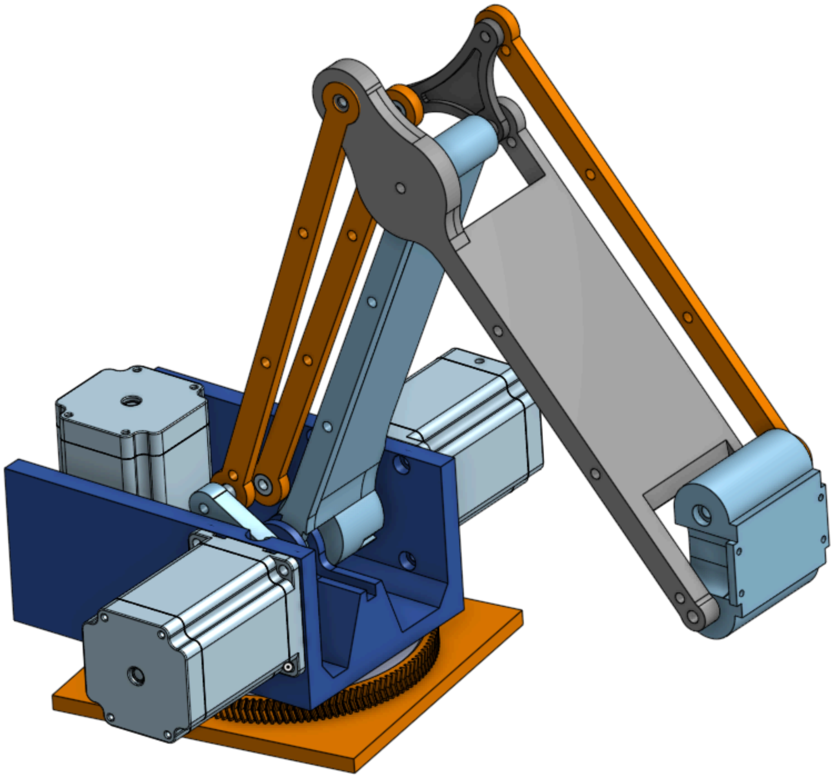


Figure 6.5.1 - Arm CAD with End Effector

6.5.2 End Effector:

The parallel gripper uses a DC servo to actuate a rack and pinion to move the parallel gripper open and closed. The end effector and servo are both mounted at the end of the robotic arm to manipulate objects. This design forgoes a dedicated wrist joint in favor of a more sophisticated shoulder joint. This allowed the robot several degrees of freedom with which it could move without having another motor in the arm potentially causing issues with weight and complexity. While any end effector could potentially be attached, we chose one with the ability to clamp, squeeze, and grab.

The figure below shows the gear system that allowed the gripper to open and close based on the turn of the servo. The movement of the clamps open or closed is dependent on a servo, adjusted to suit our movement needs. This servo turns a wheel that pulls the gripper either open or closed depending on the toggle of the controller.

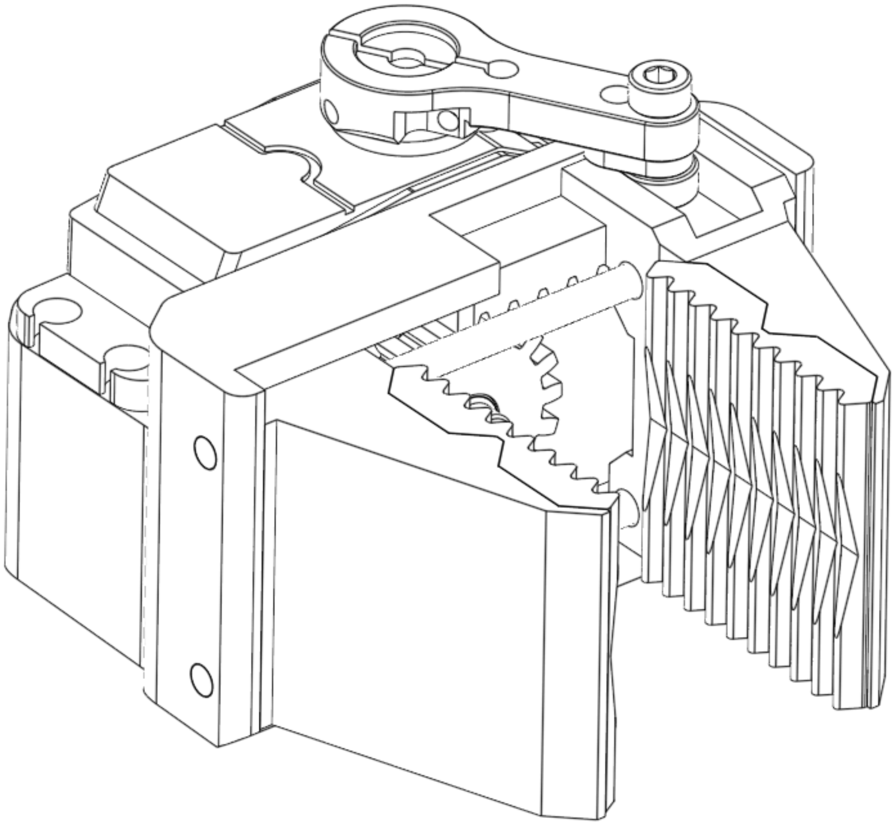


Figure 6.5.2 - End Effector CAD

6.5.2 Base:

The base of a robot arm is a critical component that provides stability, support, and often the first axis of movement. For our prototype, the main type of base that was heavily considered is a fixed base. It is a heavy, stationary platform that is made from materials such as aluminum, steel, or other kinds of metals. This offers stability, making the execution of precision tasks more ideal. Moreover, they are easy to design and construct and less expensive than other bases.

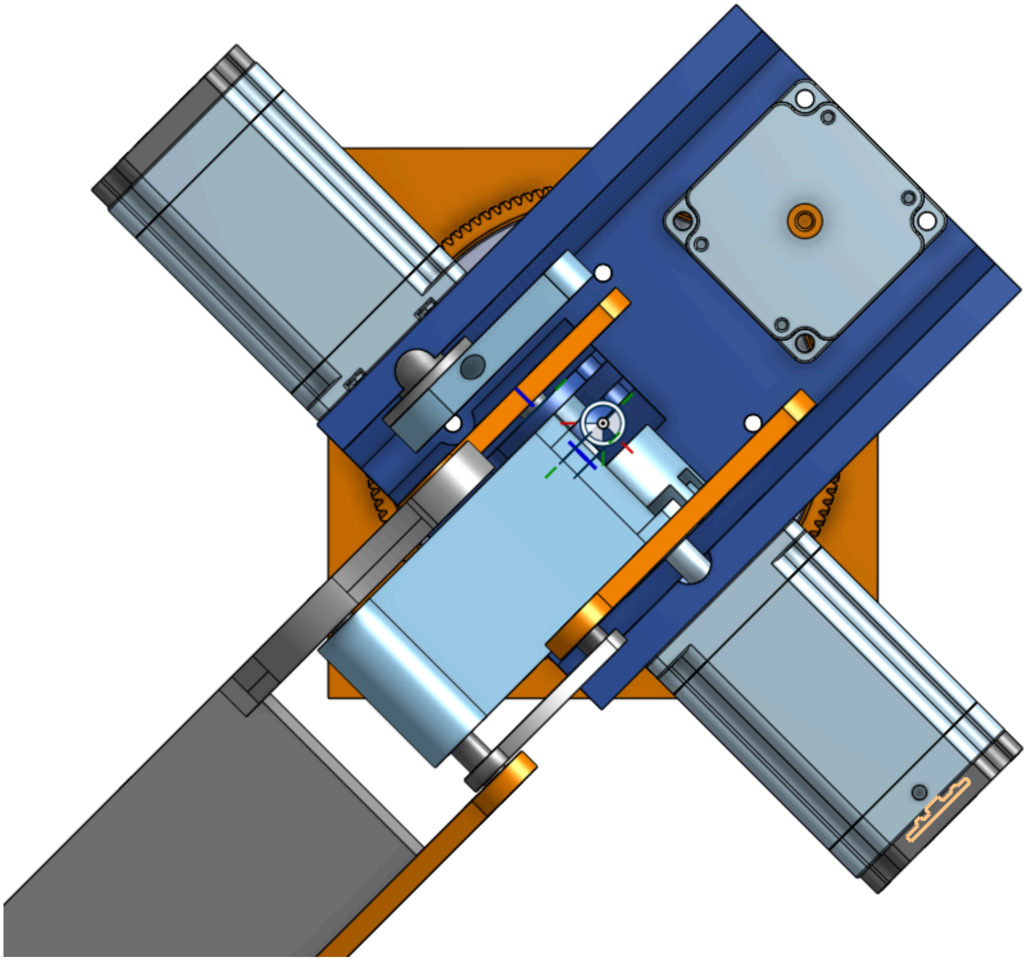


Figure 6.5.3 - Base/Gearbox Motor Mount

Chapter 7: Software Design

This section explains the systems, software, and interfaces that our robotic arm uses to simulate and actuate movement. Below are detailed diagrams of each process, simulator, and system detailed in the software section of chapters 2 and 3.

The software used for our robotic arm is made to interface our ESP32 microcontrollers and CAN buses with our joystick. The code for our robotic simulator was run using the PyBullet simulator library, with the help of the ROS system to give us a wider variety of packages, functions, and tools. The ROS system allowed us to use Python with a pre-built interface, giving us more freedom to correct, build, and personalize our code. The software we are using contains both C and Python code, which are each responsible for their own unique functions. The simulator used on our Raspberry Pi 5 took feedback from our encoders, allowing the operator to monitor the movement of our robotic arm in real time. The code used for our ESP32 microcontrollers allowed our system to process encoder and current sensor data, send it to our Raspberry Pi 5, and then use these values for movement to drive their respective motors, allowing our robotic arm to move. These values from our encoders were mapped directly to our joystick inputs, making each movement meaningful.

Later in this section, it is mentioned that the joystick buttons have the option to switch over to the end effector. While this is a function, it will be included in the robotic arm's analog movement within the software, and referenced as such going forward.

7.1 Flowcharts, Class Diagrams, and State Diagrams

Simplifying our software, we can look at our system as three basic components; the joysticks, the raspberry pi, and the microcontrollers. Each subsystem was accounted for appropriately in our software, and each required their own unique sets of code. The purposes of the software were to pass along data, parse this data, and use it efficiently to move our robotic arm in the desired manner.

By taking in positional data directly from our encoders, we were able to adjust with absolute values the position of our stepper motors. Further implementing a closed-loop system, we used this encoder feedback on our Raspberry Pi to send corrected values to our ESP32s, enabling our motors to adjust themselves to a home position or to restrict motion to specific limits, both upper and lower.

The order of operations goes from joystick input, to Raspberry Pi 5, to ESP32 microcontrollers. The joystick provided the user input, which was either simple analog joystick movement, or activation of one of three buttons. The user was able to choose to record a movement, replay a recorded movement, or freeze all movement for our motors.

All of this analog information was processed by the Raspberry Pi 5, using the EVDEV library, we were able to detect our HID device and assign variables to each motion of our joysticks and activation of our buttons. To successfully record movement, we took all input from our three joints and stored them as keyframes. Using timestamps, we were able to replay each movement relatively accurately, with most replayed movements being smooth and complete. The freeze function on our controller added a delay and stopped all other functions, returning control back to the user after a time of 3 seconds. All of this was done on our computing side, as it was more expensive memory-wise.

Information from our joysticks were passed to our ESP32s using our CAN bus system, which required each ESP32 node to have their own CAN identifier. Each encoder and current sensor for our microcontrollers were identified by separate hex values that the Raspberry Pi used to identify which data was for which motor. On our microcontroller side, the ESP32s used the same system of CAN organization to detect whether input was joystick input or button input, and which one was being sent.

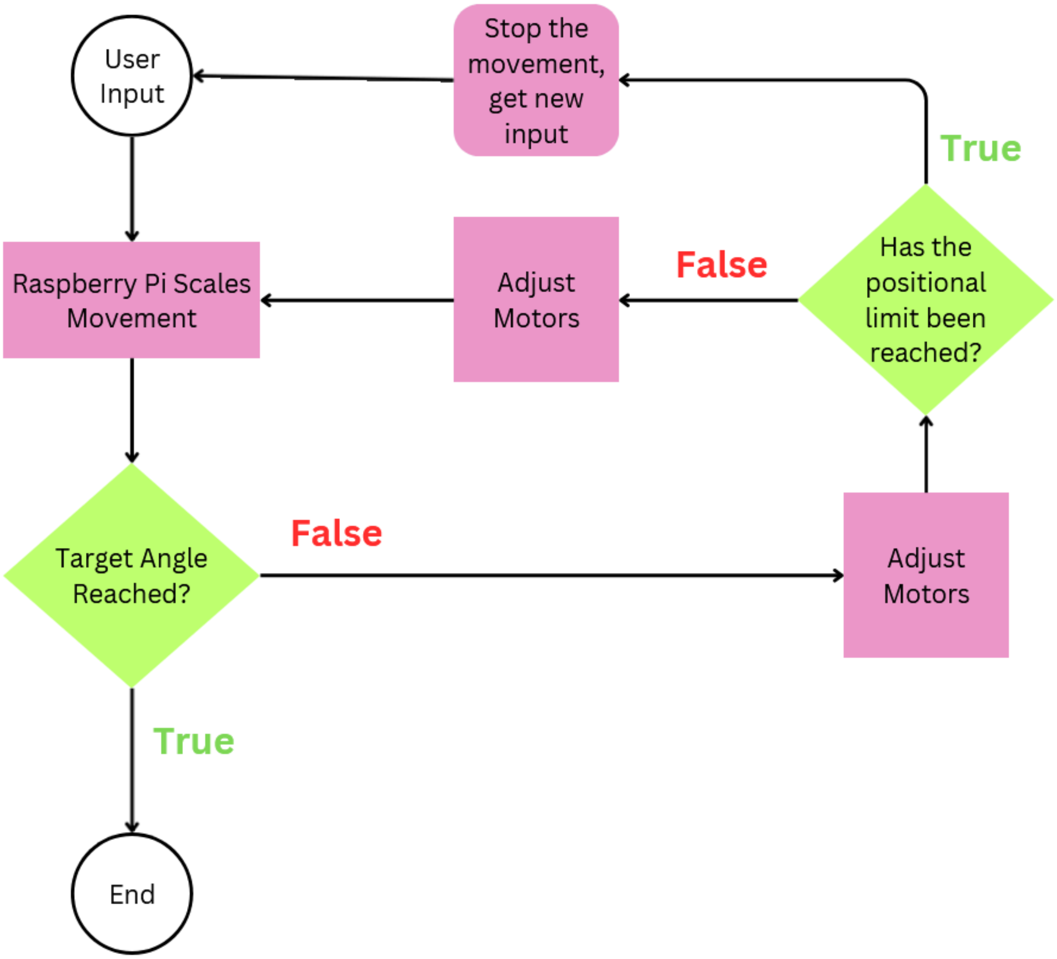

User Input
Stop the movement, get new input
True
Raspberry Pi Scales Movement
Adjust Motors
False
Has the positional limit been reached?
Target Angle Reached?
False
Adjust Motors
True
End

Figure 7.1-1 - Simplified Software Flowchart

To properly interface a simulator with our code, we used the ROS software, which provided unique objects to use when initializing our separate classes.

While many softwares use "classes" to determine relation between objects within software, ROS uses a "Node" system, which allows data transfer between multiple objects at once. Below is a diagram of all the class-relations within our software, officially called Nodes, as they relate to the structure and functionality of each device within our robotic system.

The ROS software acts as an interface between our joystick and our robotic arm, allowing each to understand and parse messages, operations, and data sent from hardware. Both physical devices are dependent on the ROS interface to communicate and function, as it acts as a ground-layer framework for all of the lower functions and communication between our devices and the simulator itself.

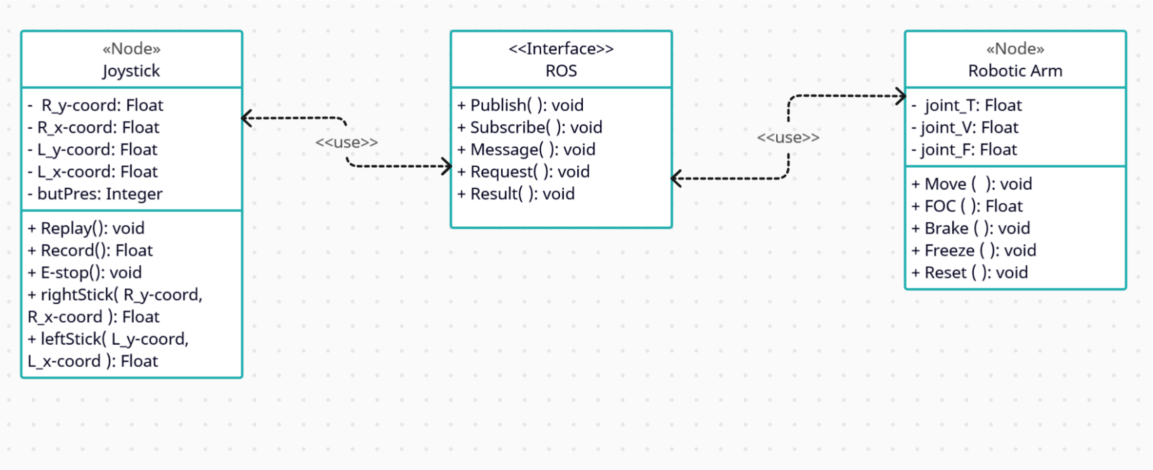


Figure 7.1-2 - Class Diagram

There are a few basic states that our robotic arm can cycle through; moving, grabbing, or stationary. The moving state encompasses any movement, of any joint, input manually from the joystick. This also includes motor angle correction, both when a limit has been reached and when finding a home position, and the replaying action available on the joysticks. This does not include movement from the end effector, this only indicates the state of the arm itself moving in the x, y, or z direction. Grabbing is a separate motion of the robotic arm which involves activating the end effector to grab an object, as well as limiting weight by reducing the torque on each motor. Finally, the stationary state includes emergency stopping, the default pose state, or being frozen due to the end of a movement being reached.

Within the movement state, on a software level, we see commands from our simulator using a movement package. These functions will pass information from node to node, allowing us to send topics from motor to motor, moving the robotic arm precisely. These functions are determined by feedback taken from the encoders on the ESP32 side, and is relevant only to our simulator at this time. The simulator was implemented as a potential solution to operator control, and if an interface were to be implemented with control sliders, it would give the operator the ability to control the movements of our robotic arm directly from the computer. This is not currently implemented due to time and computational constraints.

The stationary state will include uniquely written functions that freeze the state of the robotic arm in case of emergency, freeze the arm in case of movement timeout, and return the robotic arm to default position.

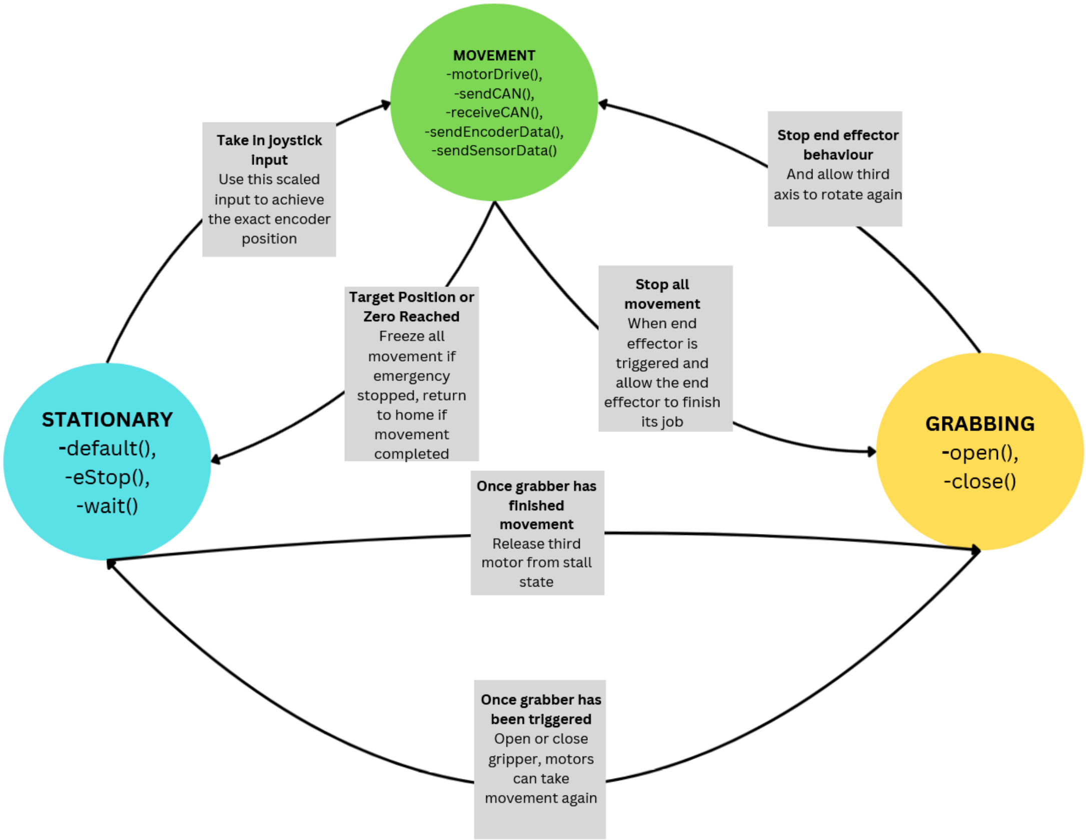

MOVEMENT
-motorDrive(),
-sendCAN(),
-receiveCAN(),
-sendEncoderData(),
-sendSensorData()
STATIONARY
-default(),
-eStop(),
-wait()
GRABBING
-open(),
-close()
Take in joystick input
Use this scaled input to achieve the exact encoder position
Stop end effector behaviour
And allow third axis to rotate again
Target Position or Zero Reached
Freeze all movement if emergency stopped, return to home if movement completed
Stop all movement
When end effector is triggered and allow the end effector to finish its job
Once grabber has finished movement
Release third motor from stall state
Once grabber has been triggered
Open or close gripper, motors can take movement again

Figure 7.1-3 - State Diagram

In chapter 2, we demonstrated how each of our components worked in a singular block diagram. Below are in-depth block diagrams of each individual section, showcasing the individual flow of code of the joystick operations, the simulator operations, and the robotic arm operations (including the end effector).

The joystick, as stated previously, includes analog input to trigger an emergency stop, movement controls, and a replay/record feature. This was done with various button inputs, as well as a connection to a Raspberry Pi computing system with the ROS software interfacing with our PyBullet simulator and main Python script, which allowed us to interface the joystick inputs with our robotic arm's simulator and ESP32s. This

Python script sent requests to our robotic arm to move to our ESP32s, which then sent responses back in a continuous loop until a motion was completed according to the motor angles determined by our Raspberry Pi code and the closed-loop functions in our ESP32s. The end effector takes messages from our Python script as well through the detection of joystick input, allowing it to open, close, and do a rudimentary grabbing for demoing purposes.

