Worcester Polytechnic Institute

MIRA

Modular Interchangeable Robotic Arm

Submitted on
April 25, 2018

Submitted by
Chris O'Shea, RBE
Alex Taglieri, RBE/CS
Ben Titus, RBE/ECE

Advised by
Susan Jarvis
Craig Putnam

A major qualifying project report submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science
This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review

Abstract

Low-cost robotic arms are becoming much more popular in educational settings. The goal of this project is to create a proof of concept for a modular robotic arm. To accomplish this, we have modified an existing arm to use our own modular control system, created a removable joint that can be connected to the end of the arm, and created an end-user interface which allows visualization of the arms movement in real time. Creating this arm will make robotics education accessible to a larger number of people, without compromising the potential for each person to gain a high quality understanding of the way robotic arms behave.

Acknowledgments

We would like to thank our project advisors Profs. Putnam and Jarvis for their guidance and support throughout this year. We would like to thank Kevin Harrington for each of his important contributions to our project such as hardware and software libraries as well as guidance with engineering decisions. Additionally, we would like to thank Profs. Bogdanov and Wyglinski for their help in debugging the CAN bus code. Finally, we would like to thank Kyle Sposato for his last minute help in implementing multithreading in our Java project.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

Currently, it is difficult for users to fully understand the motion of robotic arms and the kinematics that control them. This understanding is a crucial step in working with robotic arms safely and efficiently, but is often lacking due to the inability of diagrams and descriptions to fully convey what makes one arm operate differently than another. With the use of robotic arms becoming more common and the many different kinds of arms available, it is important to have a prototyping platform that can emulate many different kinds of arms so that the user can gain a better grasp of what components make up a robotic arm why one arm is better suited for a particular use case than another.

1.2 Goal Statement

The goal of this project is to create a proof of concept of an arm that can be reconfigured by end users such that they can create many different types of functionality from one set of principal components. To accomplish this, we need to break the arm down into modularized joints that can be rearranged to show how the combination of different kinds of joints can lead to a specific end result. We also need to create an adaptable electrical system that is able to modularly control each joint by having the capability to interface with multiple types of sensors and actuators. By doing so, we hope to take the first step into creating a standardized kit of parts and the software accompanying it to prototype almost any type of arm that is currently used.

1.3 Objectives

In order to measure our progress, we define a set of objectives that we need to accomplish in order to meet our goals outlined above. To create a proof of concept of a modular robotic arm prototyping platform, we set the following objectives:

- Research commonly used robotic arms and their uses.
- Understand the variety of different joints and what sensors and actuators are used to control them.
- Understand how the combinations of different joints affects the kinematics of the arm.
- Classify several different standardized joints and what sensors/actuators make them work.

- Create a control bus made up of connected joints with a node at each joint capable of controlling any single joint.
- Create a joint that can be added and removed from an existing arm in order to modify the arm's functionality.
- Create an arm control board which functions as an interface between a computer and our arm's control bus allowing for users to interact with the arm through a software interface.
- Create a software application which displays information about the arm in real time and allows users to easily set up their arm and send it commands.

1.4 Constraints

In order to complete this project in the alloted time, we had to place limits on our goals for the project. One such constraint was taking into account the time it would take to prototype, design and assemble a fully modular arm from scratch. Changes in the project's organizational structure led us to revise our goals (the original ones of which are reflected in Appendix A) Our solution was to create a smaller-scale example of modularity by modifying an existing arm by adding a joint that can be easily removed. By fully creating our own link that can be attached to the arm to increase utility, we proved that the idea of a modular joint is feasible and therefore those joints can be combined to create a modular arm. We also had to impose another constraint to ensure proper functionality of the arm which is that the arm can have at maximum six joints controlled at once. This constraint was decided based upon examples of other robotic arms as well as a way to make sure that the arm is structurally sound and within weight tolerances.

1.5 Summary

Once we outlined our goals and constraints for the project, we had a clear idea of where to start researching. Knowing what we had to accomplish as well as knowing the obstacles standing in our way, we were able to approach the project piece by piece, working towards our goals as well as keeping a solid perspective about the entire scope of the project. When we ran into design decisions that were not foreseeable before we got working, we referred back to our original goals and based our decisions off of these initial measures of project progress. Moving forward after defining the problem fully and how we wanted to accomplish it, we proceeded to conduct research on current robotic arm technology that we used as a reference for our arm.

2 Background

2.1 Introduction

In this section we begin with an overview of some existing robotic arms that are currently in industry use in order to gain a high level understanding of what different kinds of arms are out there. Researching the various use cases of existing robotic arms can help us define use cases for our arm. Next, we discuss modular arm technology that is in development in order to have a measure of our progress versus their projects. After examining these arms, we highlight how our project is different from the previously discussed examples and why this is important. Finally, we discuss some of the technology that had to be researched in order to inform our decision on how to design our arm and choose components.

2.2 Examples of Robot Arms

There are a few different types of robot arms available on the market today. Industrial robotic arms, several of the most common type of arms currently in use are defined as robotic systems used for manufacturing by means of an end effector. Since our arm is relatively small and is intended for handling lighter payloads compared to most industrial arms, we will begin with an overview of existing "desktop" industrial arms. Arms that fit this description have a reach of less than one meter. Such Industrial robot arms typically cost between $50,000 to $80,000 new and $25,000 to $40,000 used [2]. Some manufacturers of these industrial arms include ABB Robotics, Universal Robots, and KUKA Robotics. It's important to note that none of these arms are modular - in fact, they can't be changed at all!