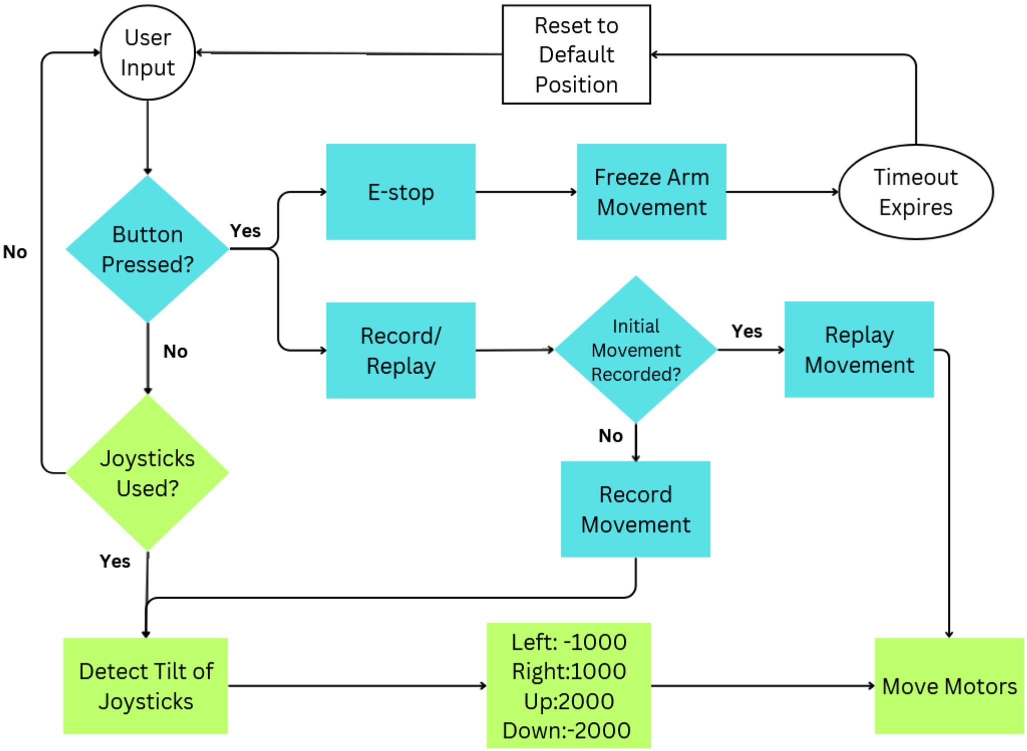


Figure 7.1-4 - Joystick Software Flowchart

Our simulator, which qualifies as a node within our ROS interface, needed to routinely take input from our ESP32 microcontrollers to run effectively. These microcontrollers gave feedback to our simulator by producing real-time parameters from our motors using our positional sensors, allowing our simulator to take these parameters and copy the movement within our 3D space, creating the most accurate movement that it possibly can, that an operator can monitor during operation. The movements If and when the motors encounter resistance from the path that is on, the closed-loop system that the motors use will bounce back to its desired limit, attempting to course-correct by checking the difference in current motor angle and expected angle. The ESP32 closed-loop code will then attempt to adjust the robotic arm's movement by returning to its default state, or below the set movement limit, which causes a sort of bouncing as the motors then fight with the resistance to return to an optimal state.This prevents damage to the arm, and to the object being encountered, when limits are set so that the torque of our motors do not break our frame our any objects collided.

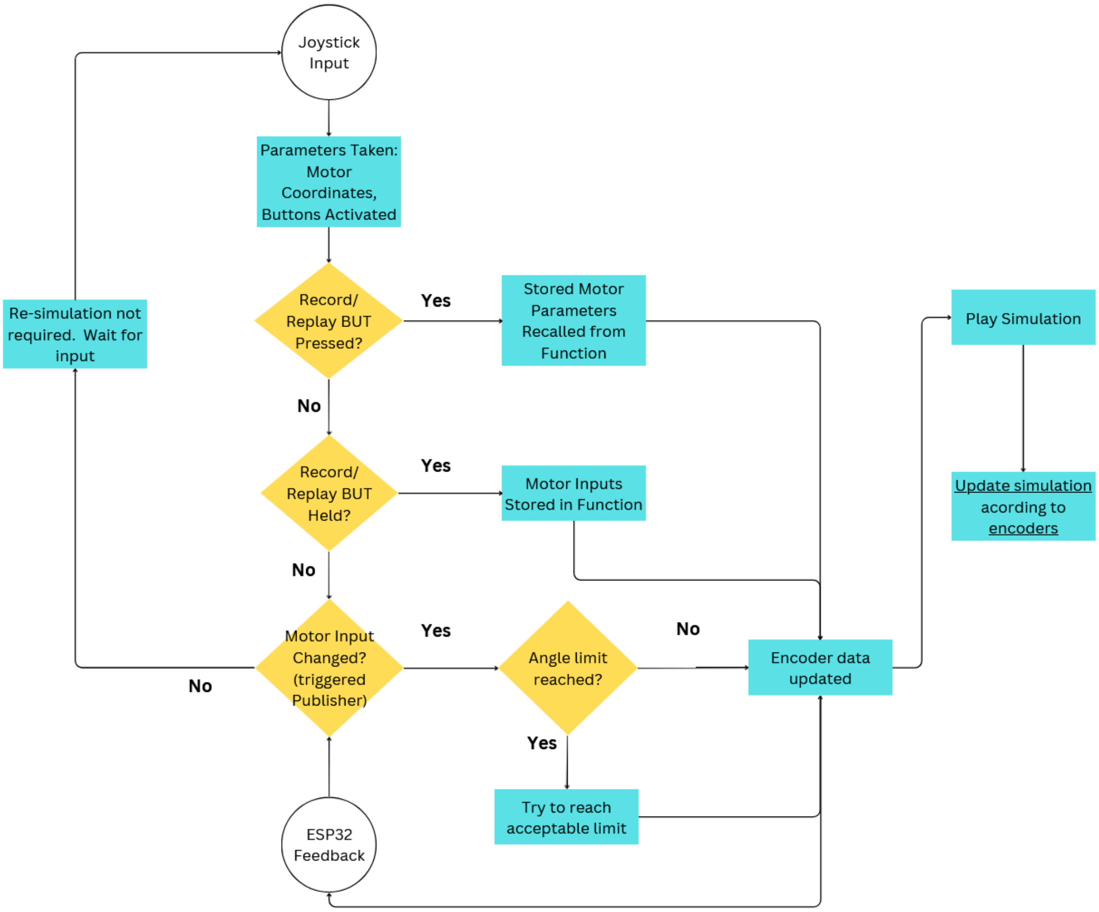

Joystick Input
Parameters Taken: Motor Coordinates, Buttons Activated
Record/ Replay BUT Pressed?
Yes
Stored Motor Parameters Recalled from Function
Play Simulation
Re-simulation not required. Wait for input
No
Record/ Replay BUT Held?
Yes
Motor Inputs Stored in Function
Update simulation acording to encoders
No
Motor Input Changed? (triggered Publisher)
Yes
Angle limit reached?
No
Encoder data updated
No
Yes
Try to reach acceptable limit
ESP32 Feedback

Figure 7.1-5 - Simulator Software Flowchart

7.2 Robot Operating System

To create this feedback-loop properly with our robotic simulator, we needed to update inputs from both the hardware and the simulation software. Allowing these two pieces to talk requires a middleware to send messages back and forth. The Robot Operating System (ROS) makes this easy, as it provides a swathe of functions to use that allow us to pass messages between nodes. The ROS software also provides libraries for specific mathematical concepts, motions, and physics-based concepts that releases the constraint on time it takes to program. This includes libraries that allow us to easily integrate motion systems, drivers, joysticks, and cameras, all of which we are attempting to implement into our project.

The simulation software was done using a Raspberry Pi computer, which gave our robotic arm an accurate approximation of what the movement looks like, and where it should end if there are no obstacles. This simulator was given input via the messaging system of the ROS software and our positional encoders, which allowed us to present the real-time movement of our robotic arm to our human operators. In future uses, this simulator could be used to control the robotic arm completely. In practical implementation, the human operator can see the axis positions of our robotic arm's motors and monitor the system accordingly so that a decision may be made to cut off system functions if something has gone awry.

The software helps us to scale down the complexity of our project, turning what would be a complicated network of motors, microchips, and sensors, into a manageable system of nodes. In this way, ROS essentially acts as our messenger between our computer and our robotic arm, standardizing our communication system between our robotic parts. The robotic arm's motors will act as the "nodes", which will talk to the other "node", our joysticks. ROS acts as the support, the lower-end software that allows our programs to function without the need for new software created entirely from scratch.

Each node was then divided into their functionality groups, having sub-parts to them, which then allowed our arm to be scalable, releasing us from the time constraint of the programming requirements. The three primary nodes we intended to communicate between were the Motors, End Effector, Joystick, and Drivers. If all of our basic goals were met, we planned on implementing ROS nodes for a Camera as well for object detection, however this did not come to fruition due to time constraints.

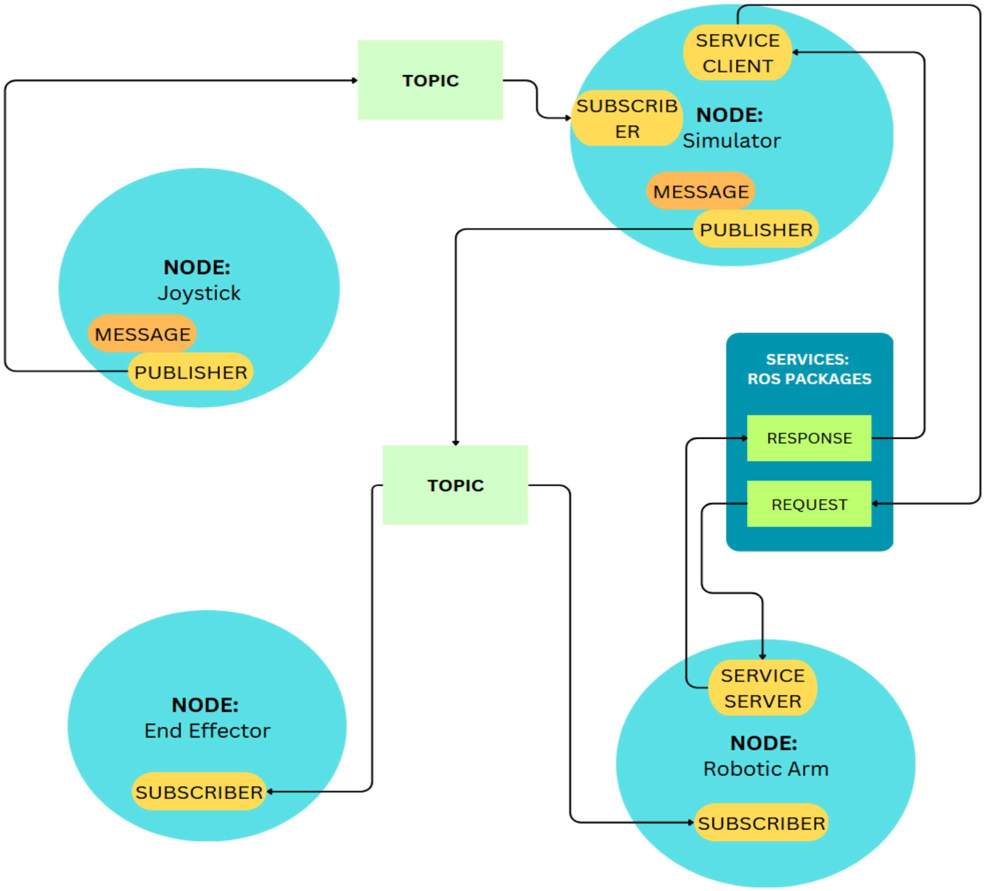


Figure 7.2-1 - Node Structure Diagram

7.3 Node Functionality

The ROS software makes communication possible, and the method in which it moves data through the data-pipeline is what makes this framework scalable. Each node is identified as either a “Publisher” or a “Subscriber” that allows nodes to send “Topics”. These Topics are the actual data being sent over from the Publisher to the Subscriber. The contents of the Topic is called a message, and one Topic may be sent to multiple Subscribers. This will allow us to send and receive new real time data, such as when the Robotic Arm encounters resistance. The new angles and position of the motors will be sent to all other nodes that may need the information.

Nodes may also send “Requests”, which will allow us to request a myriad of processes from one node to another. For example, the arm may encounter resistance, attempt to recalculate its current movement, and send a request to change angles to either keep a current limit from being exceeded, or to find a home value. This may also be a request from the simulator, to “ask” the motors to move the arm to the specified parameters. We were also able to utilize this function when replaying a user-input motion recorded via the joystick by using a built-in package to sample key frames during certain timestamps.

The software used for the joystick node allowed us to telegraph movement to our robotic arm via analog user input. The input was determined by movement from one of two joysticks, the E-stop button for emergency braking, or the Record/Replay button that allows the user to record a motion to save for later.

The joystick functionalities were held in one Python file, along with our robotic simulator, then specific requests and messages were communicated across the data pipeline that ROS used for our PyBullet and 3D robotic model. The same joystick requests were also sent to our ESP32 across our CAN data lines, which will be discussed further in this chapter as “CAN Nodes”. This meant that both joystick axes were able to send information at the same time to their desired nodes. The movement from each joystick was taken as a “Request” for our simulator, which, as described above, allowed the joystick to “ask” for movement from the Robotic Arm. For our ESP32, this data was mapped to specific physical values output by our encoders, which allowed us to set limits to send over from our joystick values. These values were parsed using CAN identifiers, sent to each encoder so that each ESP32 knew which part of the data was for which node, ignoring unneeded information, and taking in the relevant frames.

The Joystick node did not talk directly to our Robotic Arm node, instead using our Computer node to prompt movement. The Simulator node used topics and messages received from the Joystick to then send requests to the 3D model of our robotic arm. All telegraphed movement, replay and record, as well as emergency stop were sent as messages between the publisher/subscriber relationship that the Joystick/Simulator nodes have. Each function for our simulator was considered a separate topic, which meant creating a large Python file, requiring the use of multithreading to give us the option of running functions in parallel rather than serially (one-by-one). This gave us a high level of customizability for functionality, as well as a high level of organization for easy reproduction in the future.

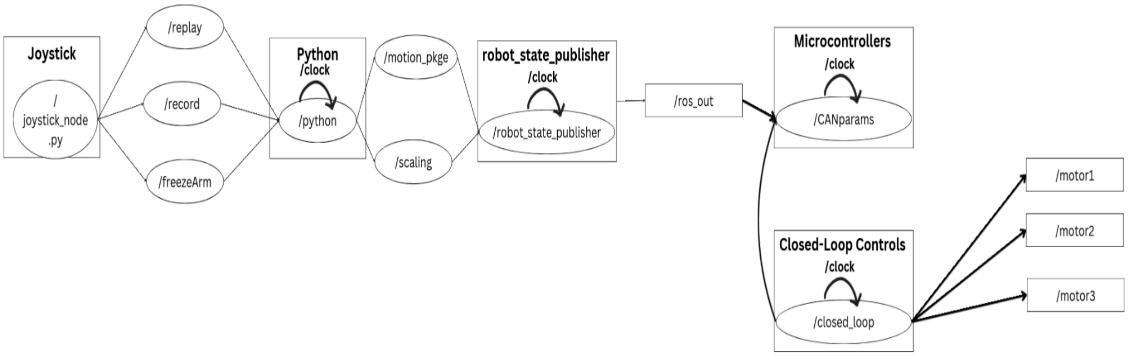


Figure 7.3-1 - Joystick Node Data Transfer Diagram

Each one of the motors then had CAN buses attached to them to facilitate communication between both the motors and the software. The motors did not use the ROS motion package to calculate the Bang Bang functions, but instead used our own code written in a C-language for our embedded systems. The systems used the physical data from the motor's angles, taken absolutely from our positional sensors, that were then transferred to the ESP32s. These systems then used the data gathered to send relevant values over to our simulator, which used the ROS system to simulate our 3D model in real time. The simulator includes supported packages and libraries that do not require us to rewrite or create any of our own Bang algorithms and functions. This allowed us to simulate and calculate our motor's needed position, then take feedback from where the motors actually are to adjust in real-time.

Using a dedicated library for many of our systems gave us more options to implement commands and communicating functions. The ESP32 microcontrollers also required software that allows them to use CAN communications. We originally planned to have this be sourced from Kevin O'Connor's Github, where he has written free-to-use CAN software that allows the microcontrollers to be compatible with CAN devices and communication. This however was not needed, as both Python and our ESP32s provide support for the CAN bus architecture with the use of the can-utils module and mcp2515-arduino. The link to Kevin O'Connor's Github repository, along with our actually used arduino libraries, can be found in Appendix C.2.

Multiple inputs were needed to be taken by the microcontrollers and sent off to other components, which made the use of CAN architecture absolutely necessary. This architecture is further explored in the section below, "Microcontroller Code"

7.4 Microcontroller Code

The robotic arm node consists of our three motors. While the end effector is connected to our robotic arm, it must function differently, and is not connected to this node. For the sake of separating the software into readable sections, we will refer to these nodes as our microcontroller nodes.

With three motors, this means three CAN boards. With each board, there exists a controller chip and a transceiver chip, creating the full node with our ESP32 as our software hub. Each ESP32 has its own encoder and motor to drive, meaning that all of its GPIO pins are dedicated to a particular motor.

This means that within our code, we needed to define all of our encoders, current sensors, and motor driving identifiers separately. With the CAN architecture, this becomes incredibly easy. We first started by defining each encoder with an ID akin to 0x100, 0x200, and so on. To simplify the nature of our code, we'll start with a flowchart to show how all of the microcontroller processes work.

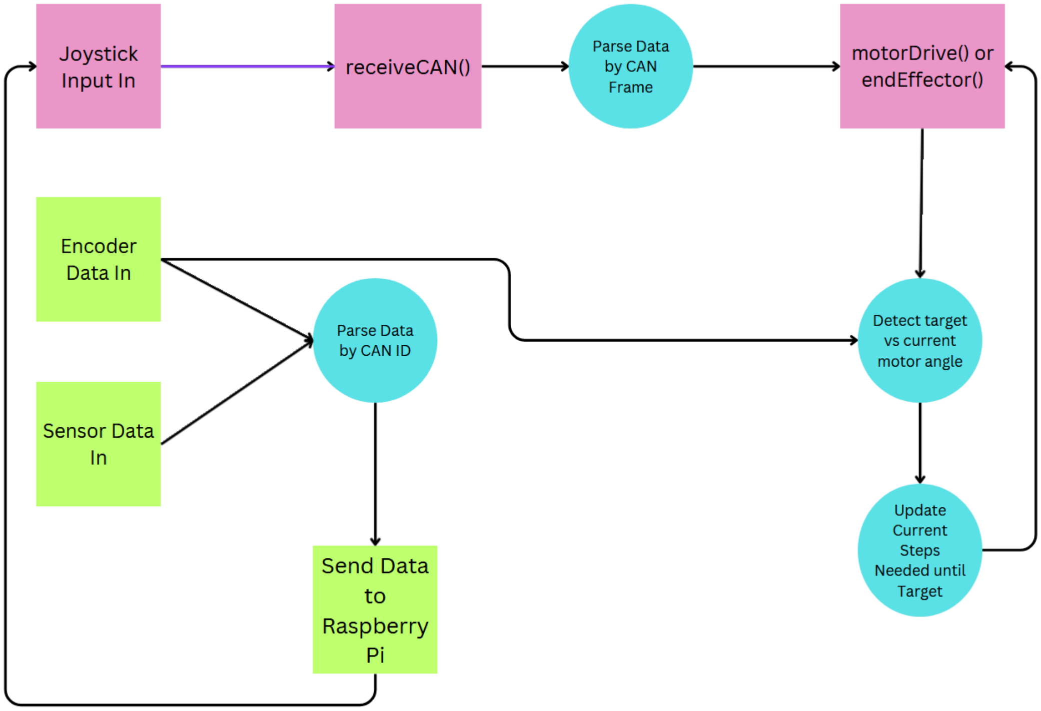

Joystick Input In
receiveCAN()
Parse Data by CAN Frame
motorDrive() or endEffector()
Encoder Data In
Parse Data by CAN ID
Detect target vs current motor angle
Sensor Data In
Send Data to Raspberry Pi
Update Current Steps Needed until Target

Figure 7.4-1 Microcontroller Logic

As we can see in the flowchart above, we start with taking in encoder and current sensor data. This input of data is constant, and by using an updater function to determine if data has changed, the program decides to send this data to our Raspberry Pi. This data is directly responsible for our closed-feedback loop, as our encoder data is crucial to driving our motors to absolute positions and giving us the full control we need to adjust incorrect positioning.

Each node also takes in separate joystick data from the Raspberry Pi, identifying the different frames of the CAN data joystick ID. This data is then passed over to our motor driver function, which takes the target positions (the scaled joystick data) and uses this to move our motors to the desired encoder position. This is done by continuously reading from the motor encoders, and comparing our target data sent to our current position. This is done in a loop until we reach our target, unless our targeted position is above our set limit, wherein our Raspberry Pi will handle the limiting behavior.

All nodes except our base node follow nearly the exact same code, and uses the exact same logic. While each node needs to initialize its CAN module, receive and send CAN messages in the same way, our base node does not drive our motors in this same way. Due to the particular step count our base motor uses, combined with the unique way it moves, it uses an incremental counter instead of a regular closed-loop feedback system. This allows us to precisely control the steps so that the range of motion is limited in a way that adds a delay between direction switching, and does not tear any wires connected to our robotic system.

7.5 Use Case Diagram

Below is a diagram that showcases how the operator, joystick, simulator, and robotic arm all interact with each other. There are no databases that our robotic arm uses, only relying on the analog movement of the joystick, which is transmitted to our simulator, then transmitted to our robotic arm. The simulator and robotic arm are transmitted data to continuously until a movement is completed.

Within the joystick, there are the options to record or replay a movement. This is connected to the analog movements input from the joystick, as the user will need to press record, then demonstrate the movement on the controller. After the user input is completed, the joystick will stop recording. The replay button then allows the software to recall the recorded information, and send this to the simulator. The analog movement will be translated directly to the simulator to be sent to the robotic arm. These movements may move the arm joints or the end effector, depending on the input. The emergency stop (E-stop) button will provide a direct way to freeze the robotic arm in its place. This does not require the input to or from the simulation, and will simply freeze all processes, keeping the arm in place for a designated slot of time before the arm is reset to its default position.

All of this is done using the ROS software and a Raspberry Pi computer, which sends the necessary data and mathematical computations to each component/device as needed.