2.2.1 ABB Robotics

ABB Robotics makes many small industrial arms. The ABB IRB 120 boasts a 580mm reach, 3kg payload, and 25kg weight. It has 6 degrees of freedom and can be mounted at any angle. The ABB IRB 1200 comes in two varieties, one with a 703mm reach and 7kg payload, and one with a 901mm reach and 5kg payload. Both of these arms have 6 degrees of freedom. The weights are similar at 52kg and 54kg respectively [2].

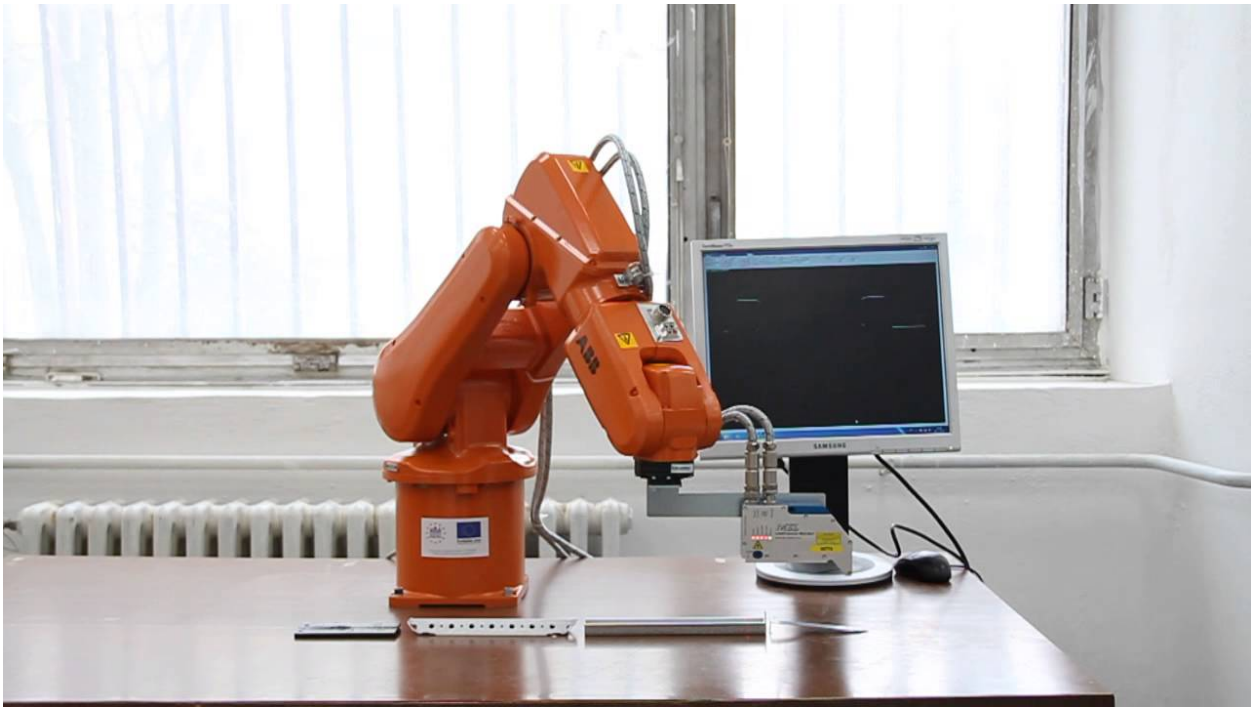

Figure 1: ABB IRB 120 arm [1]

2.2.2 Universal Robots

Universal Robots makes two robot arms in this size range. The UR3 is the smaller of the two with a reach of 500mm, payload of 3kg, and 11kg weight. A step up is the UR5 which has a 850mm reach, 5kg payload, and 18.1kg weight. Both of these arms have 6 degrees of freedom. Universal boasts that these arms are easy to implement and re-implement due to compact and lightweight construction, and simple programming interface [2].

2.2.3 KUKA AG

KUKA makes two robot arms in this size range. The KR3 R540 has a reach of 541mm, payload of 3kg, and weight of 26kg. It can be mounted on the floor, wall, or ceiling for added utility. The KR 5 sixx R650 is larger with a reach of 650mm, payload of 5kg, and weight of 127kg. It can only be mounted on the floor or ceiling. Both of these arms have 6 degrees of freedom [2].

2.2.4 Small Industrial Robotic Arm Comparison

Table 1 shows a comparison of all the robotic arms discussed in this section.

Table 1: Comparison of <1000mm reach industrial robot arms

Name	Reach (mm)	Payload (kg)	Weight (kg)	Axes
IRB120	580	3	25	6
IRB1200-7/0.7	703	7	52	6
IRB1200-5/0.9	901	5	54	6
UR3	500	3	11	6
UR5	850	5	18.1	6
KR3 R540	541	3	26	6
K5 sixx R650	650	5	127	6

2.3 Modular Arms

While there are many industrial arms in production, there are very few modular robotic arms. There is one commercially available robotic arm, the Robolink, made by igus. A few modular robot arms have been developed, including the reconfigurable modular manipulator (RMM), made by TRACLabs [3], and a single joint for the Modular Robotic Arm project MQP at WPI [4].

2.3.1 igus Robolink

Robolink is a modular robotic arm kit produced by the plastics manufacturing company igus. The kit contains parts to make an arm that is up to 6 Degrees of Freedom (DOF), with belt driven linkages powered by stepper motors that reside in the base of the robot. Robolink offers 7 individual links, ranging from 1-2 DOF and differing based upon their kind of motion (pivoting, rotating, swiveling). Each link is made of a lightweight and strong plastic or carbon fiber with cables inlaid in them, resulting in a low cost and weight arm. The cables used to control these links are made of a high strength synthetic fiber with has a tensile strength of 4,000N. Separating the actuation of each link from the joint allows the arms to be easily maneuverable with its lightweight and strong joints.

Purchasers of the kit are able to combine the links in different ways, allowing for a flexible, modular solution to robotic arms. Igus also offers their Robolink software for programming articulated arms that facilitates the programming of individual arms through the use of a simple, intuitive control software. The total cost of a kit to make a 6 DOF arm is $6000, and buying individual links will cost anywhere from $370 to $750 per link. While this price may be low cost compared to other arms such as the ABB robotic arm which can cost up to $200,000 in total, it is still not low enough for either hobbyists or people interested in learning about robotic arms who are prevented from doing so by the high entry cost. In addition to this, the belt system actuating each link requires the user to thread belts attached to the

actuators to each link in order to set up the robot. The long assembly time and intricacy also detracts from the idea of modularity because the time involved in switching configurations can inhibit users from really exploring the different workspaces and combinations this kit can create [5].

2.3.2 Reconfigurable Modular Manipulator

The reconfigurable modular manipulator developed by TRACLabs for NASA is a fully modular 7-DOF robot arm. Each joint and end effector have the same connector that provides power and control lines throughout the arm. Internal power and control circuitry take in these lines and convert them into movement. Joints can be swapped out by hand in a matter of seconds. Joints accept position or velocity data from the central communication lines and store physical characteristics about the joints in memory. This robot arm is not commercially available [3].

2.3.3 Modular Robotic Arm

This project aimed to close the market gap between inexpensive toy robot arms and expensive professional grade industrial arms. The group aimed to do this by designing a single joint that could be used to assemble a robot arm. Ultimately, a single DOF joint that was heavy, difficult to manufacture, and expensive to produce was designed and constructed. In their future recommendations section, the group stated that the goal of designing a modular robot arm was possible but their design was not the solution [4].

2.4 Our Robotic Arm System

Our modular robotic arm aims to offer a completely different use case compared to existing products. The system maintains a low cost while providing a versatile platform for anyone from novice engineers to rapid prototyping professionals. We accomplished this by avoiding expensive proprietary software and subtractive manufacturing; favoring off-the-shelf parts, 3D-printed structures, and freely available/open source software. Providing custom-built software for controlling the arm creates a plug-and-play environment suitable for most any skill level.

2.5 Technology

After examining all of these different robot arms accomplished their respective tasks, it was important to go a little more in-depth on how some very crucial components of any robotic arm are chosen and what that means for our project. Some of the most important ideas

of an arm are: where does the central processing happen? How can we tell what both the position of and the force on each joint are? It is questions like these that led us to research these key functionalities so that we could make the right choice when we design our arm.

2.6 Joint Control board

The control board is meant to be implemented as an independent module that interfaces with a main controller module. Its tasks are to send and receive data from the base controller and control the position of a single motor. As such, the factors that must be taken into account when designing the control board are methods of measuring joint position and motor torque, as well as ways to communicate with an off-board controller. Motor torque is proportional to motor current, and motor torque lets the controller know how hard the arm is pushing something. Therefore, the motor torque will be calculated from the measured current through the motor.

2.6.1 Joint Position Detection

Angular position sensing must be used to determine the joint angle of the motor. There are several commonly used methods of determining angular position, including potentiometers, optical encoders, and Hall effect sensors [6–8]. A comparison of the different angular sensors can be seen in Table 2.

Potentiometers are very commonly used to measure angular position due to their simple implementation and low cost. In addition to being low cost, potentiometers provide high linearity and accuracy [8]. Although generally robust, these sensors do not lend themselves well to many, rapid adjustments or mechanical vibrations. Both of these significantly reduce the lifespan of the sensor [6, 8]. The situations potentiometers excel in are those that require an easily adjustable voltage at low to medium adjustment frequencies, such as settings knobs on control panels or analog reference voltages as trim potentiometers [6].

Hall Effect sensors are less commonly used, and consist of a bipolar magnet rotating above a Hall effect sensor with the axis of rotation perpendicular to the plane of the sensor. Since there is no contact between the rotation and the sensor, these types of sensors have very long lifespans [6]. Unfortunately, these sensors do not provide high resolution since they are susceptible to electromagnetic interference and temperature, and also have some hysteresis [8].

Optical encoders are another method of measuring angular position. These sensors consist of a beam of light that shines on a slotted disk so that as the disk rotates, the slots break the light beam. These sensors can have very high resolutions and are resistant to shock and vibrations [7]. Like magnetic sensors, these sensors have very long lifespans since there is no mechanical connection on the sensor [6]. Unfortunately, these sensors are susceptible to foreign particles blocking the light beam from sensing the slots and causing incorrect readings.

The most common kind of optical encoder, the Quadrature encoder, does not sense absolute position; it can only read relative position, meaning that a quadrature encoder would need to be combined with some other sensor in order for the robot to be able to sense its joint angles correctly. Other encoders called Absolute Optical Encoders are capable of reading absolute position, but they are prohibitively expensive. [8].

It's worth noting that limit switches can be used to provide information about the location of a joint. Limit switches give a different voltage depending on whether they are pressed or not. When a limit switch is placed at the edge of a particular mechanism's travel range, it becomes possible to determine when the mechanism has reached one of its limits of travel.

Limit switches can be used in conjunction with quadrature encoders (which only provide relative, rather than absolute, position) to create a system which is capable of determining its absolute position. The system would need to go through a homing process at startup whereby the mechanism travels until the limit switch is pressed, at which point the encoders treat their current position as "home".

Table 2: Comparison of different angular position sensors

Sensor	Cost	Linearity	Accuracy	Lifespan	Notes
Potentiometer	$	Depends on ADC	Moderate	Short	Repeated motion at the same angle can lead to failure
Encoder	$$$	Very High	Very High	Long	Inexpensive encoders can't sense absolute position
Hall Effect Sensor	$$	High	High	Very Long	Requires special attention to surrounding magnetic fields when mounting

2.6.2 Current Sensing

Current sensing allows us to implement another kind of control that is common on robotic arms. It allows the robot to see the current at that the motor is drawing, allowing for force to be calculated using the constant resistance of the motor and Ohm's law. Current sensing

can be done in many ways. The most common way is by using a shunt resistor and an amplifier. A variant of this method is to use the resistance inherent in the wires or traces as a shunt resistor. Another common method of current sensing is to use a Hall effect sensor [9].