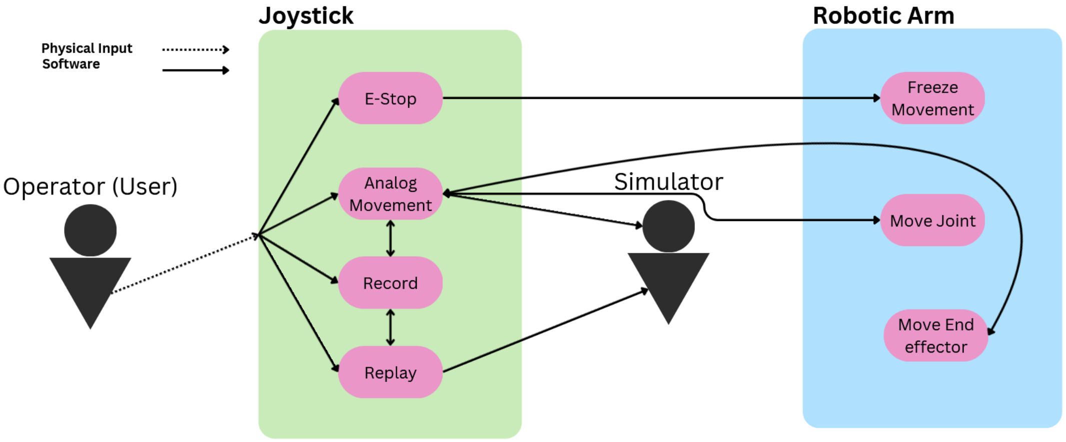


Figure 7.5-1 - Use Case Diagram

Chapter 8: Fabrication of PCBs and Construction of Prototype

8.1 Fabrication of PCBs

8.1.1 Joystick PCB Fabrication:

The Joystick PCB Fabrication involves the detailed layout and tracing of the components essential for our joystick subsystem. The diagram below shows the traces of the PCB, highlighting the placement and routing of various components. Key components such as the micro USB connector, ICSP header, joysticks, buttons, power LED, and the ATMEGA32U4 microcontroller are strategically placed for optimal performance and accessibility.

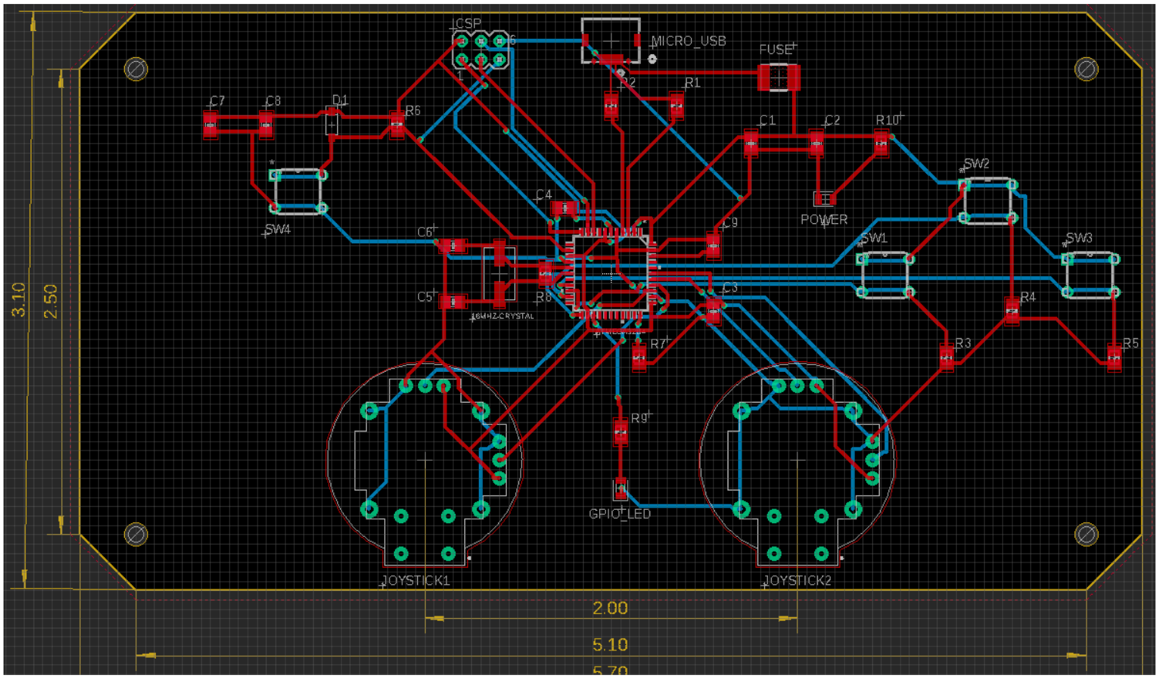


Figure 8.1.1-1 - Joystick PCB Trace Diagram on Fusion 360

The joysticks are positioned 2 inches apart to mimic modern-day console controllers, providing comfort and ease of use. Similarly, the buttons are placed in ergonomic positions for intuitive operation. The layout also includes decoupling capacitors, a crystal oscillator, and necessary resistors, ensuring stable operation and reliable performance of the joystick subsystem. This carefully designed PCB trace diagram ensures that all connections are properly routed, minimizing interference and ensuring a smooth flow of signals. The traces were routed with 0 DRC errors, confirming the correctness and reliability of the design. It is also worth noting the labeling of the components was done in a simple and strategic manner so that they're visible after manufacturing, allowing for easier soldering.

Additionally, the Joystick PCB pick-and-place diagram below highlights the component placement throughout the PCB on JLCPCB. With this we were able to see if all the components were properly sized and orientated. The manufacturer also does a check of their own based on what their software says and contacted us after realizing the crystal oscillator was too large for the footprint, but after a simple part replacement the board was readily manufactured and delivered.

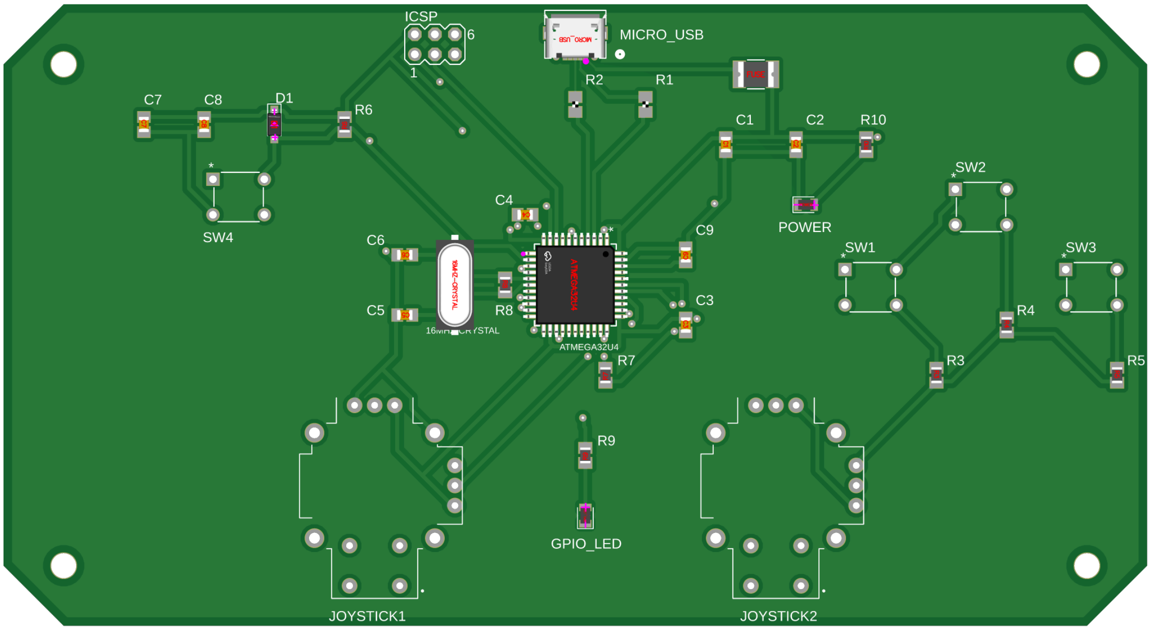


Figure 8.1.1-2 - Joystick PCB Pick-and-Place on JLCPCB

Finally, the 3D model below provides a visual representation of the Joystick PCB. While it does not include 3D models for all components due to limitations in part libraries, it effectively illustrates the overall layout and design of the PCB. This model gives a clear view of how the components are arranged and how the PCB will look when assembled, ensuring that the ergonomic and functional aspects are accurately reflected.

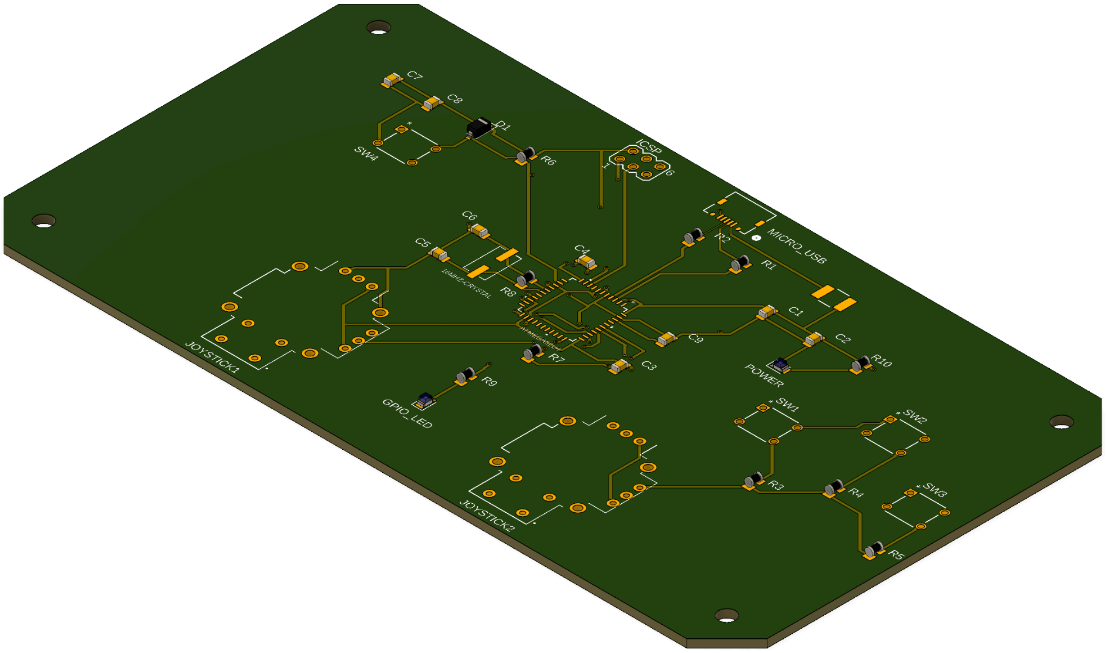


Figure 8.1.1-3 - 3D Model of Joystick PCB on Fusion 360

8.1.2 CAN Bus PCB Fabrication:

The CAN bus PCB below shows the finalized board design for our CAN bus devices. We originally used an imperfect board, which was missing its ground planes and included a flipped socket. These final boards have all proper connections, including the proper ground planes and socketed positions for the MCP2515.

Key components include the 18-dip socket for our MCP2515 controller, and our 8-dip socket for our MCP2551 transceiver. The CAN-H and CAN-L lines are connected directly to this 8-dip transceiver, and our 5V and 3.3V lines are split to ensure a safe voltage line between our MCP2515 and our ESP32s.

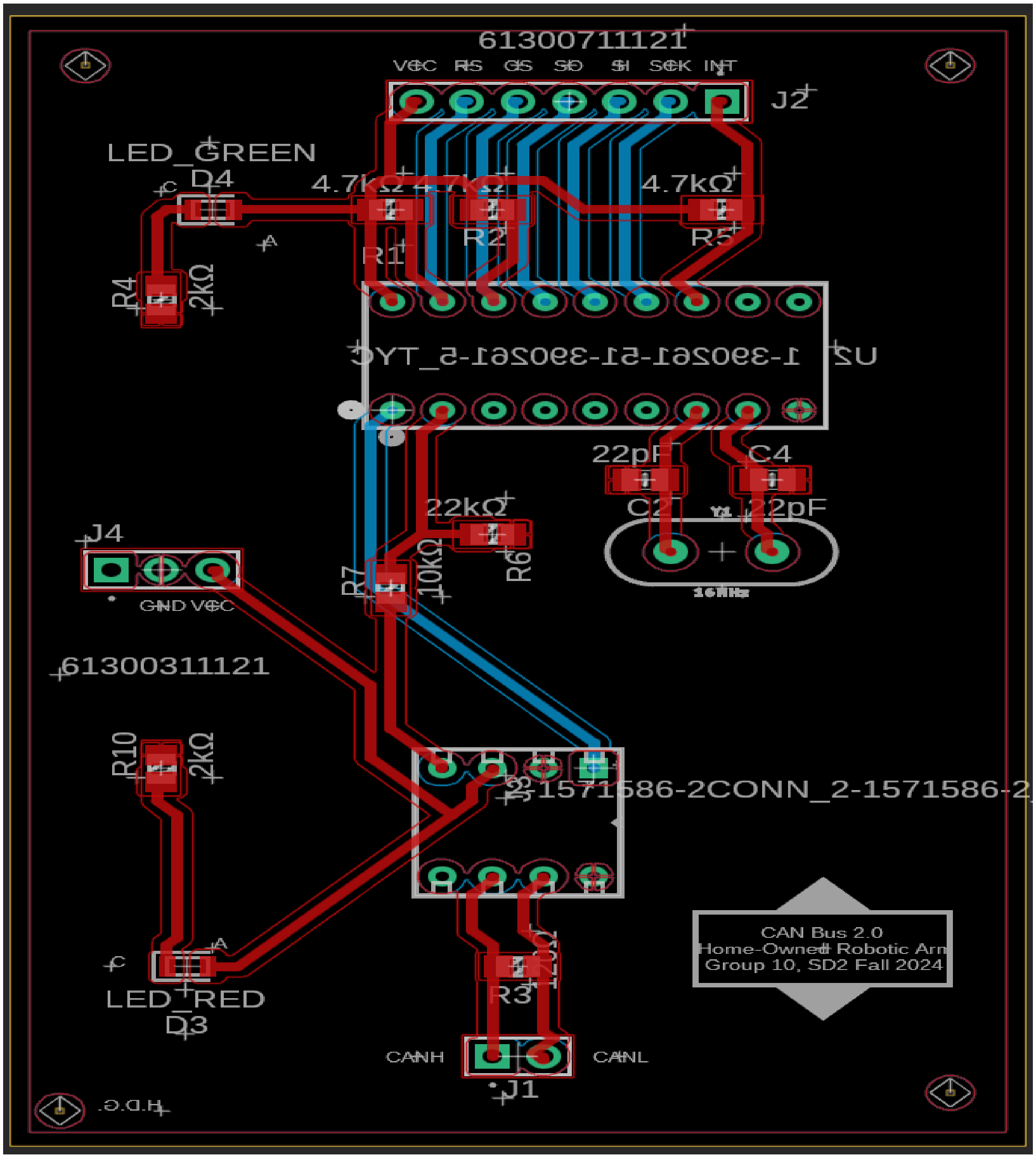

61300711121
VCC RS CS SO SI SCK INT
J2
LED_GREEN
D4
C
A
4.7kΩ
4.7kΩ
4.7kΩ
R1
R2
R5
R4
2kΩ
1-390261-51-390261-5_TYC
U2
22pF
C4
C2
22pF
22kΩ
R6
10kΩ
R7
Y1
16MHz
J4
GND VCC
61300311121
R10
2kΩ
2-1571586-2CONN_2-1571586-2
U3
CAN Bus 2.0
Home-Owned Robotic Arm
Group 10, SD2 Fall 2024
C
A
LED_RED
D3
R3
CANH
CANL
J1
H.D.G.

Figure 8.1.2-1 - CAN PCB Trace Diagram Fusion 360

Added onto the schematic is the voltage divider between the RXD lines of our MCP2551 and MCP2515 chip. This voltage divider is added due to the fact that the MCP2551 runs at 5V, and our MCP2515 runs at 3.3V. With a 5V going back into our 3.3V, this burnt out a few of our chips with our initial design. Adding this voltage line, and some bypass capacitors attached to our VDD and ground pins of both of our chips were necessary to create safe and functional boards. Using resistors of values 10k Ohms and 22k Ohms was more than enough to create this safe voltage line.

Also included is the parallel, 16MHz external crystal oscillator attached to our MCP2515. This allowed our SCK line to run at a proper frequency, giving us high data speeds and accurate data transfer during robotic operation. This clock required capacitors of at least 22pF and a distance from the chip no further than 6mm.

A red LED indicates the power line of our 5V MCP2551 chip receiving power, while a green LED indicates the power line of our 3.3V MCP2515 chip receiving power. The pin headers are for connections to all SPI connections between our ESP32 and CAN boards, power for our 5V and ground connections, and CAN-H and CAN-L data lines that all CAN boards must connect to each other with.

This model did not have any housing, but can easily be mounted to a flat surface with double-sided tape. Below is a photo of our finalized boards, first showing the front of our boards where all necessary components are soldered, then the back where we added bypass capacitors of 10 microFarads from the voltage lines to the ground lines.

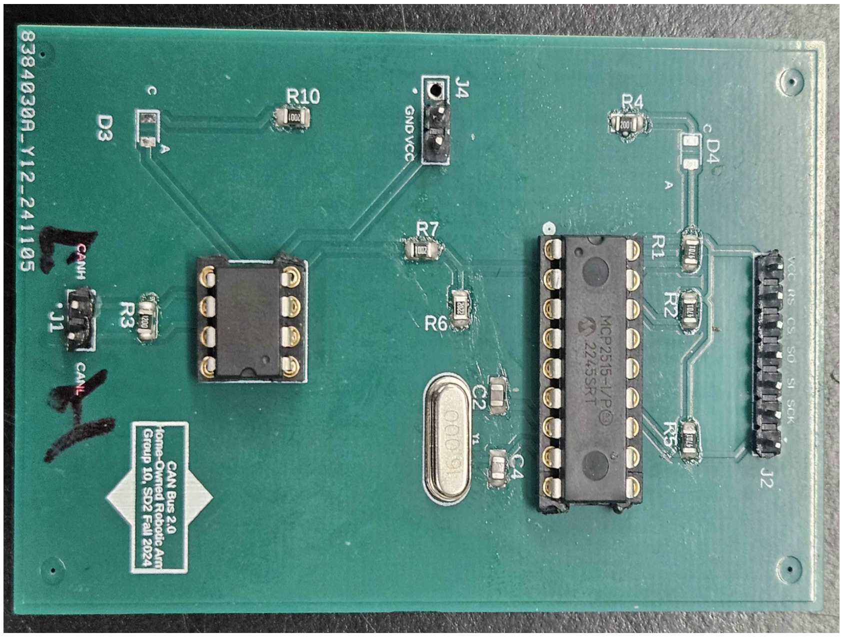


Figure 8.1.2-2 - Finalized CAN PCB Front, All Surface-Mount Components

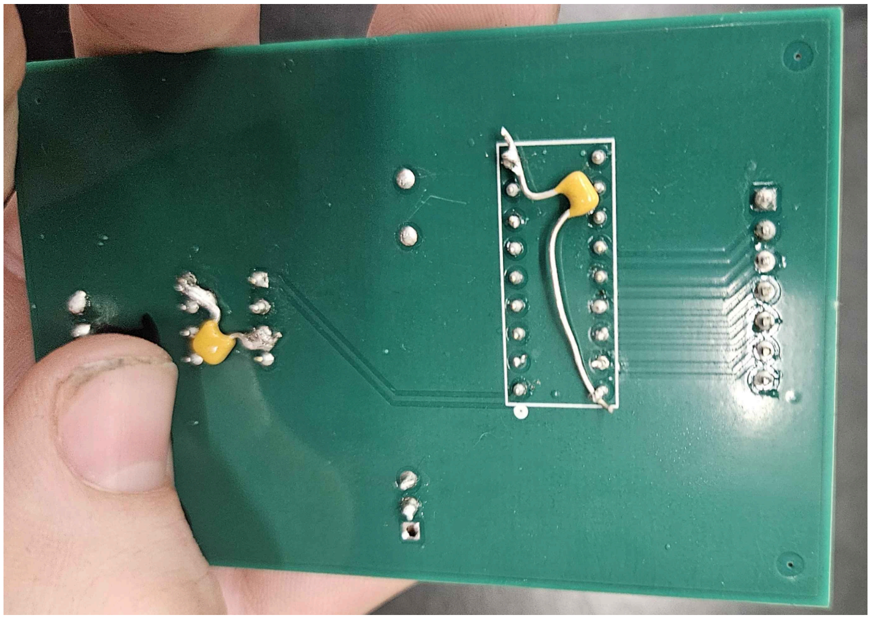


Figure 8.1.2-3 - Finalized CAN PCB Back, Bypass Capacitors

8.2 Construction of Prototype

8.2.1 Joystick Prototyping:

To prototype our joystick controller, we utilized a breadboard to connect two joysticks and three buttons to an Arduino Leonardo. The joysticks were wired to the analog pins A0 to A3 on the Arduino, allowing for the reading of X and Y axis values for each joystick. This setup enabled us to capture the directional input from the joysticks, which is what is essential for controlling the robotic arm's movement. The three buttons were connected to digital pins D0 to D2 on the Arduino to detect button presses, which could be used for additional control functions such as activating the end effector or emergency stop.

The Arduino Leonardo was connected to an external laptop running the provided code, which initialized the serial communication at a baud rate of 9600. This setup allowed us to read the joystick and button outputs using the Serial Monitor in the Arduino IDE. The serial output displayed the X and Y values for each joystick and the state of each button, providing real-time feedback for testing and debugging. This prototype setup was crucial for verifying the functionality of our joystick controller before integrating it into the final PCB design and ensuring that all components worked together seamlessly with the intended microcontroller. By using this breadboard setup, we were able to make adjustments and troubleshoot any issues that arose during the development process.