Shunt resistors are used in either high side or low side configuration. They are simple to integrate, low cost, and capable of measuring both AC and DC currents. The downsides to this method are relatively large insertion loss that increase exponentially with current, large thermal drift that must be compensated for, as well as large system noise from amplification. There are two main implementations of shunt resistors, high side and low side [9].

Low side current sensing means that the shunt resistor is placed in the return current path. This method is simpler to implement since the voltage on the shunt resistor is with respect to ground, so it can simply be amplified. Some problems exist with this, however, since the resistor separates the current path from ground. In this configuration, the circuitry used to measure the voltage on the shunt resistor will not report a fault if the system experiences a short circuit [9].

High side current sensing means that the shunt resistor is placed on the forward current path. This configuration is able to detect short circuit faults, an advantage to using this configuration over low side current sensing. An additional advantage is that the return current path is directly connected to ground. The downside to high side current sensing is that it requires a differential amplifier since the voltage across the shunt resistor is very close to supply voltage. [9].

Trace resistance sensing is very similar to using a shunt resistor, but there are some slight differences. Since there isn't a way to control the resistance of a copper trace, the system must be calibrated after being assembled. Another key difference is the amount of amplification needed. Copper traces have very low inherent resistance, so a very large amplification must be used. This large gain imposes a limitation on the maximum measurable bandwidth set by the gain bandwidth product of the amplifier [**?**].

Hall effect sensors are commonly used to measure current as well. These sensors can measure current intrusively or non-intrusively, as well as in open loop or closed loop configurations. Non-intrusive devices measure current by wrapping wire around a toroid that focuses the magnetic field on a sensor in a break in the ring of the toroid, or placing the Hall effect sensor on top of the current to be measured. These work fairly well, but are very susceptible to noise from magnetic fields upwards of 10cm away. Methods of shielding these sensors exist, but are complicated and expensive to implement. Intrusive sensors route current through the device and measure the generated magnetic field with a Hall effect device near the current path. Open loop applications take the voltage generated on the Hall effect sensor and condition it to whatever output is needed. Closed loop sensors reroute the sensed current to a secondary coil that is used to generate a proportional current to the measured current. This proportional current is then used as feedback to reduce error [9].

Insertion loss caused by these sensors is very small. Since these sensors measure current

by induction, they can only measure current in a specific frequency band, and high currents at high frequencies can cause these devices to overheat. Most of these frequencies are DC to some upper limit determined by the physical characteristics of the sensor, usually around 100kHz. These sensors cannot be used on their own, since they have an inherent voltage offset, called misalignment voltage, and suffer from high thermal drift. Integrated ICs that compensate for these factors are fairly widespread, allowing for very easy integration [9].

2.6.3 Off-Board Communication

There are many types of communication protocols that could be used to communicate with the main controller. Common protocols include SPI, I^2C, RS232, RS485, and CAN. Of these, SPI and I^2C are meant mostly for chip to chip communication while RS232, RS485, and CAN are all meant for module to module communication [10]. A comparison of these protocols can be seen in Table 3.

SPI is a full duplex, synchronous serial link consisting of 3 lines, SCLK, MOSI, MISO, and an additional line for every peripheral, CS. Data rates of up to 10MHz or more are possible due to the elimination of addressing with the CS lines and dedicated clock line [10]. Using SPI for controller-to-controller communication presents a problem, however. Since the data transfer rate is controller by the master, the slave could fall behind on processing data. This can be avoided by only transmitting data one direction at a time. Typically, SPI is limited to onboard communications since its signal degrades fairly quickly over distance [11].

I^2C is a half duplex, synchronous, multi-master bus consisting of a clock and data line. Data rates of up to 3.4MHz can be reached, and each device has a unique address or multiple addresses to avoid overlap. An interesting aspect of I^2C is clock stretching. Clock stretching is when a slave pulls the clock low to stall the master until it has enough time to process information. Typically, I^2C is limited to onboard communication since its signal degrades fairly quickly over distance [10].

RS232 is a common full duplex interface that consists of two transmitter/receiver pairs. The protocol limits communication to 1 sender and 1 receiver per line. Data rates of up to 115.2KHz are possible at a range of up to 200ft. Data is typically sent in 8N1 format with 8 data bits, no parity bit, and 1 stop bit or 7E1 format with 7 data bits, even parity bit, and 1 stop bit [10].

RS485 is a full duplex multi-master protocol that consists of up to 32 transceivers on the bus. Data transmission rates of up to 10Mbps and distances of up to 4000ft are possible. Transmission can be reduced to half duplex by removing one transceiver at each node. Data is sent much the same as in RS232 with either 8N1 or 7E1 being common formats [10].

CAN is a half duplex multi-master bus protocol that allows for many nodes to connect and send data on the two transmission lines. Messages are sent with unique addresses that

also act as arbitration for bus priority. Packets are fully defined with 11 or 29 bit addresses, 0-8 bytes of data, and some additional control and verification bits [12, 13]. Data rates of up to 1MHz and distances of up to 3000ft are possible. Multiple error checks are implemented at the hardware level since packets are predefined, allowing the controller to load a transmit buffer and let the transceiver send a message or wait until a receive buffer is full before reading the message [11].

HID (Human Interface Device) is a communications protocol [14] that defines two entities: the host and the device. It works by having devices define a data packet and a HID descriptor for the specific device. The host can then receive interrupts from the device during which the pre-defined data packet is transmitted from device to host.

Table 3: Comparison of off-board communication protocol performance

Protocol	Max Distance	Max Speed	Wires needed	Notes
SPI	Within circuit board	10MHz	SCLK, MOSI, MISO, + 1 CS for each node	No addresses needed
I^2C	Within circuit board	3.4MHz	2	Address included in message
RS232	200 feet	115.2KHz	4	Can include parity bit
RS485	4000 feet	10Mbps	4	Can transmit fast or far but not at same time
CAN	3000 feet	1MHz	2	Resilient signal

3 Methodology

Many engineering decisions were made during the course of this project. Decisions as large as how any joints the arm would support, and as small as which type would be fitting for a certain variable. These decisions were all made in order to uphold the design goals of our project.

3.1 System Design

The essential design of our project involved breaking a robot arm into modular joints and connecting each one to a CAN Bus. The CAN Bus is a very good communication protocol for implementation of multiple modular devices into one system. All of the joints would connect via the CAN Bus to one base module as picture in the figure below. This base acts as a translator, scheduler and power distribution system for the arm. It receives messages from the computer, translates them into CAN messages and sends them out at predefined times during the arm's configuration period and runtime. Finally, the base take in motor power, logic power and ground and connects each board so that it gets the power it needs. The computer is the main interface that our end user will interact with. It contains a GUI which takes input from the user, formulates packets and sends them to the base. The computer also has to read in data from the base to receive updates from the joints of the arm. In order to make sure that we were always dealing with the most up to date information, we decided that it would be best to store information about how each different joint is configured and its current status in a data container that gets accessed from both the communications side and the user interface. Because we have information about each joint stored on the computer, we are able to offload calculations from the joint board to the computer which would otherwise take up more processing power from the joint board's embedded microprocessor.

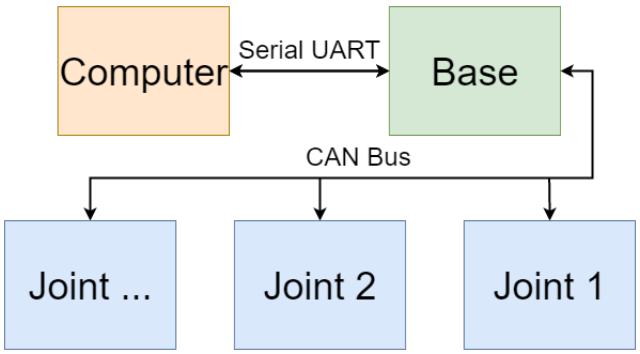

Figure 2: Block diagram of the entire modular arm system

3.2 Fourth Joint and Existing Arm

In order to show that our control system can be integrated with other arms, we designed a fourth joint to connect to an existing arm. By showing that we can add a newly designed joint to our existing arm, we are creating a proof-of-concept that shows the versatility of our control system. The design of the fourth joint was done in Solidworks, a software that allows users to design objects in a 3-D space [15]. Once we had designed the fourth link, we moved forward to the rapid-prototyping stage where we took our parts designed in Solidworks and converted them to 3-D printable models. We then used the software Cura [16] and the Lulzbot Taz 6 3-D printer [17] to fabricate a first iteration of our fourth joint. From here we continued to refine our parts and re-print pieces with tighter tolerances until they came together to make a fourth joint that interfaces with our control system. To find drawings of our Solidworks parts, please refer to Appendix H A.7

3.2.1 Fourth Joint Design

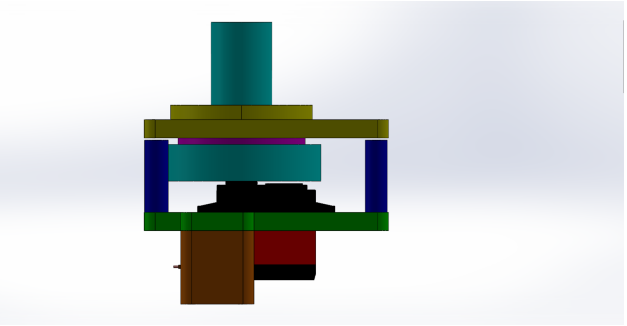

Figure 3: Joint 4 CAD Model

The fourth joint features a shaft which is supported by both a thrust bearing and a radial bearing. It can therefore tolerate loads that are parallel to the shaft, which the arm would encounter when picking objects straight up, as well as loads that are perpendicular to the shaft - forces that generate torques about the base of the motor.

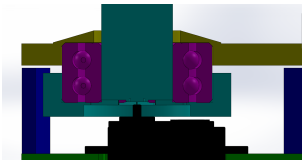

Figure 4: Cutout View of Bearing Mount

Our decision to create a joint that is actuated using a brushed DC motor was undertaken in order to prove that we can control DC motors in addition to the servos that already exist

on the arm. Next, we decided that since most of the joints currently on the arm provide rotation that results in the end effector moving vertically, we wanted a joint that provides parallel rotation.

To accomplish this we removed the control logic from a servo that already was used on the arm, converting it into a brushed DC motor. Next, we designed a direct-drive mount for the motor we fabricated so that the output shaft would rotate along the motor's axis of rotation. We built a mount for the motor and a structure to provide support for both radial and axial load on the shaft. This structure made use of multiple radial bearings, placed to keep the shaft in line and eliminate any friction between moving and static pieces of the joint. We also included a thrust bearing on the main shaft in order to stop thrust loads from being placed directly on the servo. Finally, we designed an idler-shaft that sits in two bearings so that it can freely rotate with negligible friction. The purpose of this idler shaft is to rotate in a 1:1 gear ratio with the main shaft using a timing belt to connect the two shafts. At the bottom of the idler shaft there is a magnet whose changing magnetic field is read by a Hall effect sensor mounted onto the bottom of the fourth joint so that it sits exactly 1mm from the magnet, allowing for optimal position reading.