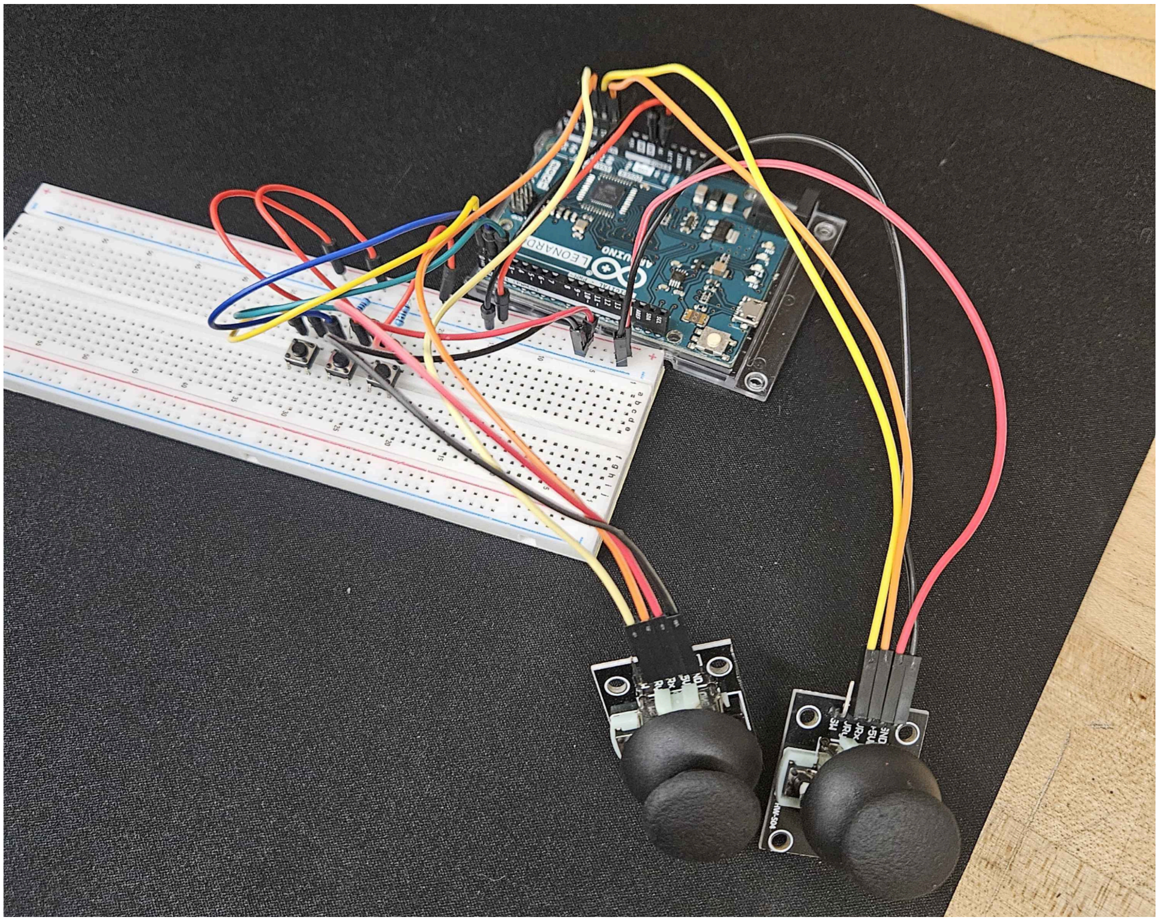


Figure 8.2.1 - Joystick and Button Prototyping with Arduino Leonardo

8.2.2 Motor Prototyping:

To test our motors controlled by an H-bridge, we connected them to an off-the-shelf dual H-bridge, the L298N. We then connected the L298N's IN1, IN2, IN3, and IN4 inputs to a logic level shifter, allowing the Pico to safely control the board while also powering both boards and programming the Pico. During testing at various speeds, we realized the necessity of flyback diodes to protect the circuit components from the motor's back EMF. We successfully implemented speed control by varying the time between steps, causing the motor to spin faster or slower.

While we achieved basic speed control, we have yet to reach torque control, which requires modulating the signal's amplitude rather than just its frequency. This aspect is critical for precise movements and will be a focus in Senior Design 2, where we will implement Field-Oriented Control (FOC) to manage both speed and torque. Additionally, we plan to integrate current and positioning sensing feedback to enhance the system's responsiveness and efficiency. This will allow us to refine our control algorithms further and ensure smooth, precise operation of the robotic arm under various loads and conditions. We were also able to achieve very accurate positional control and resolution.

During prototyping, we found that we need to design additional safety mechanisms to protect our control circuit from the motors.

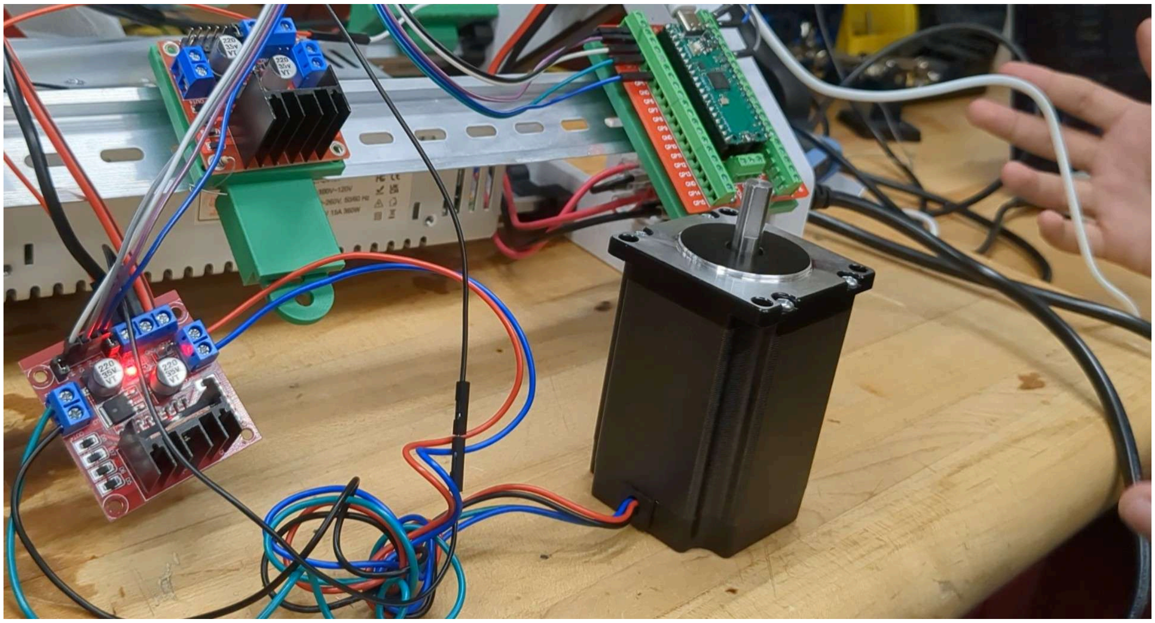


Figure 8.2.2 - Motor Prototype Connected with H-bridge, Microcontroller, and Gate Drivers

8.2.3 CAN Bus Communication Prototyping:

During our original prototyping and proposing of this project, we originally used the RP2040 microcontrollers, before making the decision to switch over to the easier and more accessible ESP32 microcontrollers. Due to that fact, our original prototypes use the RP2040 microcontrollers.

Each of our four motors uses their own ESP32 microcontroller in real practice, connected to a CAN bus, to send data serially between each other and to our Raspberry Pi computer. To test the functionality of our CAN buses, however, we purchased three pre-built CAN modules from Amazon. Using a logic level shifter, we were able to make the 5V MCP2551 CAN module compatible with our 3.3V RP2040 and ESP32 microcontrollers.

During prototyping, to create a proper connection between our modules and our CAN communication, we required two nodes to be set up. Node 1 acted as our transceiver, consisting of an RP2040 microcontroller, a logic level shifter, and a CAN module dev board. Node 2 acted as our receiver, and consisted of a Raspberry Pi 2 Model B microprocessor and a second CAN module dev board. For our official prototype in Senior Design 2, we used a Raspberry Pi 5. The Raspi 2 microprocessor has a voltage tolerance from 4.7-5.25V, so it was safe to use the CAN module without a logic level shifter. All of these components were connected via a breadboard for circuit connections. The CAN modules themselves were connected to each other via their CAN-L (CAN Low) and CAN-H (CAN High) pins.

The Raspi 2 microprocessor was used to load our needed code, which can be found in Appendix C, to our RP2040 microcontrollers. The initial set up of the Raspberry Pi required the purchase of a new SD card and OS loaded onto it. This also required us to set up the coding environment that our project will be using long-term, allowing us to have a transferable computing system in the case that we would need a new Raspi 2 component.

The microcontroller itself was placed on a screw-pin housing. This was for organization, and did not change the functionality of the components. The microcontroller was powered using a USB-C connection. We then connected the proper pins to a logic level shifter, which allowed us to bring our CAN module dev-boards down from 5V to 3.3V, allowing us to not burn out our RP2040 microcontroller. We loaded our RP2040 initially within SPI mode, allowing us to use the CAN code for communication. The actual communication was sent through our Raspi 2 microprocessor to confirm communication.

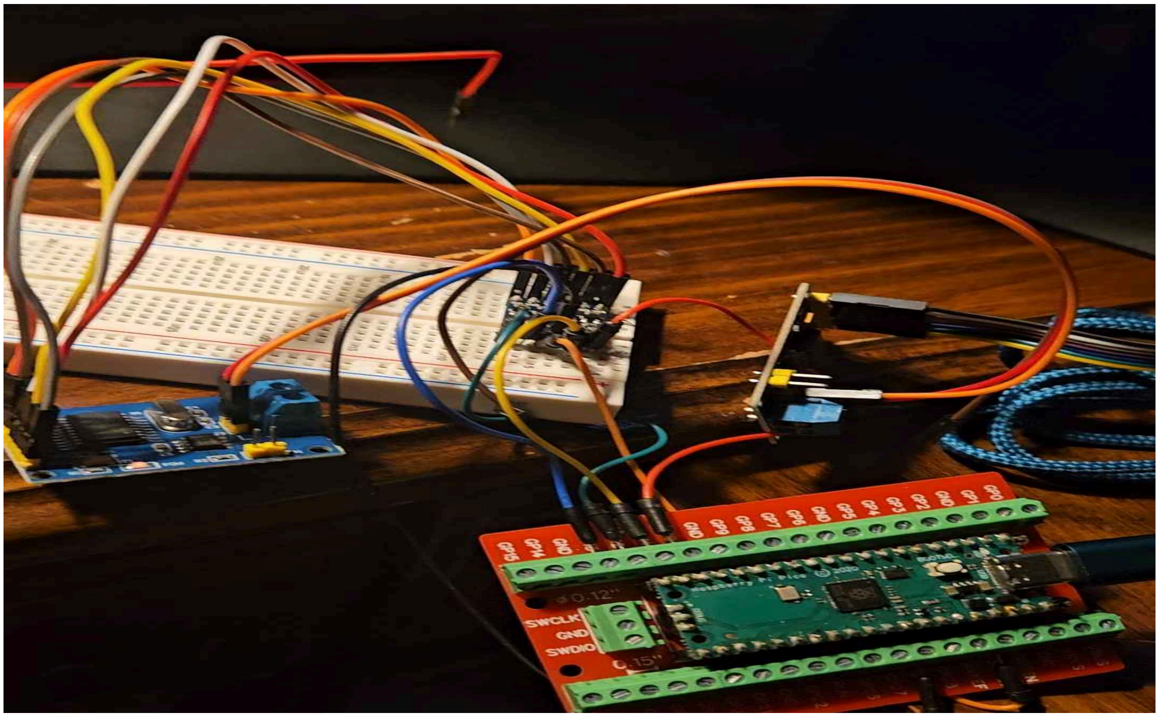


Figure 8.2.3 - CAN modules and microcontroller connections

The image above demonstrates the connections between our CAN module devboards and our RP2040 microcontrollers. For prototyping purposes, we used a logic-level voltage shifter to handle the 5V intolerance of our microcontroller. The voltage shifter sits on the breadboard, connected on the high voltage end to the CAN module, and connected to our RP2040 on the low voltage side.

Two microcontrollers were needed with this exact setup to act as separate nodes. Node 1 acted as a transceiver node, while Node 2 acted as a receiver. This is required for CAN connections, since the modules cannot communicate if only one is connected. These nodes were then connected to one master host (Raspberry Pi), which facilitated reading communication across the lines and debugging. Both nodes had to be identical to function, and in future testing these CAN devices will have different transceivers to adjust for the voltage discrepancy between our microprocessor, microcontrollers, and CAN modules.

During prototyping with our new microcontrollers, we used a very similar process, except with different GPIO pinouts. To properly test our CAN boards, we hooked our finalized PCBs into working development boards as a benchmark, ensuring our code worked, then our hardware, one by one. Once this was accomplished, we then started integrating our CAN boards, until we had all three of our own custom PCBs in our system communicating correctly with our ESP32s.

8.2.4 Power Supply Prototyping:

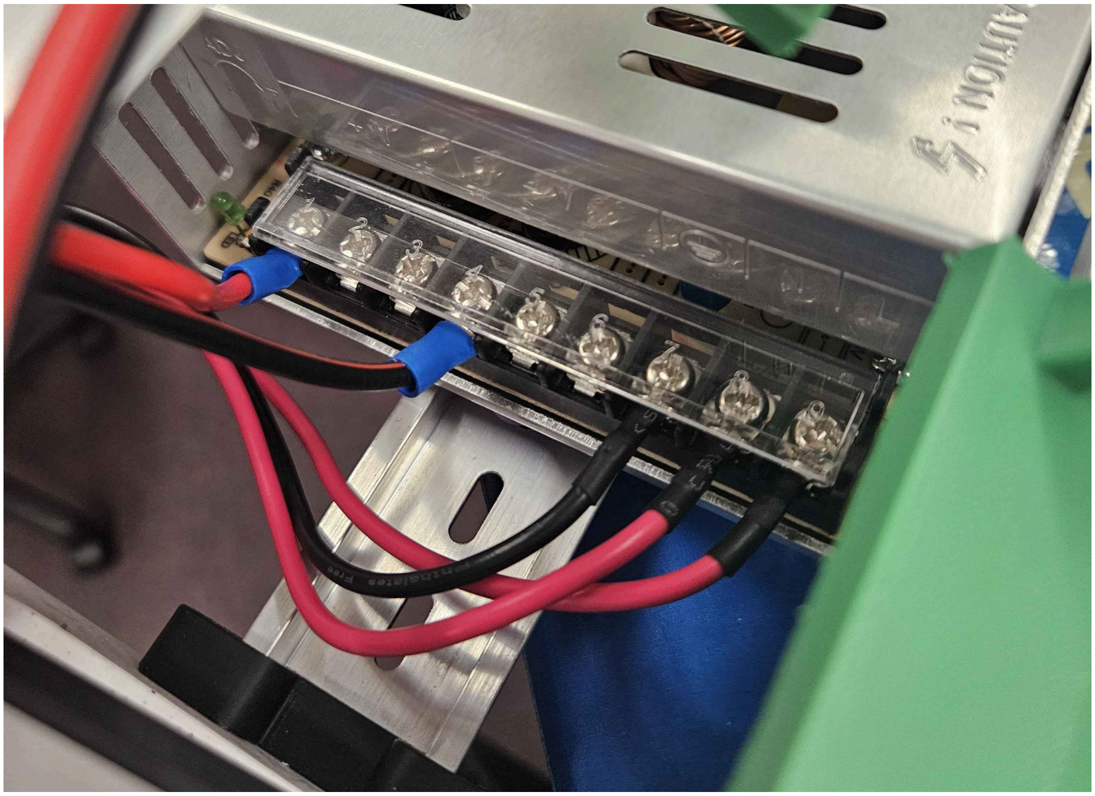


Figure 8.2.4-1 - Power Supply Connections

To prototype our power supply and power distribution system we used a pre-owned power supply with similar specs and dimensions to our selected power supply. 120 VAC to 24 VDC just like our chosen power supply. Their prototyping involved hooking up the power supply to an enclosure we also already owned and routing the wires. The enclosure contains a cable and switch that can be connected to a standard U.S wall outlet to feed the power supply the power to be distributed through the rest of the design. The power supply has one pin for hot, neutral and ground coming from the outlet source. The power supply has 3 positive and negative pins each of which output 24 DC volts. For ease and so the power could be further broken down and distributed, the power is routed from the 20 VDC output pins of the power supply to a separate part of the circuit where the power can be more easily distributed.

The ground, neutral, and hot lines coming from the outlet connection all had preexisting forked crimp connectors that could be installed directly into the screw terminals of the power supply. We then had to route the power from the power supply itself to the distribution network. To accomplish this we selected a roll of dual red-black 14 gauge copper wire. This wire had the convenience of being both available to us in the lab and in the same colors as the input wires from the outlet. After that it was a matter of cutting and stripping the wire so crimp connectors could be installed on both ends. On the power supply end we used forked connectors to be installed into the screw terminal of the power supply. On the other end we used a different style connector intended to be installed into the plastic connectors we have on the power distribution network.

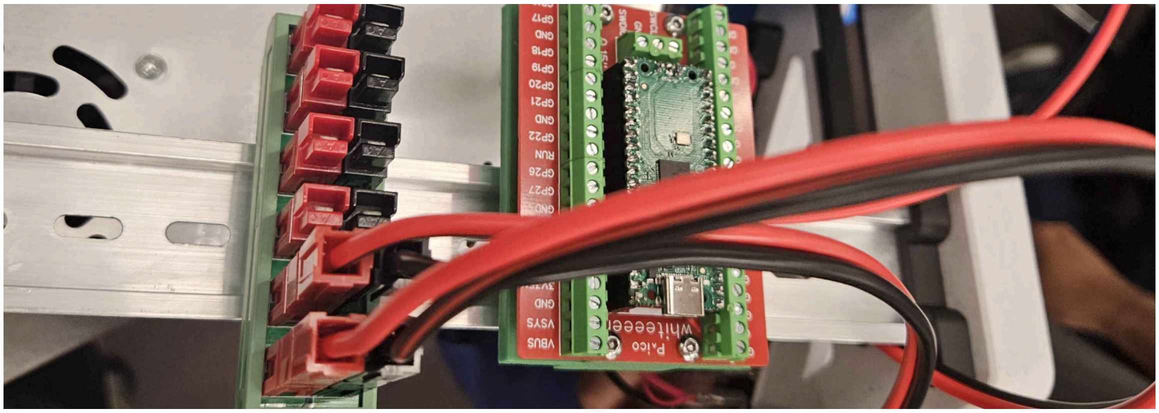


Figure 8.2.4-2 - Power Distribution Connections

8.2.5 Regulator Prototyping

To prototype our regulators for the power distribution subsystem, we first evaluated the voltage levels each of our components need to be powered at and how much current they will draw. To do this we consulted our power distribution table which is broken down into 3.3, 5, and 24 volts. We also initially included a 24 to 10 volt regulator for the gate drivers, but that was axed as the project moved away from creating a motor controller and toward implementing one. To step down from 24 volts we selected a TPS54308DDCR adjustable buck regulator and created a layout to run it at 3.3 volts.

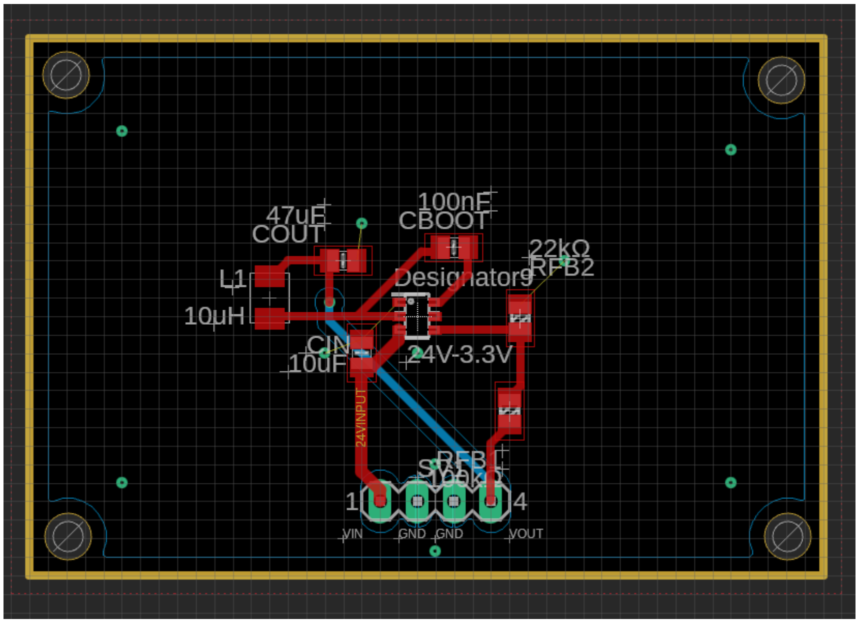


Figure 8.2.5-1 - 24 to 3.3 Buck Regulator

Below is the layout for the 3.3 to 5 step up circuit using a TPS613222ADBVR boost regulator. The layout was created based on the specifications laid out in the datasheet, though the layout itself had some significant differences from the recommended layout due to knowledge limitations at that point in the design process. A similar result occurred with the 24 to 3.3 regulator leading to the creation of a third regulator circuit to serve as a standalone replacement to that 24 to 3.3 board.

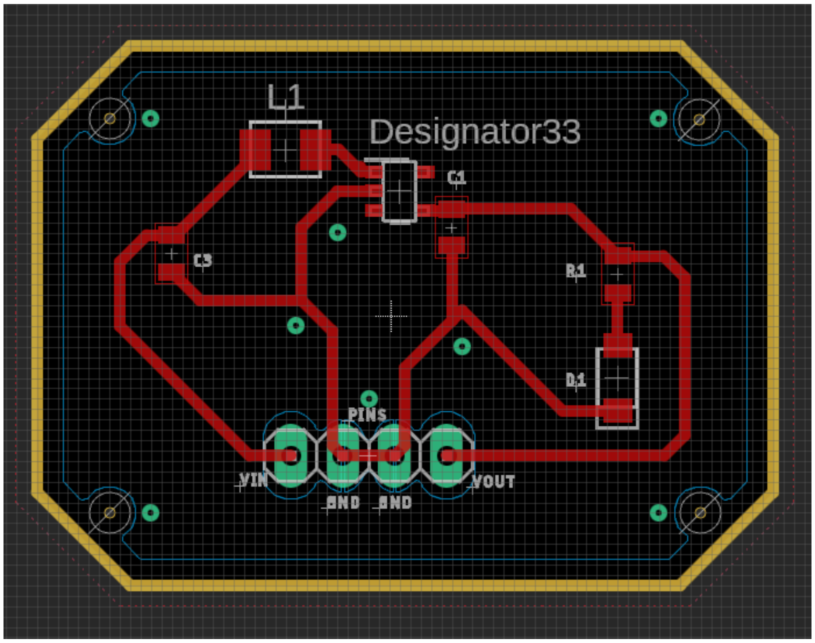


Figure 8.2.5-2 - 3.3 to 5 Boost Regulator

Next is the revised step down adjustable regulator. It was created as a late addition due to previously unknown failures shown by the earlier 24 to 3.3 regulator. To avoid issues related to layout sensitivity that we had been experiencing with some of the Texas Instruments surface mount regulators, we opted for the through-hole LM2576T-ADJ regulator. This came with the added benefit of being able to be assembled on short notice. The datasheet for this regulator includes a formula that shows the feedback resistor ratio required for a particular output voltage. To design our circuit for 3.3 volts, our feedback resistors had values of 3.4kohm and 2kohm respectively. Since the output depends on these resistors, we were also able to adjust the 24 to 3.3 regulator into a 24 to 5 volt regulator to power boards that took a 5 volt input voltage instead of a 3.3v input.

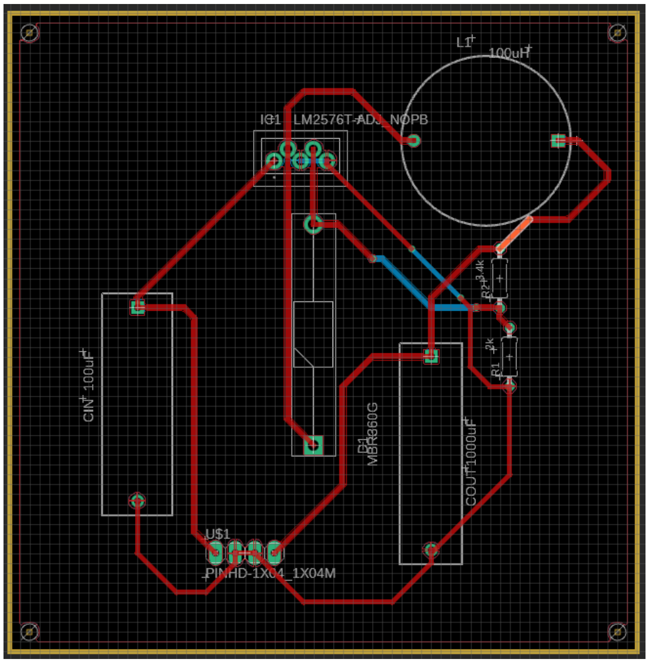


Figure 8.2.5-1 - 24 to 3.3 Buck Through-hole

8.2.6 MCU Prototyping

The MCU PCB fabrication centers on integrating essential components for the microcontroller unit, enabling robust functionality and seamless connectivity within the system. The detailed schematic and layout ensure optimal placement and routing for key components, such as the microcontroller, voltage regulators, level shifters, interface headers, and LEDs. This meticulous arrangement facilitates reliable performance and streamlined interfacing with other subsystems.

Figure 8.1.2-1 - MCU PCB Trace Diagram in Fusion 360
The design incorporates a carefully routed power distribution network, supporting +3.3V and +5V voltage domains with decoupling capacitors strategically positioned for noise reduction and stability. Key features include:

- A 3.3V regulator with input and output capacitors ensuring stable power delivery.

- Headers for SPI, I2C, and UART interfacing, simplifying connections to external peripherals.
- An HCT125 level shifter for reliable logic-level conversion.
- Reset and configuration push buttons for debugging and control.
- A power LED to indicate active status.

This PCB was designed with an emphasis on accessibility and reliability, using 0 DRC error routing to ensure correct electrical connections and minimize interference. The component labels are positioned for clarity during assembly and debugging.

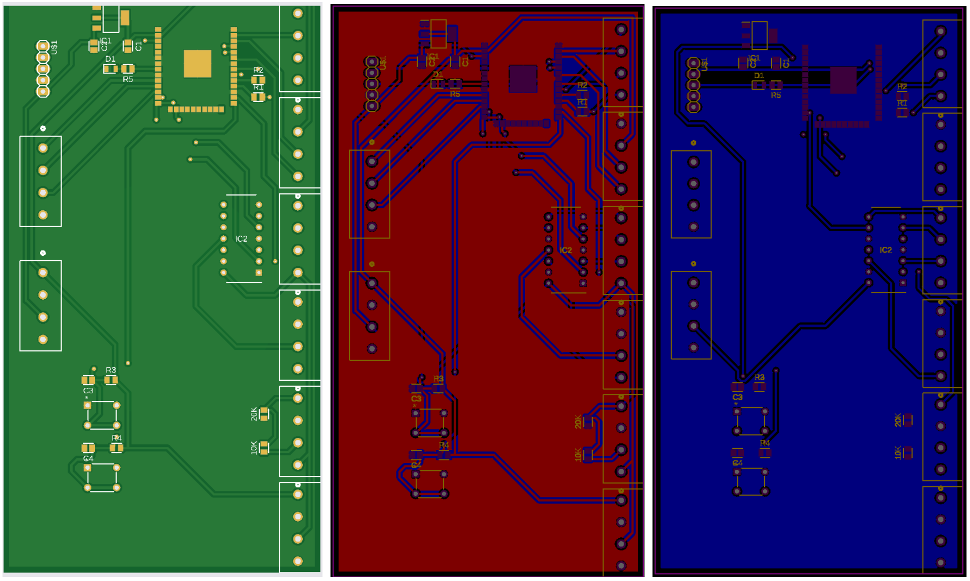


Figure 8.2.5.1-1 - MCU Board

8.2.6.1 Unified MCU Prototyping

In addition to the dedicated MCU PCB, we also designed a unified PCB that combines the schematic of the MCU with the standalone 24 to 3.3 and 3.3 to 5 volt regulators so the MCU board can be connected directly to the 24 volt power supply. In place of a 5 to 3.3

surface mount linear regulator, there is a TPS54308DDCR buck regulator to convert 24 volts to 3.3. This 3.3 line powers the MCU itself as well as some other connected devices such as the CAN controller or encoders. Connected to that is the TPS613222ADBVR boost regulator. While these two integrated regulators use the same schematic as the earlier standalone regulators used for prototyping, they do have a different layout and correction such as an inductor with a more appropriate current rating are the same. In sum total, this board is able to run all the functionalities of the microcontroller and its connected devices while shifting the voltage levels to their appropriate levels through inbuilt regulator circuits.

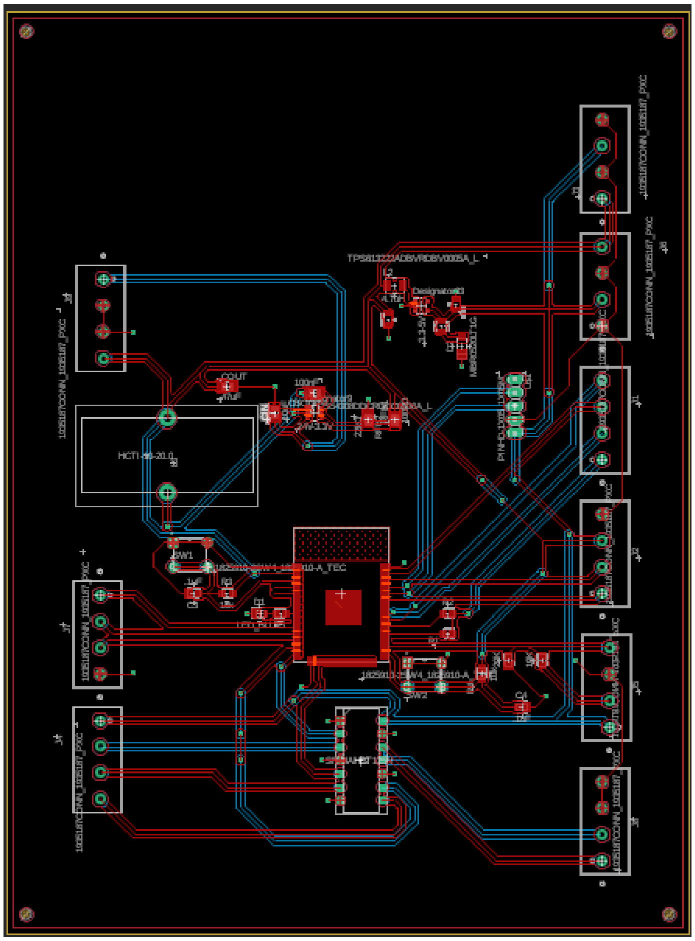


Figure 8.2.6.1-1 - Unified Regulator MCU

Chapter 9: System Testing and Evaluation

9.1 Testing Methodology

During the initial prototyping phase of our project, we were able to run tests to ensure not only the quality of our hardware, but the functionality of our systems. This also allowed us to plan for future designs, including our testing process for Senior Design 2. It should be noted that the initial tests done were very limited, and as a result are lacking in robustness, but were tested to the best of our current capabilities and access to hardware.

For each of the highlighted specification we document our testing methodology and results below, which includes both the initial process and future plans for testing specific components:

9.1.1 Motor Control Accuracy:

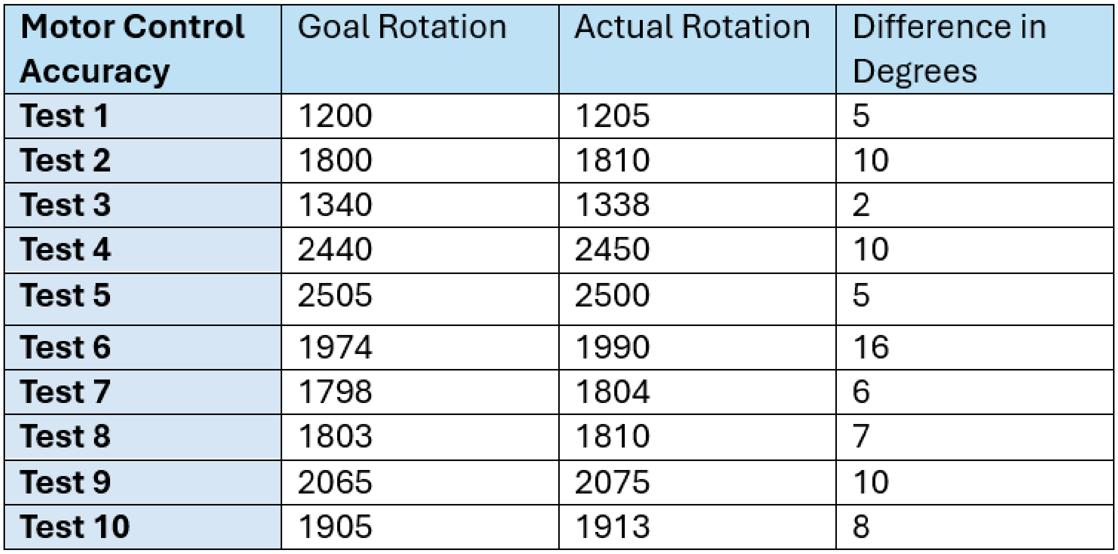
Target: ±10 degrees

Procedure:

1. Setup the motor with a high precision encoder attached to the output shaft.
2. Command the motor to move to predetermined positions ranging from 0 to 360 degrees.
3. Use software to record the encoder readings at each commanded position.
4. Analyze the difference between the commanded position and the encoder reading. Repeat the test multiple times to ensure consistency.

Results:

Table 9.1.1 - Motor Control Accuracy



Motor Control Accuracy	Goal Rotation	Actual Rotation	Difference in Degrees
Test 1	1200	1205	5
Test 2	1800	1810	10
Test 3	1340	1338	2
Test 4	2440	2450	10
Test 5	2505	2500	5
Test 6	1974	1990	16
Test 7	1798	1804	6
Test 8	1803	1810	7
Test 9	2065	2075	10
Test 10	1905	1913	8

The motor control system was rigorously tested to ensure accurate positioning and responsiveness. During the final evaluation, the motors achieved an accuracy of ±10 degrees in maintaining the desired position. Testing revealed no significant latency issues in the system, and the implementation of the Bang-Bang algorithm on the ESP32 provided consistent and reliable results. This confirms that the control system met the specified project requirements for precision.

9.1.2 Maximum Load Capacity:

Target: < 0.5kg

Procedure:

1. Attach a load incrementally to the arm until the maximum load capacity is reached.
2. Monitor and record the performance of the motor and structural integrity of the arm as load increases.

3. Observe any degradation in motor control or mechanical failure.
4. Validate that the arm can operate normally under the maximum load without any compromise in functionality.

Results:

Table 9.1.2 - Maximum Load Capacity

Load Capacity	Object	Weight	Held?
Test 1	End Effector Servo	0.039kg	Yes
Test 2	House Key Chain	0.23kg	Yes
Test 3	Hex Key	0.065kg	Yes
Test 4	Metal Krimpers	0.072kg	Yes
Test 5	Box of Metal Screws	0.045kg	Yes
Test 6	Earbud Charging Case	0.05kg	Yes
Test 7	Wallet	0.134kg	Yes
Test 8	Small Bread Board	0.132kg	Yes
Test 9	USB Adapter Hub	0.215kg	Yes
Test 10	Small Wooden Block	0.345	No

The robotic arm was tested to determine its maximum load capacity. During SD2, the arm successfully handled loads ranging from 0.01 kg to 0.3 kg without any noticeable degradation in performance. The results validated the system's structural integrity and motor torque specifications, demonstrating that the arm could reliably perform within the specified load range. This was confirmed during repetitive trials, showcasing the arm's capability to manage typical industrial tasks. The reason for not hitting a maximum of 0.5kg was due to the physical frame not being mechanically strong enough to withstand the full force of our motors' torque. While the motors are very capable, the material our robotic arm was printed with was not. Were we to have access to an aluminum 3D printer or carbon/steel, our arm would easily hold this higher weight.

9.1.3 Peak Power Draw:

Target: < 550 Watts

Procedure:

1. Operate the arm at maximum capacity using all functions simultaneously.
2. Measure the power consumption using a high accuracy power meter.
3. Record the peak power draw during operation.
4. Verify that the peak does not exceed target wattage under normal or stressed conditions.

Results:

Table 9.1.3 - Peak Power Draw

Peak Power Draw	Expected Max Watts (W)	Actual Watts (W)	Conditions
Test 1	550	36.66	First Axis
Test 2	550	34.35	Second Axis
Test 3	550	37.01	Third Axis
Test 4	550	43.85	First and Third Axis
Test 5	550	45.90	First and Second Axis
Test 6	550	44.23	Second and Third Axis
Test 7	550	43.12	All Three Axis
Test 8	550	44.46	All Three Axis and End Effector
Test 9	550	60.53	Resistance on First Axis
Test 10	550	65.34	Resistance On Second Axis

Peak power draw measurements were taken under maximum load conditions. The robotic arm's power consumption peaked at 65 watts, remaining well below the specified maximum of 550 watts. This efficiency highlights the effectiveness of the power distribution system, ensuring stable operation even during high-demand scenarios. It also shows that when the motors are not run at full capacity they are well under their maximum current ratings.

9.1.4 Response Time:

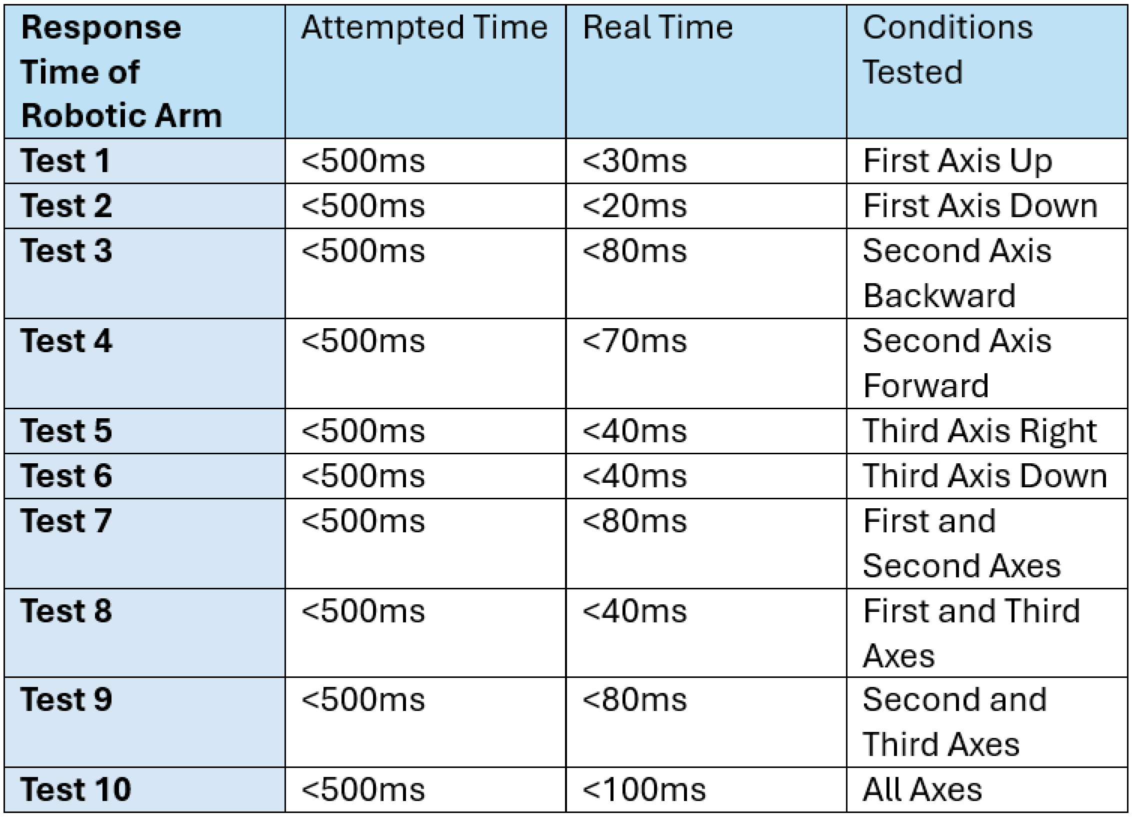
Target: < 500 ms

Procedure:

5. Operate the arm one axis at a time.
6. Use a stopwatch to measure how quickly both joystick response is sent and the robotic arm responds.
7. Do this with increasing stress to the system, adding multiple axis at a time until all three run at the same time.
8. Verify that the maximum time for all axis to respond to movement, even when running simultaneously, does not exceed 500ms.

Results:

Table 9.1.4 - Response Time



Response Time of Robotic Arm	Attempted Time	Real Time	Conditions Tested
Test 1	<500ms	<30ms	First Axis Up
Test 2	<500ms	<20ms	First Axis Down
Test 3	<500ms	<80ms	Second Axis Backward
Test 4	<500ms	<70ms	Second Axis Forward
Test 5	<500ms	<40ms	Third Axis Right
Test 6	<500ms	<40ms	Third Axis Down
Test 7	<500ms	<80ms	First and Second Axes
Test 8	<500ms	<40ms	First and Third Axes
Test 9	<500ms	<80ms	Second and Third Axes
Test 10	<500ms	<100ms	All Axes

The response time for both autonomous operation and teleoperation was measured during SD2. The system demonstrated real-time responsiveness that aligned with the project's performance criteria of 500ms, ensuring seamless control of the robotic arm. The response was near instantaneous as the joysticks operate using HID functionality and can easily be interpreted and sent to the respective motor MCU to control the motor.

9.1.5 Software Testing:

Target: Compiles and works with the robotic arm to move all motors and respond to all joystick input.

Procedure:

1. Setup functional microcontrollers to handle code.
2. Test each necessary function one by one until fully functional.
3. Integrate these systems together one by one to ensure no breaks in the code, and system functions as expected.
4. Use multiple virtual systems to verify the system's processing time and compatibility with sending data.

Results:

Software was used for running all three initially prototyped hardware. This included the joystick subsystem, the motor control, and the CAN bus communication system. The software used for each system was able to be tested using virtual modules, and in the case of our CAN communication system, virtual CAN devices. This helped significantly in the process of learning how to read and communicate over the devices via software.

The software for both our joystick subsystem and our motor control was able to be tested directly by using Visual Studio Code and a python terminal. The results of our software's output were printed directly to the terminal, giving us feedback for both the scripts ran as well as the state of our hardware connected to our software. These scripts are prototypes used to test how we would pass on information from hardware to a simulator, which will initiate control and movement within our robotic arm. Each test was successful and worked as intended.

For further testing with our motors, all functions used within our ESP32 were tested piece by piece. Each motor was tested individually with these functions, and once this testing was deemed successful, we were able to attach our joystick input to a virtual machine and transfer CAN bus data between this virtual machine and our ESP32s. Before having the Raspberry Pi available to us, this allowed us to see the full functionality of our future Ubuntu environment, and the behavior and response time of our joysticks, motors, and ESP32 system.

Chapter 10: Administrative Content

This section provides a summary of the financial and scheduling aspects related to our project. It includes details on budget estimates, the bill of materials, and work allocation for designing and constructing our system. This chapter will be further developed in Senior Design 2 as we acquire additional parts and finalize the total cost.

10.1 Budget and Financing

This section outlines the projected expenses for our project, detailing the quantity of each required item, its estimated per unit cost, and the overall anticipated expenditure. It is important to note that the specific parts listed are preliminary and subject to change as the project progresses. The table below provides an estimate of the total cost, which will be distributed among the five team members. These estimates will be updated as we finalize the components and procure additional parts.

Table 10.1-1 - Motor Subsystem Budget Table

Category	Component	Quantity	Unit Cost	Price Range
Power Supply	24V Power supply	1	\$20 - \$70	\$20 - \$70
Microcontroller	General-purpose microcontrollers	1	\$4 - \$8	\$4 - \$8
Sensors	Current sensors	3	\$1 - \$2	\$2 - \$6
Sensors	Rotary encoder	4	\$2.50 - \$10	\$10 - \$40
Motor Driver	Gate driver IC	4	\$1 - \$4	\$4 - \$16
Motor Driver	MOSFETs	32	\$0.1 - \$0.3	\$1 - \$3
Passive Components	Capacitors	>5	\$0.01 - \$0.1	\$0.1 - \$1
Passive Components	Resistors	>5	\$0.01 - \$0.1	\$0.01 - \$0.1

Category	Component	Quantity	Unit Cost	Price Range
Communication	CAN bus transceiver and receiver	1	$15 - $25	$15-$25
Connectors	Various connectors	>5	$0.01 - $0.1	$0.5 - $2
Miscellaneous	Components	-	$10 - $20	$10 - $20
Motor	2-Phase Stepper Motor	4	$5 - $15	$20 - $60
Total Cost		$67.6 - $198.09		

Below, we present the budget specifically allocated for our joystick subsystem. This table outlines the expected costs and quantities of the components necessary for the development and integration of the joystick control unit within our project.

Table 10.1-2 - Joystick Subsystem Budget Table

Category	Component	Quantity	Unit Cost	Price Range
Microcontroller	General-purpose microcontroller	1	$4 - $8	$4 - $8
Communication	USB Cable	1	$1 - $5	$1 - $5
Communication	USB Cable	1	$1 - $5	$1 - $5
User Interface	Joysticks	2	$2.50 - $7.50	$5 - $15
User Interface	Buttons	4	$0.1 - $1	$0.4 - $4
Connectors	Various connectors	>5	$0.5 - $15	$0.5 - $15
Passive Components	Capacitors	>5	$0.01 - $0.10	$0.1 - $1

Category	Component	Quantity	Unit Cost	Price Range
Passive Components	Resistors	>5	$0.01 - $0.10	$0.01 - $1
PCB	Custom Final PCB	1	$10-$50	$10 - $50
Miscellaneous	Heat sinks, mounting hardware, etc.	-	$5-15	$5 - $15
Total Cost		$27 - $119		

10.2 Bill of Materials

Below we have the bill of materials for components actually purchased. This list includes items used for both prototyping and the final PCB, and will be updated and continued into Senior Design 2 as we progress and finalize the PCB. The table provides a comprehensive overview of the necessary components and their costs, reflecting our ongoing development. It is separated into a table for the Motor Subsystem and the Joystick Subsystem. This approach ensures transparency in our budgeting process and allows for efficient tracking of expenses as we advance through the project stages. By meticulously documenting each component, we aim to streamline the procurement and assembly processes, reducing potential delays and cost overruns.