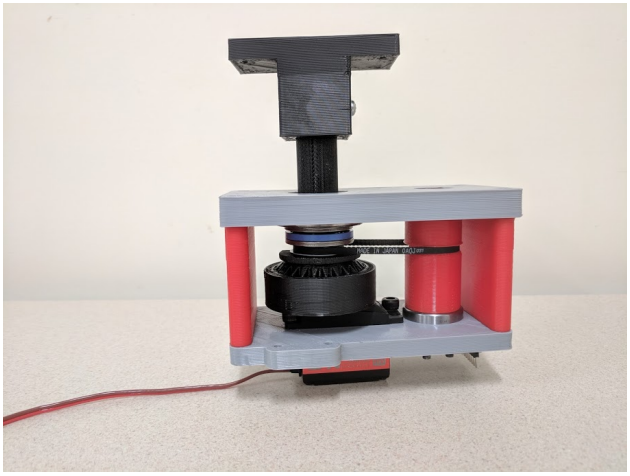

Figure 5: Side View of Final Removable Joint

3.2.2 Fourth Joint Prototyping

The process of prototyping our fourth joint started with creating an interface so that we can connect it to the already existing arm. We decided for the sake of simplicity and modularity to attach our fourth joint where the current end-of-arm-tooling would normally connect. The original end-of-arm mount can be seen in Figure 6. Keeping the mounting method standardized is important to maintaining modularity. A prototype of a joint that implements the standard can be found in Figure 7.

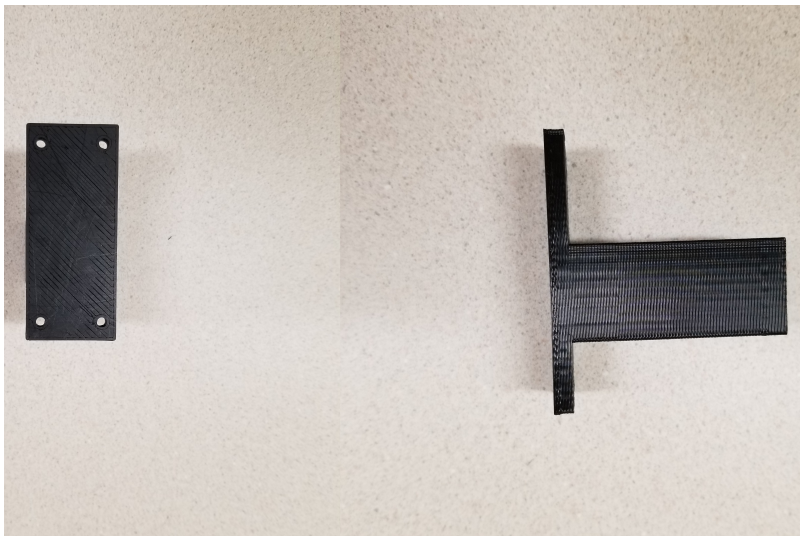

Figure 6: Current Arm End Effector Mount

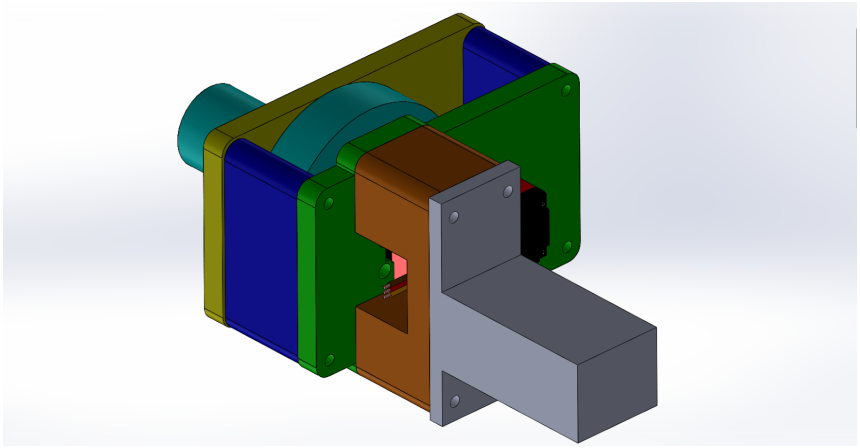

Figure 7: Joint 4 on End Effector

To create a prototype of our fourth joint, we determined the mechanical requirements of our arm and worked backwards to create a rapid prototyping model (RPM). RPMs are usually CAD models which are able to be turned into a functional model using a 3-D printer or other method. Once the functional model was 3-D printed, we would assemble the parts and test how they all fit together. With each new functional model we revised our RPM and printed a new functional model in order to meet the requirements for our fourth joint. This process of building, revising based on testing a physical prototype was only made possible due to the advances in recent years in 3-D printing technology making it feasible to create these prototypes so quickly while still having them be robust.

3.2.3 3-D Printing

3-D printing is convenient because it allows the user to manufacture parts in a novel way. Overall, we chose to 3-D print our fourth joint because it was what we were most familiar with and we had easy access to multiple 3-D printers. While the process of 3-D printing a part might not be as accurate as other more conventional methods of machining parts, it is a much easier method to learn and has a much quicker turnaround time. For our choice of 3-D printer, we used the LulzBot Taz 6 [17] with a single extruder head (Version 2.1) and 2.65mm filament. This printer is readily available to us through the Undergraduate Robotics Lab. The resolution of the printer was fine enough that it was easily able to achieve the tight tolerances needed to print parts such as the timing belt teeth.