Table 10.2-1 - Bill of Materials for Motor Subsystem

Component	Supplier	Quantity	Unit Cost	Total Cost	Notes
Stepper Motor	Stepper Online	4	$23.80	$95.20	NEMA 23 2.4NM
Rotary Encoder	DigiKey	4	$5.76	$23.04	Hall Effect position sensors

Component	**Supplier**	**Quantity**	**Unit Cost**	**Total Cost**	**Notes**
Current Sensor	DigiKey	8	$5.69	$45.52	Hall Effect current sensor
ESP32	DigiKey	4	$2.50	$7.50	Microcontroller for motor subsystem
CAN Bus Controller	Digikey	4	$2.44	$9.76	CAN bus controllers to communicate with transceivers
CAN Bus Transceiver	Digikey	4	$1.52	$5.54	Transceivers translate signals so RP2040
CAN Bus Reference Board	Cyberebee	1	$2.95	$2.95	Development board for reference
Power Supply	Amazon	1	$29.99	$29.99	AC-DC Power Supply
Parallel Gripper with servo	Digikey	1	$36.24	$36.24	End effector
Raspberry Pi 5	Owned	1	$0	$0	CPU Simulator
Motor PCB	JLCPCB	3	8.96	36	Custom PCB for each motor.
Total Cost			$261.36		

Table 10.2-2 - Bill of Materials for Joystick Subsystem

Component	Supplier	Quantity	Unit Cost	Total Cost	Notes
Analog Joystick	Amazon	8	$1.62	$12.99	Generic Analog Joystick. Using 2.
Mechanical Buttons	Amazon	100	$0.06	$5.99	4-pin buttons for reliability. Using 4.
Arduino Leonardo	Amazon	1	$24.99	$24.99	Microcontroller with HID capabilities.
USB-A to USB-B Connection	Owned	1	$0	$0	We have the USB cable for Arduino on hand.
Joystick PCB	JLCPCB	5	$16.82	$84.09	Custom PCB for joystick controller.
Total Cost			$128.06		

10.3 Project Milestones

10.3.1 Milestones for SD1:

Table 10.3.1 - Senior Design 1 Milestones

Milestone Category	Milestones	Due Date
Project Planning and Initial Research	- Finalize project scope and objectives - Conduct a thorough literature review on current robotic arms and FOC motor controllers - Identify key technologies and components required for the prototype	05/17/24
Divide & Conquer	- Turn in initial 10-page Divide and Conquer Document	05/31/24

Milestone Category	Milestones	Due Date
Conceptual Design and System Architecture	- Develop the conceptual design of the robotic arm - Create detailed system architecture diagrams, including hardware and software components - Define the control algorithms and kinematics for the arm	05/31/24
Component Selection and Procurement	- Select and procure motors, sensors, controllers, and other essential components - Ensure all components meet the basic requirements for the prototype	05/31/24
Mechanical Design and Fabrication	- Design the mechanical structure of the multi-axis arm - Fabricate and assemble the mechanical components for integration in SD2	06/21/24
75-Page Draft	- Turn in 75-page draft for final report encompassing chapter 2,3,4,5, and 10	07/08/24
First Iteration of PCB Design and Fabrication	- Design the first iteration of the PCB for motor control and communication - Wire all necessary components on a Fusion 360 and ensure the circuit is functional and meets datasheet requirements	07/09/24
Communication Setup	- Establish communication from all motors back to a central computer - Implement protocols for data exchange and control commands - Test the communication system for reliability and responsiveness	07/09/24

Milestone Category	Milestones	Due Date
Software Development and Testing	- Develop the basic software for controlling the arm, including position, velocity, and torque control - Implement forward and inverse kinematics algorithms - Begin preliminary testing of the control system and software	07/16/24
Functioning Arm Prototype	- Finalize the assembly and test major components - Ensure the prototype meets basic functional requirements, even if it does not hit all specifications - Document the performance and any deviations from the initial specifications	07/23/24
Final 150-Page SD1 Report and Prototype Video	- Turn in final report for SD1 consisting of all chapters and 150 pages - Turn in prototype video showing major working components	07/23/24

10.3.2 Milestones for SD2:

Table 10.3.2 - Senior Design 2 Milestones

Milestone Category	Milestones	Due Date
Finalize PCB Fabrication	- Finish designing and finalize PCB layouts - Send PCBs for fabrication, at least 3 - Solder components onto the fabricated PCBs	08/20/24
System Integration	- Connect PCBs to the Raspberry Pi - Integrate motor drivers, sensors, and power supply - Test communication from PCBs and Raspberry Pi	09/01/24

Milestone Category	Milestones	Due Date
Robot Arm Construction	- Assemble the robotic arm and mechanical components - Connect motors to the robotic arm and power it	09/09/24
Final System Testing	- Conduct comprehensive testing of the entire system - Refine and debug as necessary, ensuring all components and systems are working together seamlessly	10/01/24
Review and Evaluation	- Prepare for final review(s) and demonstration(s)	11/26/24

Chapter 11: Conclusion

Robotic arms have been integral to automated industries for decades, predominantly designed for versatility rather than specific, tailored applications. This approach allows users to select different end effectors for varied tasks, but it also means that once deployed, not enough consideration is given to specific operational safety beyond basic functionality. Automation has become formulaic with not much thought to it beyond how quickly products can be produced, and while safety has not gone completely neglected as a feature, this mindset has led to some level of poor quality control in terms of danger for the operators. The industry standard for automated robotic arms has historically neglected to account for force the arms can reach, and how they move through obstacles that may be in their path. This lack of precautions has led to horrific incidents that have largely gone unchecked by countless corporations across the globe. This oversight has resulted in fatal incidents due to inadequate safety measures, highlighting a critical need for improvements in industry standards.

Our project addresses these concerns by integrating a system in which these robotic arms can learn, take in human operator input, and are programmed with existing limits that prevent each motor from reaching its maximum force and torque used. By using a robotic simulator alongside our actual robotic arm, we are able to show the movement and accuracy of our motors to our human operators in real time, which can allow them to monitor the behavior of our design, and stop it if need-be with the added features of an emergency stop button on our joystick. The addition of a closed-looped system means that our robotic arm will return to a “home” position, allowing operators to release control of the robotic arm without damage to them or our design.

The automation industry is run on the mindset of how quickly products can be produced, being controlled by a need to keep up with a fast-paced market. Due to this, speed and reliability were paramount to our design. When these features are kept in mind, the inclusion of safety features like an emergency stop and a closed-loop system may initially seem counterintuitive. However, it is far more beneficial to the engineers who work with these robotic behemoths to take precautions. Safeguarding the engineers who operate alongside these machines is essential for long-term productivity and well-being. It is worth noting that the overall efficiency of our robotic arm is not slowed significantly due to these safety features, and the benefits of allowing the robotic arm to detect resistance and to slow its speed far outweigh the need for expedited automation. Our robotic arm was built with the intention of striking a balance between efficiency and safety, ensuring that it could operate smoothly while reducing the likelihood of accidents.

When it comes to automated design, the added features are minimal, but make a very large difference in terms of reliability and safety. Utilizing a closed-loop system with our motors, facilitated by microcontrollers, current sensors, and encoders, allowed us to implement a more reliable system. These technologies enable precise adjustments in motor currents, optimizing performance without compromising safety. These algorithms are the key to allowing our robotic arm to slow its movement when it detects resistance, or an object that it can not move through. Designing the robotic arm with specific limit control was crucial for maintaining consistency and reliability in demanding industrial environments, where deviations in operation could lead to costly downtime or safety incidents.

By using a CAN structure attached to our motor architecture, a communication system that is often used in automation, our motor system was able to send information between joints and an active simulator that runs in real-time, designed to further catch errors and correct them before the arm has a chance to make them. With the use of microcontrollers, we were able to precisely control the angle and speed of our joint motors, allowing any error correction to happen efficiently and far before any devastating failure is experienced.

Moreover, our custom joystick subsystem further enhanced operational flexibility by enabling intuitive teleoperation and advanced task automation.. The teleoperative device used dual joysticks for precise multi-axis control, supplemented by essential functions such as emergency stops and task recording for repetitive operations. For the operator, this means a more intuitive interface and ease-of-access with the robotic design. By functioning in this manner, it allows operators to initiate tasks in more efficient and smoother ways than if the robotic arm had just been pre-programmed with a set level of operation, or designed for specific motions.

While our design leverages common industry practices for end effector versatility, our implementation distinguishes itself through intelligent motor management. By enabling the robotic arm to detect motor resistance and adjust operations accordingly, we enhance its capability to handle objects efficiently and safely. The resistive detection of our motor architecture allows our robotic arm to “detect” if an object is too heavy to pick up without the need for precise sensors to read the weight of an object, or a camera to detect and identify correctly the object the arm is encountering.

In conclusion, our project is designed to account for the shortcomings of a widely formulaic industry in the form of a more safety-oriented design. The focus of our project on specific control over motors greatly enhances the ability of an automated tool to operate safely and reliably. By implementing these features, our design highlights major flaws within a well-established system that could be corrected easily, and without the use of invention on the part of engineering and design teams. While these algorithms and hardware designs are by no means easy, the work it takes to implement them is crucial and necessary to demonstrate the weaknesses of the industrial automotive field, and how it could be better designed in the future.

Appendices

Appendix A – References

[1] "Hall Effect Joysticks," iFixit, [Online]. Available: https://www.ifixit.com/Wiki/Hall-Effect_Joysticks. [Accessed: 12-Jun-2024]

[2] "Hall Effect Sensor," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Hall_effect_sensor. [Accessed: 12-Jun-2024]

[3] "Potentiometer," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Potentiometer. [Accessed: 12-Jun-2024]

[4] "What is a Hall Effect Joystick and Why Don't They Develop Drift?," How-To Geek, [Online]. Available: https://www.howtogeek.com/829622/what-is-a-hall-effect-joystick-and-why-dont-they-develop-drift/. [Accessed: 12-Jun-2024]

[5] Harte, C., "Design and Evaluation of a Low-Cost Three-Axis Hall-Effect Joystick," JSAN, vol. 2, no. 1, pp. 85-96, 2013. [Online]. Available: https://www.mdpi.com/2227-7080/2/1/85/pdf. [Accessed: 12-Jun-2024]

[6] FANUC America Corporation, Fanuc Robots By Series, M800iA/60 Robot | FANUC America, FANUC America Corporation 2024

[7] ABB, ABB Robots Aid Rapid Automated Testing For COVID19 Virus, ABB robots aid rapid automated testing for COVID19 virus, ABB Corporation 2021

[8] "Is Google Getting Worse? This is What Leading Computer Scientists Say," Fast Company, [Online]. Available:https://www.fastcompany.com/91012311/is-google-getting-worse-this-is-what-leading-computer-scientists-say. [Accessed: 12-Jun-2024]

[9] "Google Search SEO," The Verge, [Online]. Available:https://www.theverge.com/2024/5/2/24147152/google-search-seo-publishing-housefresh-product-reviews. [Accessed: 12-Jun-2024]

[10] Millett, Peter. “Choosing the MOSFET Drivers for Motion Control.” Power Electronics News, 1 June 2020, www.powerelectronicsnews.com/choosing-the-mosfet-drivers-for-motion-control/#:~:text=Choosing%20the%20MOSFETs,higher%20than%20the%20supply%20voltage.

[12] “Using Power Mosfets in DC Motor Control Applications.” Nexperia, www.nexperia.com/applications/interactive-app-notes/IAN50004_using-power-MOSFETs-in-DC-motor-control-applications#:~:text=Here%20the%20MOSFETs%20are%20switched,provide%20the%20desired%20voltage%20polarity. Accessed 4 July 2024.

[13] “What Is a MOSFET?: Toshiba Electronic Devices & Storage Corporation: Americas – United States.” Toshiba Electronic Devices & Storage Corporation | Americas – United States, toshiba.semicon-storage.com/us/semiconductor/knowledge/faq/mosfet_common/what-is-a-mosfet.html#:~:text=MOSFET%20stands%20for%20metal-oxide,and%20source%20(S)%20terminals. Accessed 3 July 2024.

[14] Fiore, Carimine. “Stepper Motors Basics: Types, Uses, and Working Principles.” Monolithicpower, Available: www.monolithicpower.com/learning/resources/stepper-motors-basics-types-uses. Accessed 4 July 2024.

[15] MOSFET - Basics of MOSFET, Operation, Types, Applications, Faqs, byjus.com/physics/mosfet/. Accessed 4 July 2024.

[16] MathWorks Corporation, “Model and Control a Robotic Arm Using Simscape”, https://www.mathworks.com/help/robotics/ug/model-and-control-a-manipulator-arm-with-simscape.html, 2024 The MathWorks Inc.

[17] MathWorks Corporation, “Robotics System Toolbox Documentation”, https://www.mathworks.com/help/robotics/index.html, 2024 The MathWorks Inc.

[18] MathWorks Corporation, “Simscape Documentation”, https://www.mathworks.com/help/simscape/index.html, 2024 The MathWorks Inc.

[19] Madhaiyan, Bhuvana, “Introduction to Gate Drivers in Power Electronics”, The Talema Group, Available: https://talema.com/gate-driver-introduction/, Accessed July 3rd 2024.

[20] "Response from ChatGPT," OpenAI, [Online]. Available: chat.openai.com. [Accessed: 20-June-2024]

[21] Andrade, Raymond. What Is ROS2? - Framework Overview. A brief description of the ROS2 framework for beginners. 18 June 2021. Youtube, https://www.youtube.com/watch?v=7TVWlADXwRw.

Appendix B – ChatGPT Prompt Examples

““Prompt: explain the pros and cons between CUI - AMT10, REV Through Bore Encoder, and MagAlpha - MA702.

ChatGPT: Certainly! Let's compare the CUI AMT10-REV and the MagAlpha MA702 encoders based on their pros and cons:
CUI AMT10-REV Through Bore Encoder …
Pros:
1. Cost-Effective: Generally more affordable compared to magnetic encoders like the MagAlphaMA702.””

Prompt: What kind of AC-DC power supply would be good for a robot arm? Provide an example on picking a good power supply.

ChatGPT: When selecting an AC-DC power supply for a robot arm, several factors need to be considered to ensure it meets the requirements of your specific application. Here are the key considerations and an example of how to pick a good power supply:

Voltage Requirements:Determine the voltage rating required by the robot arm. This information can typically be found in the robot arm's specifications.

Current Requirements: Calculate the total current draw of all the components in the robot arm, including motors, controllers, sensors, and any other peripherals.

Power Rating:The power supply should be capable of delivering sufficient power (Wattage), calculated as Voltage × Current.

Safety Margins: Add a safety margin of 20-30% to the total power requirement to ensure reliable operation and to accommodate any unexpected power surges.

Form Factor: Ensure the physical dimensions and mounting style of the power supply are suitable for the installation space in your robot arm setup.

Regulation and Ripple: Look for power supplies with good voltage regulation and low ripple/noise characteristics to ensure stable operation of sensitive electronics.

Efficiency: Higher efficiency power supplies generate less heat and are more reliable.

Certification: Choose power supplies with necessary safety and compliance certifications (e.g., CE, UL).

Example:Let's go through an example to illustrate how to pick a good power supply for a hypothetical robot arm.

Specifications of the Robot Arm; Voltage: 24V DC

Current: Motor 1: 2A, Motor 2: 1.5A. Motor 3: 1A

Controller and Sensors: 0.5A

Total Current: 5A (2A + 1.5A + 1A + 0.5A)

Power Calculation; Total Power Requirement:24V×5A=120W

Adding a 30% safety margin: 120W×1.3=156W

Selecting the Power Supply; Based on the above requirements, a suitable power supply would be:

Mean Well LRS-200-24

Voltage Output: 24V DC

Current Output: 8.5A

Power Output: 204W

Efficiency: 89%

Certifications: UL, CE

Form Factor: Compact design with screw terminal connections

Features; High Efficiency: Ensures less heat generation and better reliability.

Overload Protection: Protects the power supply and connected components from damage due to excessive current.

Cooling by Free Air Convection: No fan needed, reducing noise and potential failure points.

Compact Size: Easy to integrate into various robot arm setups.

Conclusion: When selecting a power supply for a robot arm, ensure it meets the voltage and current requirements, includes a safety margin, has good efficiency and regulation, and fits the physical constraints of your setup. The Mean Well LRS-200-24 is an excellent example of a power supply that meets these criteria.

Prompt: Out of the four kinds of current sensors (Hall Effect, Shunt Resistors, Rogowski Coil, Current Transformer), which one would be best for a Robot Arm? Go into detail

ChatGPT: Choosing the best current sensor for your robot arm depends on several factors, including accuracy, size, cost, ease of integration, and response time. Here is a detailed comparison of the four types of current sensors:

1. Hall Effect Sensors; Description: Hall effect sensors use the Hall effect to measure the magnetic field generated by the current flowing through a conductor.

Advantages; Non-invasive: They can measure current without making direct electrical contact.

Isolated: Provides electrical isolation between the sensor and the measured current, enhancing safety.

Wide range: Suitable for both AC and DC current measurements.

Response time: Fast response time, suitable for dynamic measurements.

Disadvantages:Temperature sensitivity: Performance can be affected by temperature variations.

Cost: Typically more expensive than shunt resistors.

Suitability for Robot Arm: Hall effect sensors are well-suited for your robot arm due to their non-invasive nature, electrical isolation, and ability to measure both AC and DC currents. These features provide flexibility and safety, which are important for robotic applications.

2. Shunt Resistors; Description: Shunt resistors measure current by detecting the voltage drop across a known resistance.

Advantages;Accuracy: Highly accurate for both AC and DC current measurements.

Cost: Generally low cost.

Size: Compact and easy to integrate into circuits.

Disadvantages; Power dissipation: Can generate heat due to power loss (I^2R loss).

Non-isolated: Direct electrical contact with the circuit, which can be a safety concern.

Suitability for Robot Arm; Shunt resistors are suitable for applications where high accuracy and low cost are priorities. However, the lack of electrical isolation and power dissipation might be concerns in a robot arm application.

3. Rogowski Coil; Description: Rogowski coils are used to measure AC currents by detecting the rate of change of the magnetic field.

Advantages; Non-invasive: No direct electrical contact with the conductor.

Wide range: Capable of measuring a wide range of AC currents.

Lightweight and flexible: Easy to install around conductors of various shapes and sizes.

Isolated: Provides electrical isolation.

Disadvantages; AC only: Cannot measure DC currents.

Signal processing: Requires integrator circuitry to obtain current measurement.

Suitability for Robot Arm; Rogowski coils are not suitable for this application because they cannot measure DC currents, which are necessary for your 24VDC robot arm.

4. Current Transformer (CT); Description: Current transformers are used to measure AC currents by stepping down the current to a lower, more manageable level.

Advantages; Non-invasive: No direct electrical contact with the conductor.

Isolated: Provides electrical isolation.

Durability: Robust and reliable for long-term use.

Disadvantages; AC only: Cannot measure DC currents.

Size: Typically bulkier than other types of sensors.

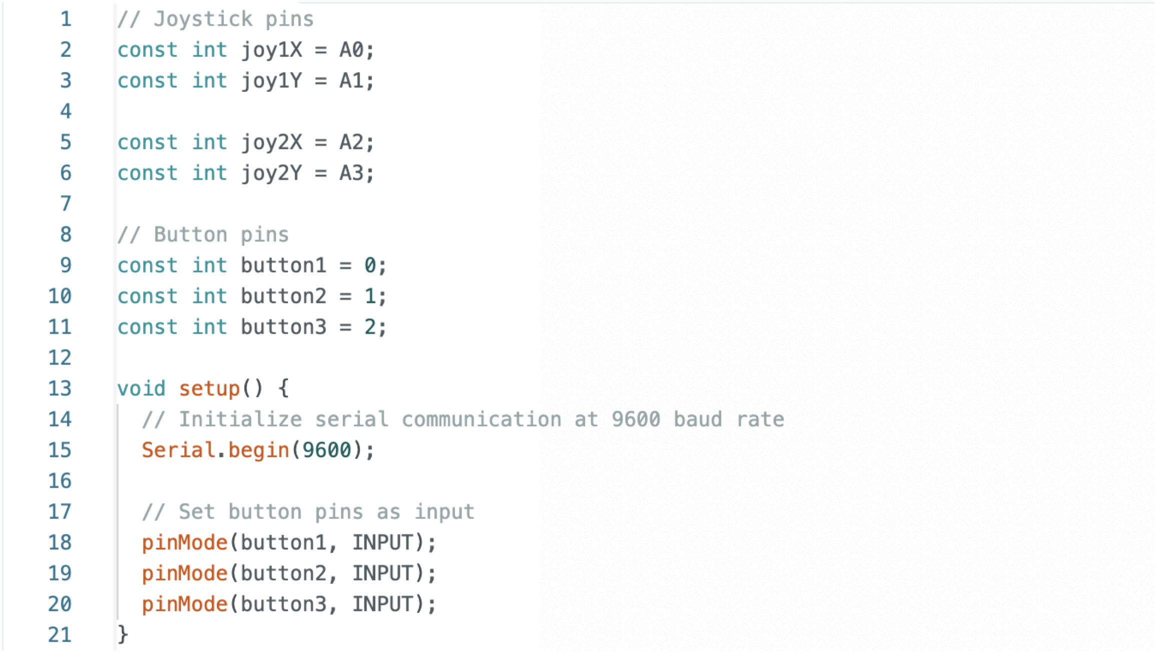
Limited range: Designed for specific current ranges.

Suitability for Robot Arm: Current transformers are not suitable for this application because they cannot measure DC currents.

Recommendation: For your robot arm, the Hall Effect Sensor is the best choice. It provides the necessary electrical isolation, can measure both AC and DC currents, and has a fast response time suitable for dynamic measurements in robotic applications. The non-invasive nature and robustness of Hall effect sensors make them ideal for integration into a robot arm, ensuring safety and reliability.