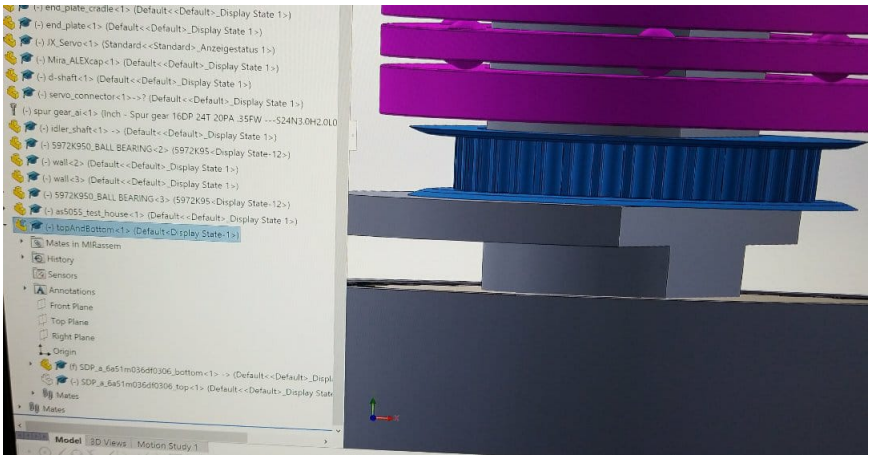

Figure 8: CAD Model of 3D-printable Pulley

3.2.4 Motor Selection

The performance of a DC motor can be characterized by measuring several key values: the stall torque, the stall current, the free running RPM, and the free running current. It's important that all the values be measured when the motor is given the same input voltage.

Specifications:

Input Voltage: 8.4V
Stall torque: 32.3 kg cm stall
Stall current: 5.25 A stall
Free running RPM: 0.1 seconds/60 degrees (100 rpm)
Free running current: 0.23 A

Table 4: HV5923MG Motor Curve Data from laboratory testing

Speed (RPM)	Torque (N-m)	Torque (in-lbf)	Current (A)	P_{out} (W)	Efficiency (%)	P_{in} (W)	Heat (W)	back-EMF (V)
0	3.16	27.98	5.25	0	0	44.1	44.1	0
7	2.94	26.02	4.9	2.15	5.24	41.15	38.99	0.56
13	2.75	24.34	4.6	3.74	9.69	38.62	34.87	1.04
20	2.53	22.38	4.25	5.3	14.85	35.67	30.37	1.61
27	2.31	20.42	3.89	6.52	19.94	32.71	26.19	2.17
33	2.12	18.74	3.59	7.32	24.24	30.18	22.87	2.65
40	1.9	16.79	3.24	7.94	29.17	27.23	19.29	3.21
47	1.68	14.83	2.89	8.24	33.96	24.28	16.04	3.78
53	1.49	13.15	2.59	8.24	37.9	21.75	13.51	4.26
60	1.26	11.19	2.24	7.94	42.25	18.8	10.86	4.82
67	1.04	9.23	1.89	7.32	46.18	15.85	8.53	5.38
73	0.85	7.55	1.59	6.52	48.99	13.32	6.79	5.86
80	0.63	5.6	1.23	5.3	51.09	10.37	5.07	6.43
87	0.41	3.64	0.88	3.74	50.49	7.41	3.67	6.99
93	0.22	1.96	0.58	2.15	44.12	4.88	2.73	7.47
100	0	0	0.23	0	0	1.93	1.93	8.03

Figure 9: HV5923MG Servo Motor

3.2.5 Arm Base

The mechanical side of the base module is relatively simple because the physical structure comes from the pre-existing arm. In order to make this compatible with our controls system, all we have to do is mount our joint and encoder boards to it. This is fairly simple since we use a slightly modified version of the encoder board on the existing arm with the same mounting scheme. Our joint board mounting scheme focused around mounting each board in a place where it would not be impeded by the movement of the arm during runtime.