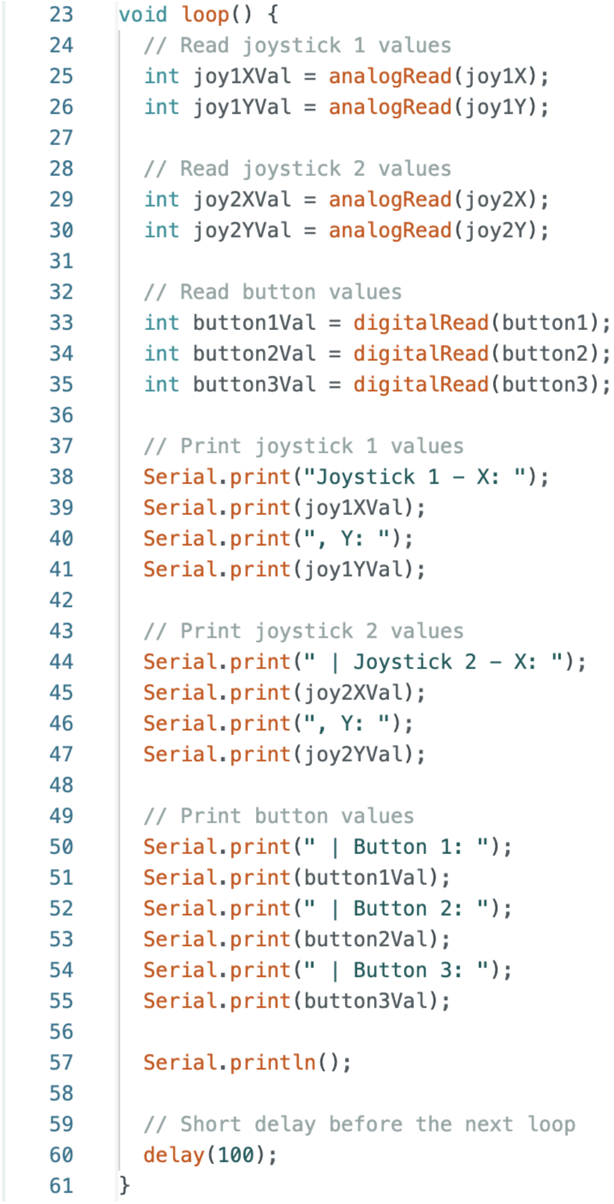
Appendix C – Programming Code

C.1 - Arduino Code for Joystick and Button Prototyping:



```
// Joystick pins
const int joy1X = A0;
const int joy1Y = A1;

const int joy2X = A2;
const int joy2Y = A3;

// Button pins
const int button1 = 0;
const int button2 = 1;
const int button3 = 2;

void setup() {
  // Initialize serial communication at 9600 baud rate
  Serial.begin(9600);

  // Set button pins as input
  pinMode(button1, INPUT);
  pinMode(button2, INPUT);
  pinMode(button3, INPUT);
}
```

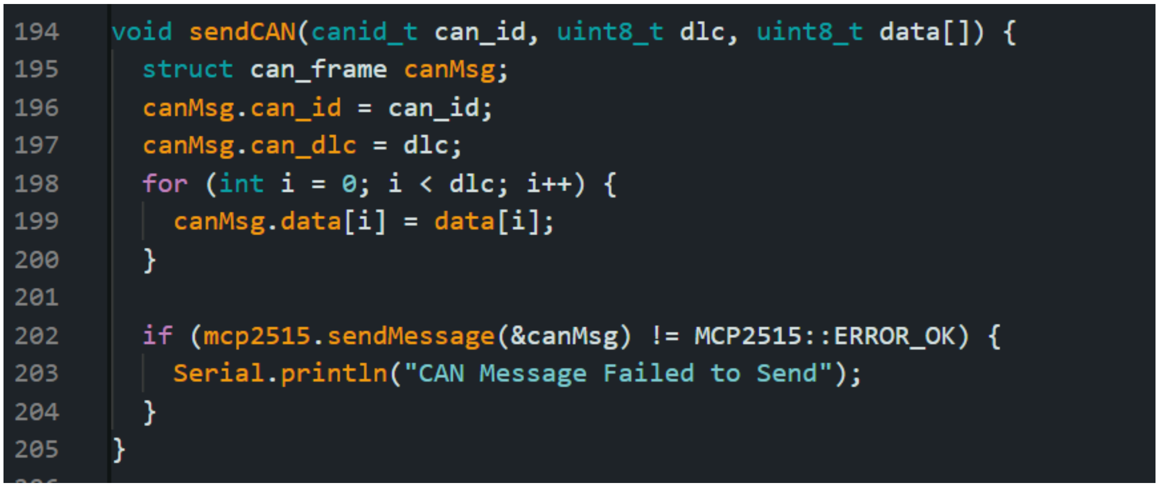
Figure C.1-1 - Setup code for digital and analog inputs on Arduino IDE



```
void loop() {
  // Read joystick 1 values
  int joy1XVal = analogRead(joy1X);
  int joy1YVal = analogRead(joy1Y);

  // Read joystick 2 values
  int joy2XVal = analogRead(joy2X);
  int joy2YVal = analogRead(joy2Y);

  // Read button values
  int button1Val = digitalRead(button1);
  int button2Val = digitalRead(button2);
  int button3Val = digitalRead(button3);

  // Print joystick 1 values
  Serial.print("Joystick 1 - X: ");
  Serial.print(joy1XVal);
  Serial.print(", Y: ");
  Serial.print(joy1YVal);

  // Print joystick 2 values
  Serial.print(" | Joystick 2 - X: ");
  Serial.print(joy2XVal);
  Serial.print(", Y: ");
  Serial.print(joy2YVal);

  // Print button values
  Serial.print(" | Button 1: ");
  Serial.print(button1Val);
  Serial.print(" | Button 2: ");
  Serial.print(button2Val);
  Serial.print(" | Button 3: ");
  Serial.print(button3Val);

  Serial.println();

  // Short delay before the next loop
  delay(100);
}
```

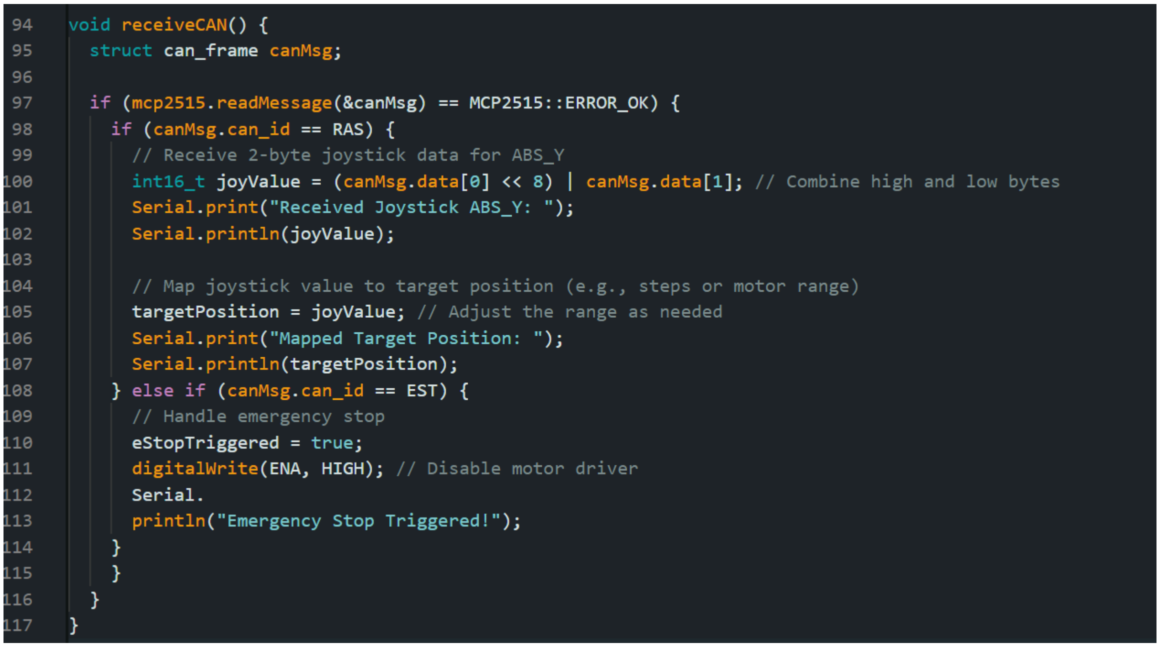
Figure C.1-2 - Running loop to test user inputs on Arduino IDE

C.2 Microcontroller and CAN Repository



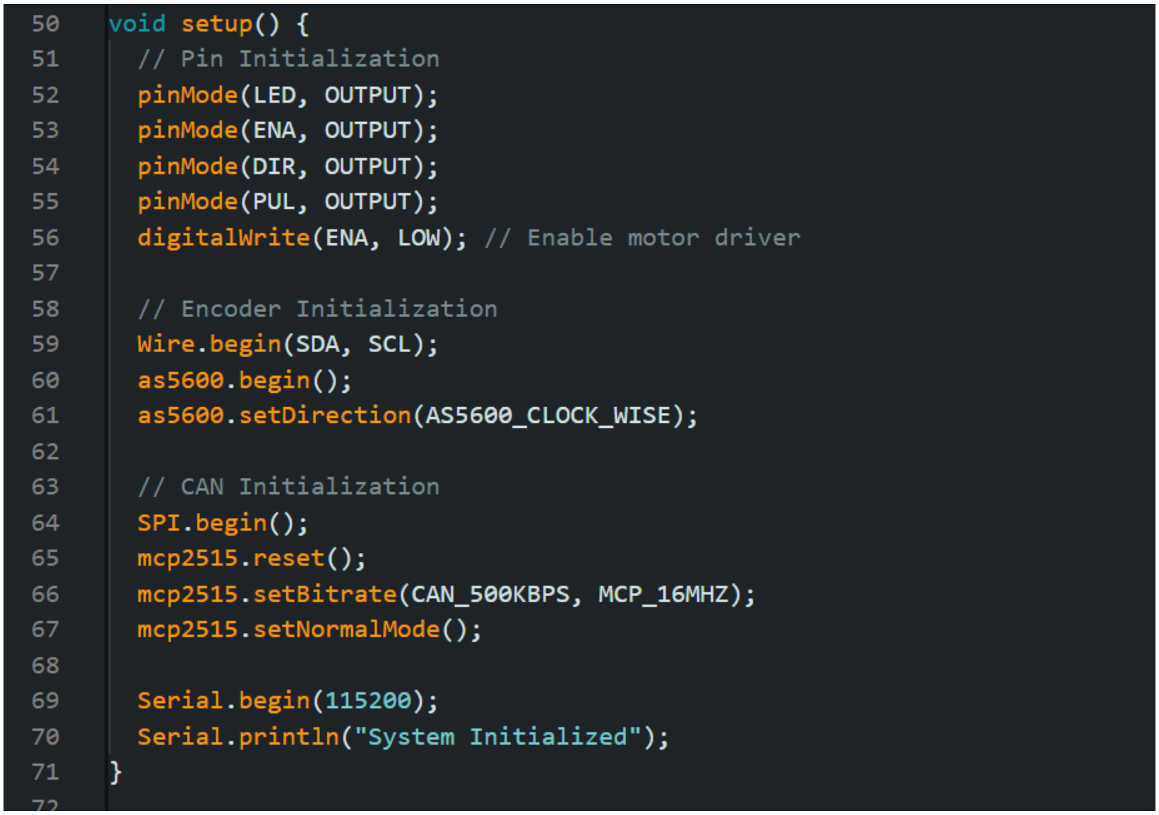
```
void sendCAN(canid_t can_id, uint8_t dlc, uint8_t data[]) {
  struct can_frame canMsg;
  canMsg.can_id = can_id;
  canMsg.can_dlc = dlc;
  for (int i = 0; i < dlc; i++) {
    canMsg.data[i] = data[i];
  }

  if (mcp2515.sendMessage(&canMsg) != MCP2515::ERROR_OK) {
    Serial.println("CAN Message Failed to Send");
  }
}
```

Figure C.2-1 - Code used to send messages across our CAN network for all nodes



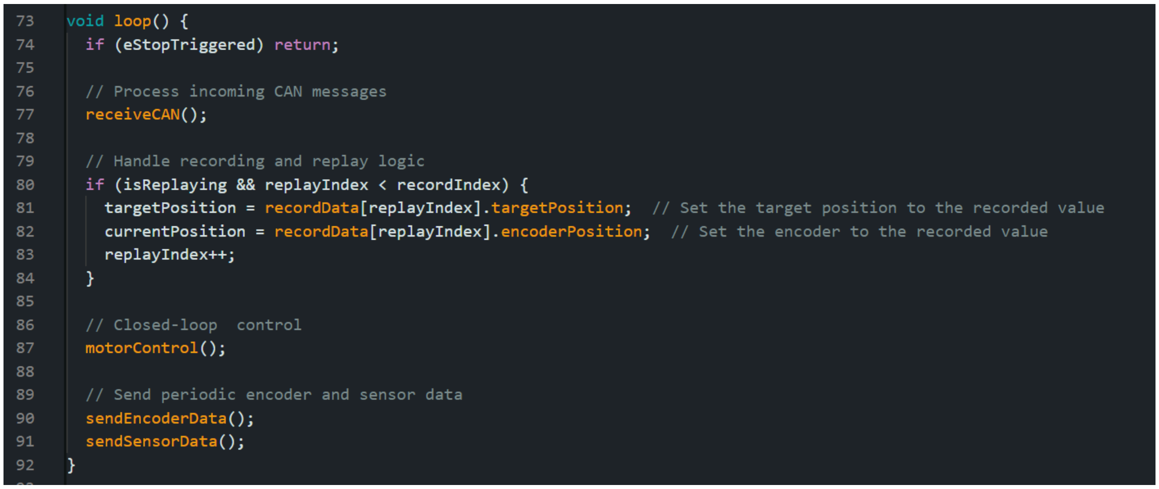
```
void receiveCAN() {
  struct can_frame canMsg;

  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {
    if (canMsg.can_id == RAS) {
      // Receive 2-byte joystick data for ABS_Y
      int16_t joyValue = (canMsg.data[0] << 8) | canMsg.data[1]; // Combine high and low bytes
      Serial.print("Received Joystick ABS_Y: ");
      Serial.println(joyValue);

      // Map joystick value to target position (e.g., steps or motor range)
      targetPosition = joyValue; // Adjust the range as needed
      Serial.print("Mapped Target Position: ");
      Serial.println(targetPosition);
    } else if (canMsg.can_id == EST) {
      // Handle emergency stop
      eStopTriggered = true;
      digitalWrite(ENA, HIGH); // Disable motor driver
      Serial.
      println("Emergency Stop Triggered!");
    }
    }
  }
}
```

Figure C.2-2 - Code used to receive messages across our CAN network for all nodes



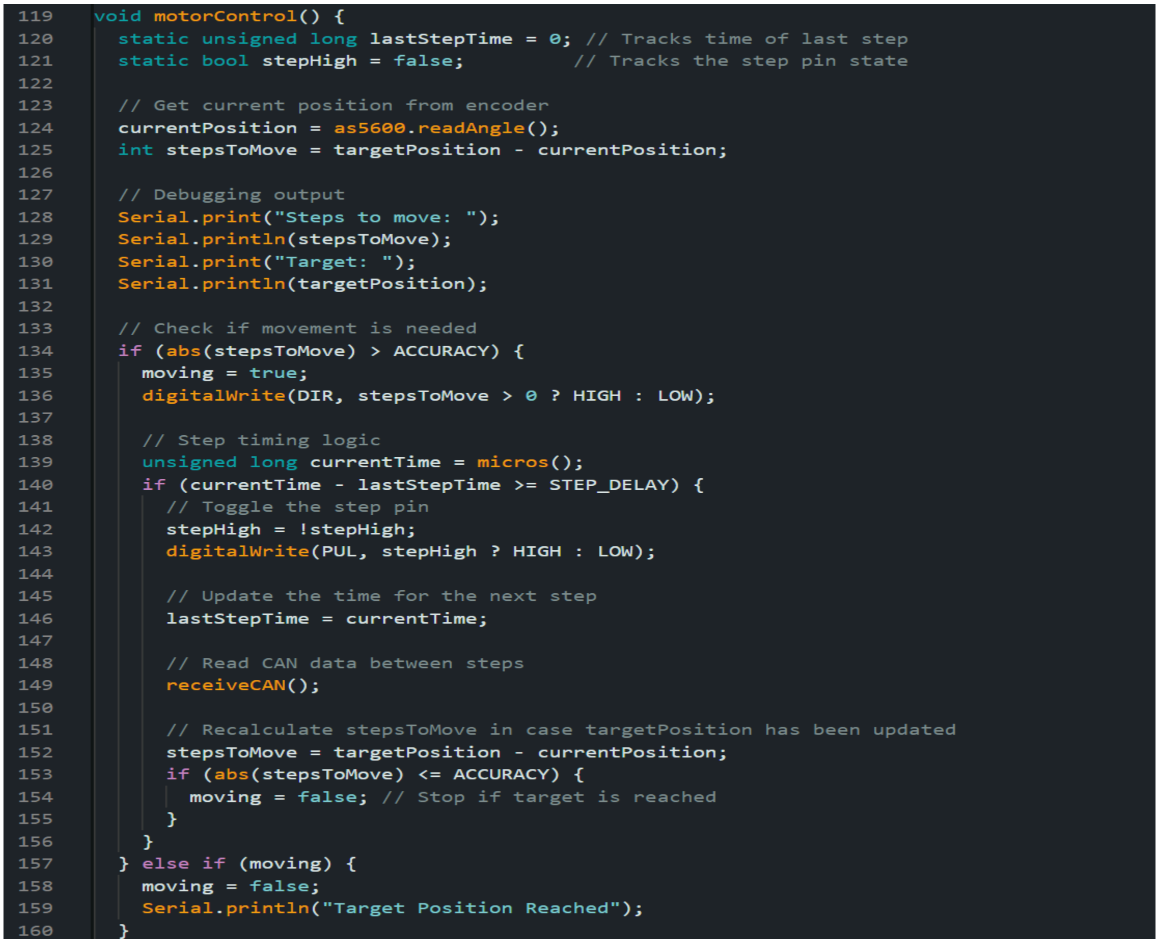
```
void setup() {
  // Pin Initialization
  pinMode(LED, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(DIR, OUTPUT);
  pinMode(PUL, OUTPUT);
  digitalWrite(ENA, LOW); // Enable motor driver

  // Encoder Initialization
  Wire.begin(SDA, SCL);
  as5600.begin();
  as5600.setDirection(AS5600_CLOCK_WISE);

  // CAN Initialization
  SPI.begin();
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS, MCP_16MHZ);
  mcp2515.setNormalMode();

  Serial.begin(115200);
  Serial.println("System Initialized");
}
```

Figure C.2-3 Setup() code for all motor nodes



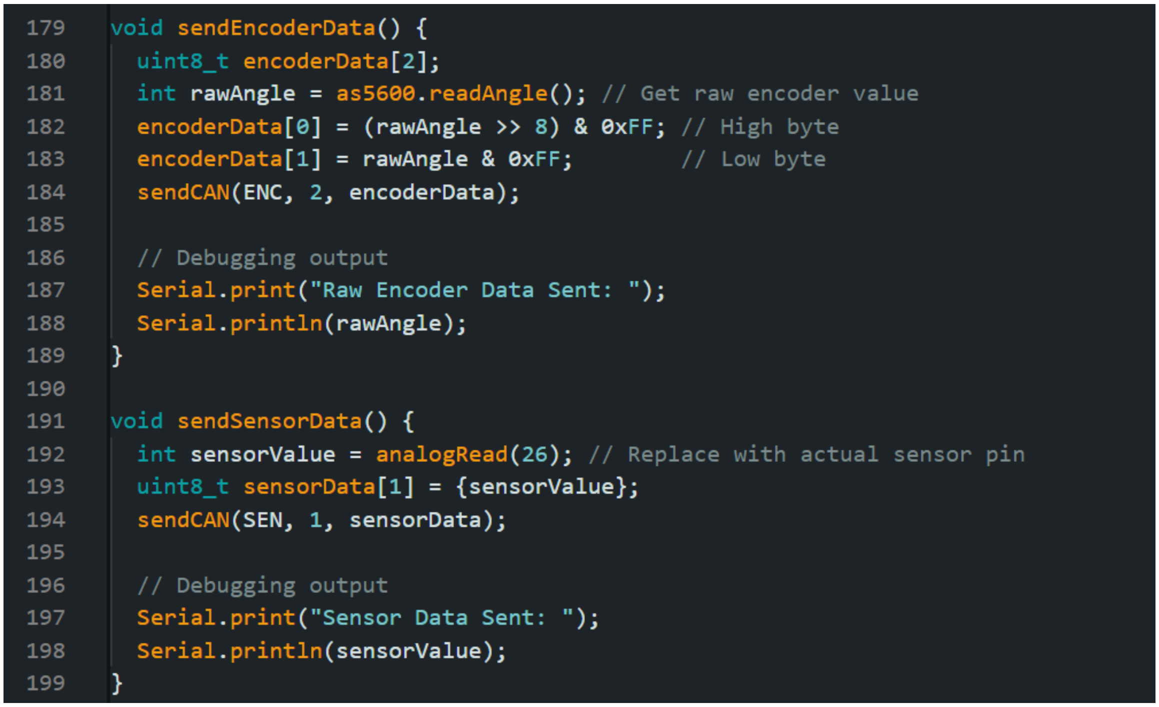
```
void loop() {
  if (eStopTriggered) return;

  // Process incoming CAN messages
  receiveCAN();

  // Handle recording and replay logic
  if (isReplaying && replayIndex < recordIndex) {
    targetPosition = recordData[replayIndex].targetPosition;  // Set the target position to the recorded value
    currentPosition = recordData[replayIndex].encoderPosition;  // Set the encoder to the recorded value
    replayIndex++;
  }

  // Closed-loop  control
  motorControl();

  // Send periodic encoder and sensor data
  sendEncoderData();
  sendSensorData();
}
```

Figure C.2-4 - Loop() used for nodes 1 and 3



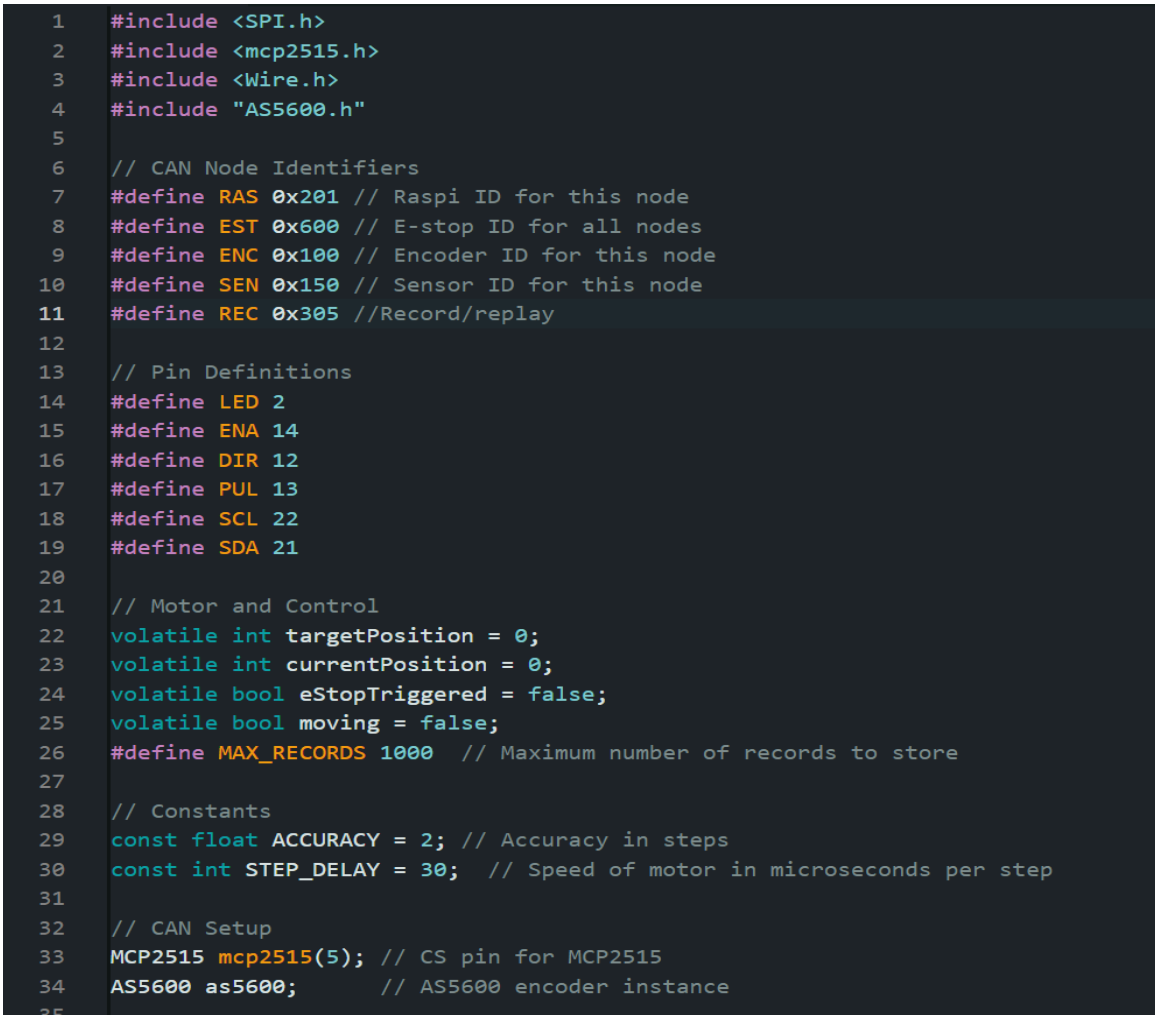
```
void motorControl() {
  static unsigned long lastStepTime = 0; // Tracks time of last step
  static bool stepHigh = false;         // Tracks the step pin state

  // Get current position from encoder
  currentPosition = as5600.readAngle();
  int stepsToMove = targetPosition - currentPosition;

  // Debugging output
  Serial.print("Steps to move: ");
  Serial.println(stepsToMove);
  Serial.print("Target: ");
  Serial.println(targetPosition);

  // Check if movement is needed
  if (abs(stepsToMove) > ACCURACY) {
    moving = true;
    digitalWrite(DIR, stepsToMove > 0 ? HIGH : LOW);

    // Step timing logic
    unsigned long currentTime = micros();
    if (currentTime - lastStepTime >= STEP_DELAY) {
      // Toggle the step pin
      stepHigh = !stepHigh;
      digitalWrite(PUL, stepHigh ? HIGH : LOW);

      // Update the time for the next step
      lastStepTime = currentTime;

      // Read CAN data between steps
      receiveCAN();

      // Recalculate stepsToMove in case targetPosition has been updated
      stepsToMove = targetPosition - currentPosition;
      if (abs(stepsToMove) <= ACCURACY) {
        moving = false; // Stop if target is reached
      }
    }
  } else if (moving) {
    moving = false;
    Serial.println("Target Position Reached");
  }
```

Figure C.2-5 - Motor driving function for nodes 1 and 3



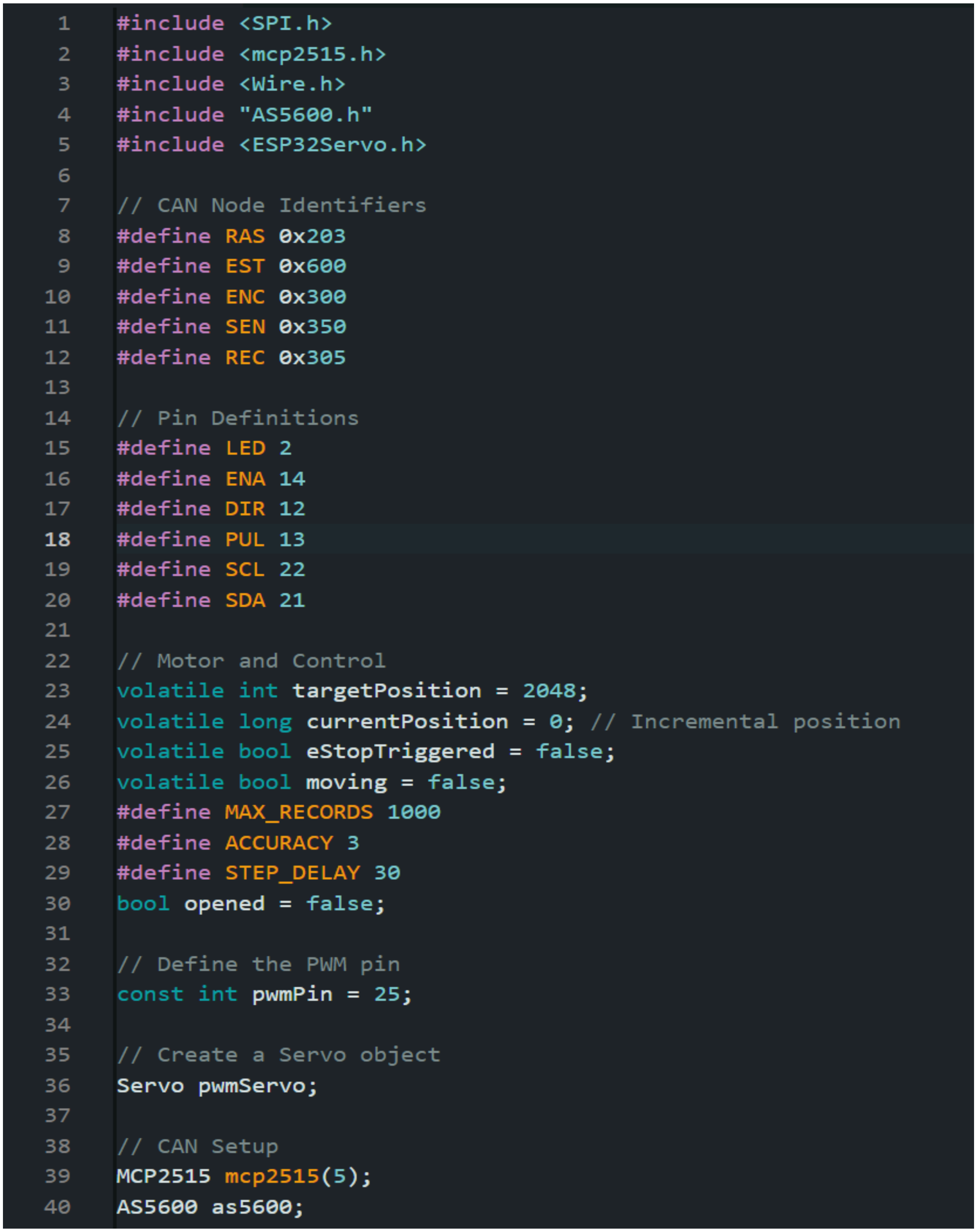
```
void sendEncoderData() {
  uint8_t encoderData[2];
  int rawAngle = as5600.readAngle(); // Get raw encoder value
  encoderData[0] = (rawAngle >> 8) & 0xFF; // High byte
  encoderData[1] = rawAngle & 0xFF;        // Low byte
  sendCAN(ENC, 2, encoderData);

  // Debugging output
  Serial.print("Raw Encoder Data Sent: ");
  Serial.println(rawAngle);
}

void sendSensorData() {
  int sensorValue = analogRead(26); // Replace with actual sensor pin
  uint8_t sensorData[1] = {sensorValue};
  sendCAN(SEN, 1, sensorData);

  // Debugging output
  Serial.print("Sensor Data Sent: ");
  Serial.println(sensorValue);
}
```

Figure C.2-6 Code used to send current sensor and encoder data for nodes 1 and 3

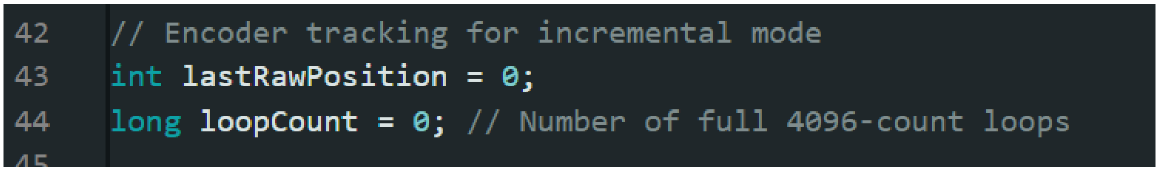


```
#include <SPI.h>
#include <mcp2515.h>
#include <Wire.h>
#include "AS5600.h"

// CAN Node Identifiers
#define RAS 0x201 // Raspi ID for this node
#define EST 0x600 // E-stop ID for all nodes
#define ENC 0x100 // Encoder ID for this node
#define SEN 0x150 // Sensor ID for this node
#define REC 0x305 //Record/replay

// Pin Definitions
#define LED 2
#define ENA 14
#define DIR 12
#define PUL 13
#define SCL 22
#define SDA 21

// Motor and Control
volatile int targetPosition = 0;
volatile int currentPosition = 0;
volatile bool eStopTriggered = false;
volatile bool moving = false;
#define MAX_RECORDS 1000  // Maximum number of records to store

// Constants
const float ACCURACY = 2; // Accuracy in steps
const int STEP_DELAY = 30;  // Speed of motor in microseconds per step

// CAN Setup
MCP2515 mcp2515(5); // CS pin for MCP2515
AS5600 as5600;      // AS5600 encoder instance
```

Figure C.2-7 Code used for node 1 initialization

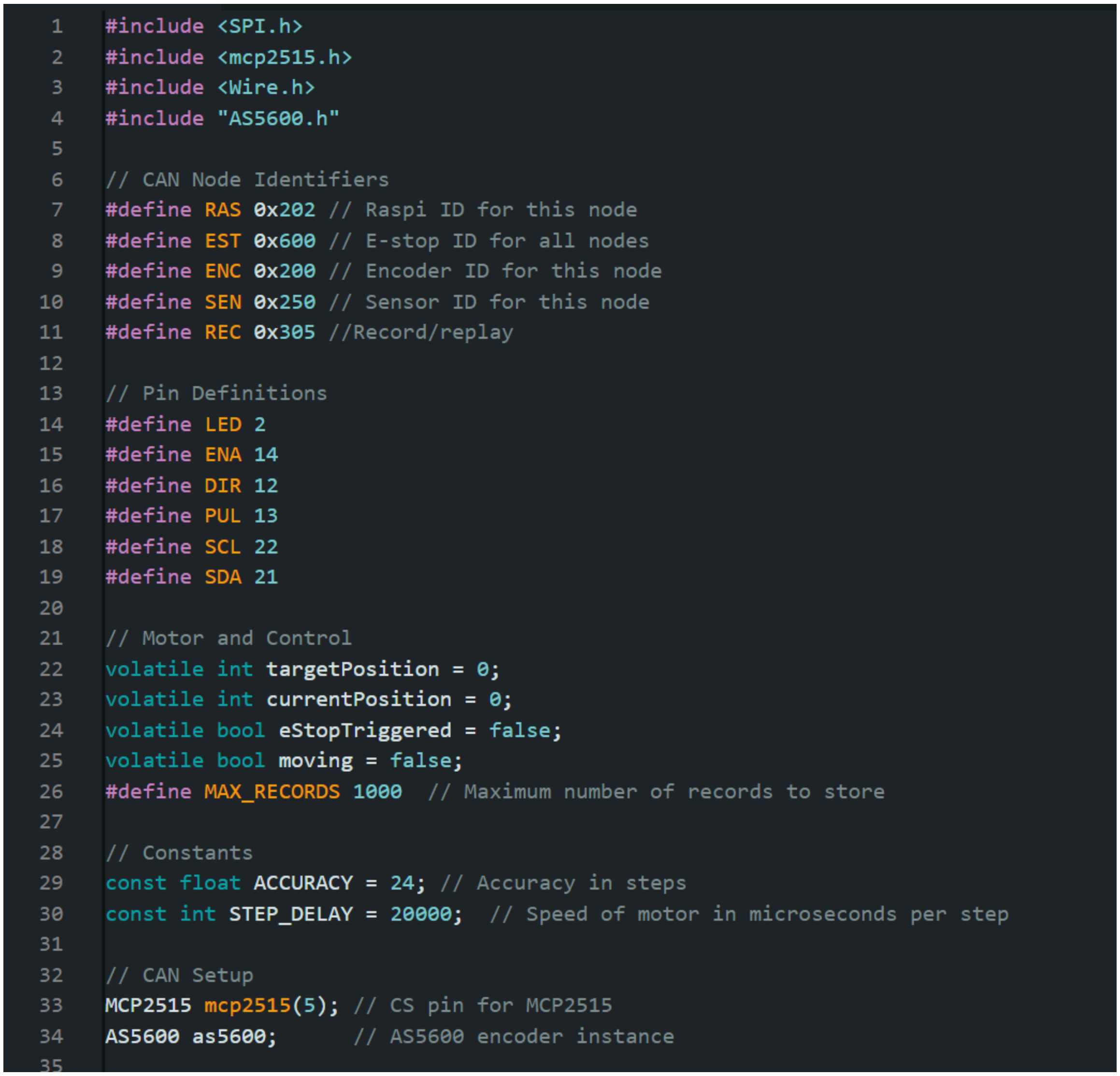


```
#include <SPI.h>
#include <mcp2515.h>
#include <Wire.h>
#include "AS5600.h"
#include <ESP32Servo.h>

// CAN Node Identifiers
#define RAS 0x203
#define EST 0x600
#define ENC 0x300
#define SEN 0x350
#define REC 0x305

// Pin Definitions
#define LED 2
#define ENA 14
#define DIR 12
#define PUL 13
#define SCL 22
#define SDA 21

// Motor and Control
volatile int targetPosition = 2048;
volatile long currentPosition = 0; // Incremental position
volatile bool eStopTriggered = false;
volatile bool moving = false;
#define MAX_RECORDS 1000
#define ACCURACY 3
#define STEP_DELAY 30
bool opened = false;

// Define the PWM pin
const int pwmPin = 25;

// Create a Servo object
Servo pwmServo;

// CAN Setup
MCP2515 mcp2515(5);
AS5600 as5600;
```



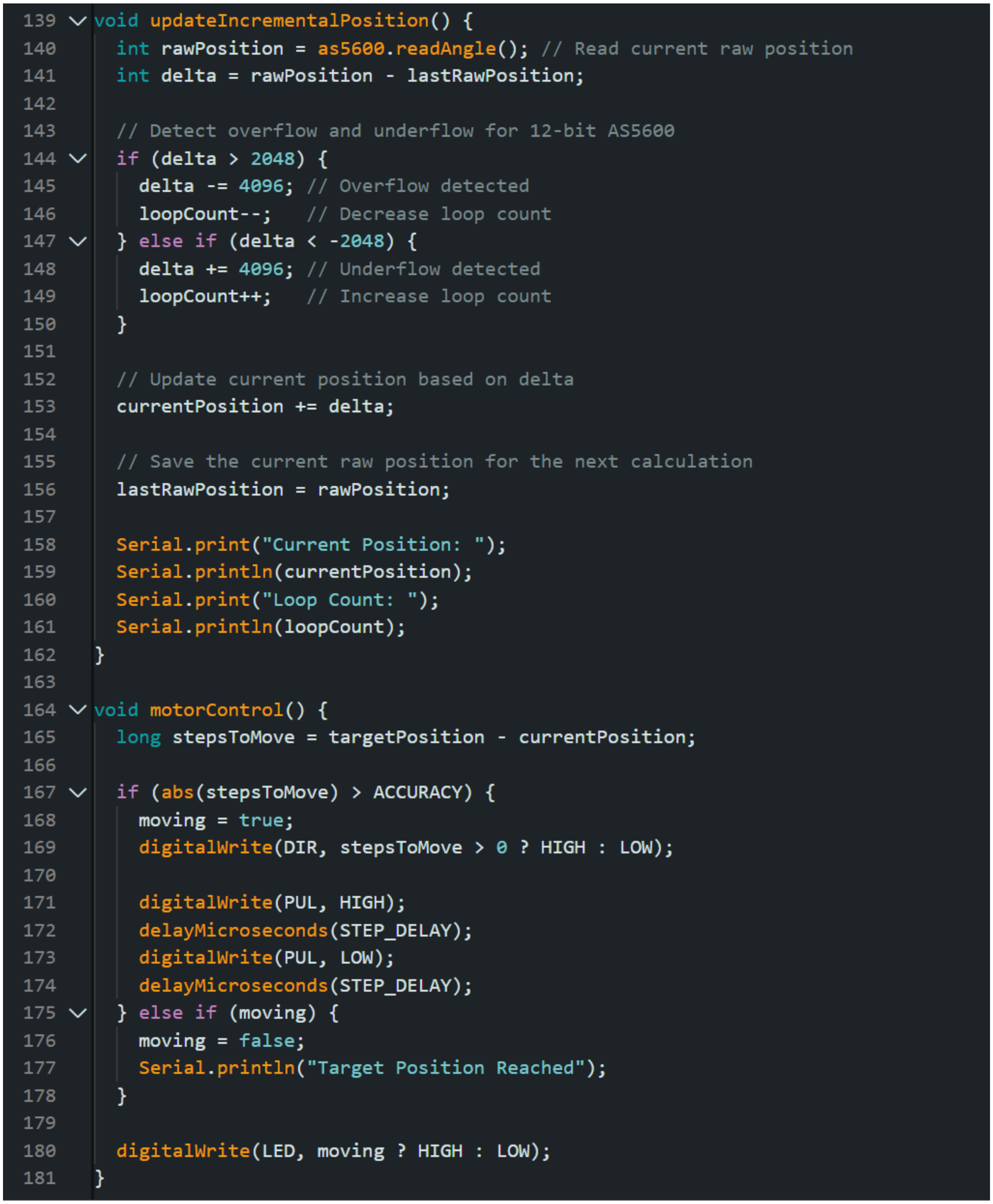
```
// Encoder tracking for incremental mode
int lastRawPosition = 0;
long loopCount = 0; // Number of full 4096-count loops
```

Figure C.2-8 Code used for node 2 initialization



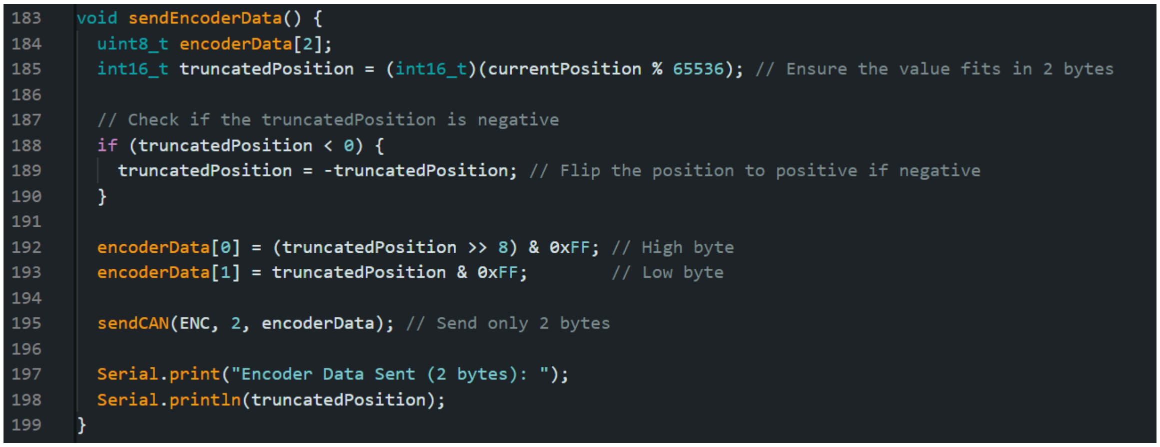
```
#include <SPI.h>
#include <mcp2515.h>
#include <Wire.h>
#include "AS5600.h"

// CAN Node Identifiers
#define RAS 0x202 // Raspi ID for this node
#define EST 0x600 // E-stop ID for all nodes
#define ENC 0x200 // Encoder ID for this node
#define SEN 0x250 // Sensor ID for this node
#define REC 0x305 //Record/replay

// Pin Definitions
#define LED 2
#define ENA 14
#define DIR 12
#define PUL 13
#define SCL 22
#define SDA 21

// Motor and Control
volatile int targetPosition = 0;
volatile int currentPosition = 0;
volatile bool eStopTriggered = false;
volatile bool moving = false;
#define MAX_RECORDS 1000  // Maximum number of records to store

// Constants
const float ACCURACY = 24; // Accuracy in steps
const int STEP_DELAY = 20000;  // Speed of motor in microseconds per step

// CAN Setup
MCP2515 mcp2515(5); // CS pin for MCP2515
AS5600 as5600;      // AS5600 encoder instance
```

Figure C.2-9 - Code used for node 3 initialization



```
void updateIncrementalPosition() {
  int rawPosition = as5600.readAngle(); // Read current raw position
  int delta = rawPosition - lastRawPosition;

  // Detect overflow and underflow for 12-bit AS5600
  if (delta > 2048) {
    delta -= 4096; // Overflow detected
    loopCount--;   // Decrease loop count
  } else if (delta < -2048) {
    delta += 4096; // Underflow detected
    loopCount++;   // Increase loop count
  }

  // Update current position based on delta
  currentPosition += delta;

  // Save the current raw position for the next calculation
  lastRawPosition = rawPosition;

  Serial.print("Current Position: ");
  Serial.println(currentPosition);
  Serial.print("Loop Count: ");
  Serial.println(loopCount);
}

void motorControl() {
  long stepsToMove = targetPosition - currentPosition;

  if (abs(stepsToMove) > ACCURACY) {
    moving = true;
    digitalWrite(DIR, stepsToMove > 0 ? HIGH : LOW);

    digitalWrite(PUL, HIGH);
    delayMicroseconds(STEP_DELAY);
    digitalWrite(PUL, LOW);
    delayMicroseconds(STEP_DELAY);
  } else if (moving) {
    moving = false;
    Serial.println("Target Position Reached");
  }

  digitalWrite(LED, moving ? HIGH : LOW);
}
```

Figure C.2-10 Code used for motor driving node 2



```
void sendEncoderData() {
  uint8_t encoderData[2];
  int16_t truncatedPosition = (int16_t)(currentPosition % 65536); // Ensure the value fits in 2 bytes

  // Check if the truncatedPosition is negative
  if (truncatedPosition < 0) {
    truncatedPosition = -truncatedPosition; // Flip the position to positive if negative
  }

  encoderData[0] = (truncatedPosition >> 8) & 0xFF; // High byte
  encoderData[1] = truncatedPosition & 0xFF;        // Low byte

  sendCAN(ENC, 2, encoderData); // Send only 2 bytes

  Serial.print("Encoder Data Sent (2 bytes): ");
  Serial.println(truncatedPosition);
}
```

Figure C.2-11 Encoder sending code for node 2

McLinden, Ian. *ESP32PWM*. GitHub, 2024, https://github.com/ianmclinden/ESP32PWM.

Tillaart, Rob. *AS5600*. GitHub, 2024, https://github.com/RobTillaart/AS5600.