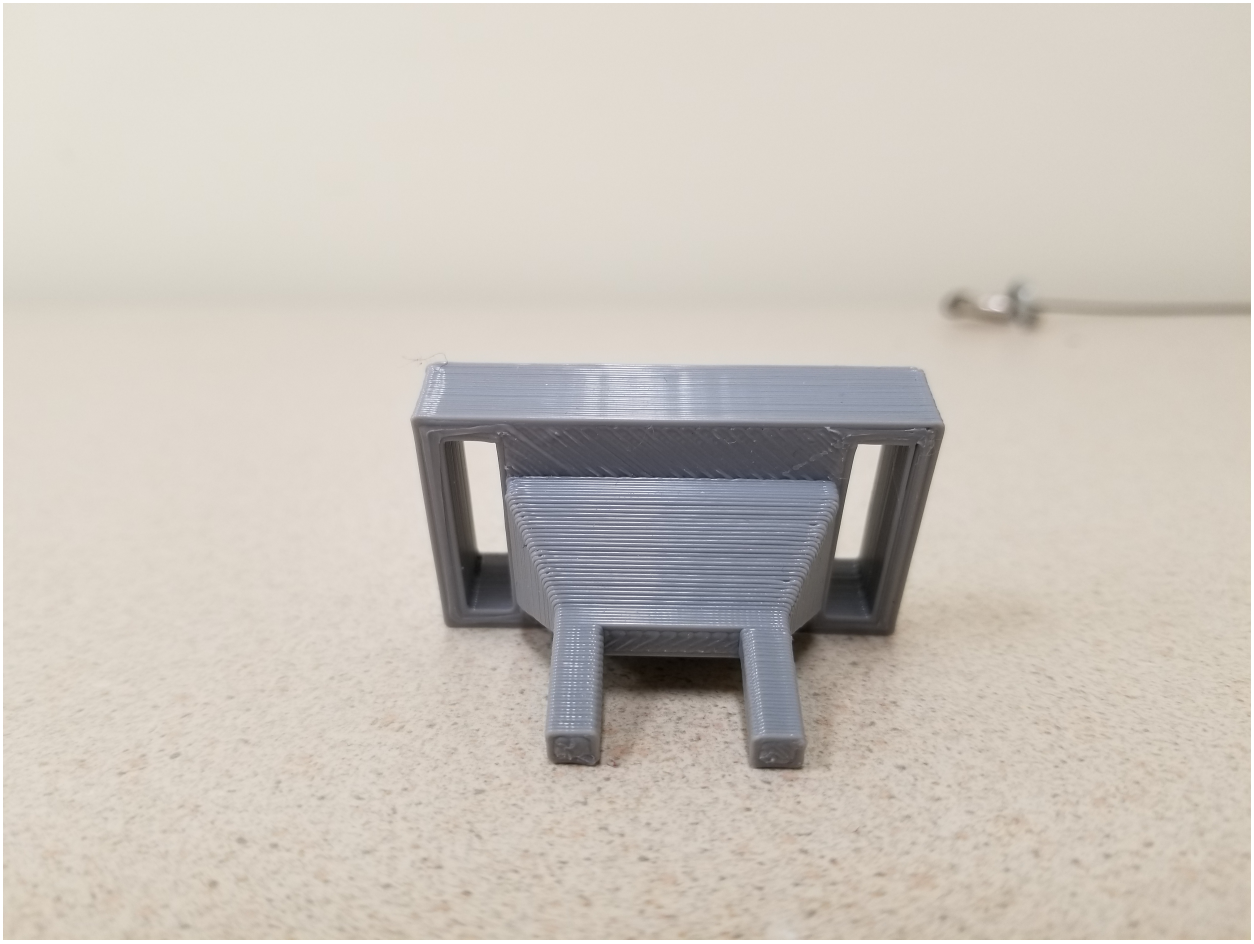

Figure 10: TM4C123G to arm mount

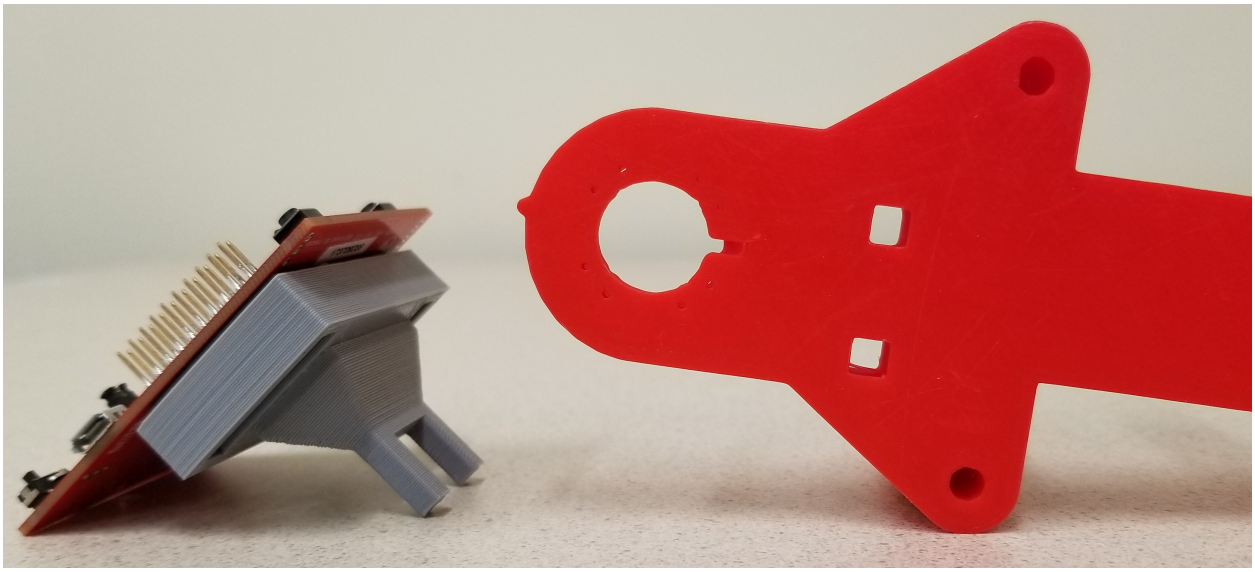

Figure 11: Mounting holes on arm and mounted TM4C123G

We have not only made sure that none of the mechanical components of the arm would come into contact with the board but also that none of the wiring between boards would be unplugged. This meant that we had to mount the boards far enough away from moving parts while still keeping it close enough to the sensors and motors that needed to interface with it. We decided to mount it in the same position for each category of joint type to keeps these needs consistent. Making use of an already existing area used to mount the cabling for the previous iteration of the arm seemed like the most efficient way to go about this. We designed a mechanical interface that was able to press-fit into our joint board and connect to the previous mounting solution.

3.3 Joint Control Board

The joint control board sits at the heart of the entire functionality of the project. Without a robust and capable design, any arm constructed would not function properly. Selecting the types of sensors to use for the control board was a very important step of the joint control board design. The role of the joint board is a very flexible one. It is able to handle multiple different kinds of joints and integrate with the sensors on each joint. Therefore, it has to accommodate multiple different combinations of sensors and use them to control the arm. By making the joint board so robust, it is able to accomplish many different functions, making it perfect for our modular control system.

3.3.1 Overview of Design

The joint board underwent significant revisions, outlined in Section 3.3.5, but the depth of revision significantly dropped after the decision was made to use the TM4C123GH6PM 32-bit ARM processor. This processor had many peripherals internally [18] that simplified the joint board design significantly. A block diagram of the final joint control board can be seen in Figure 12.

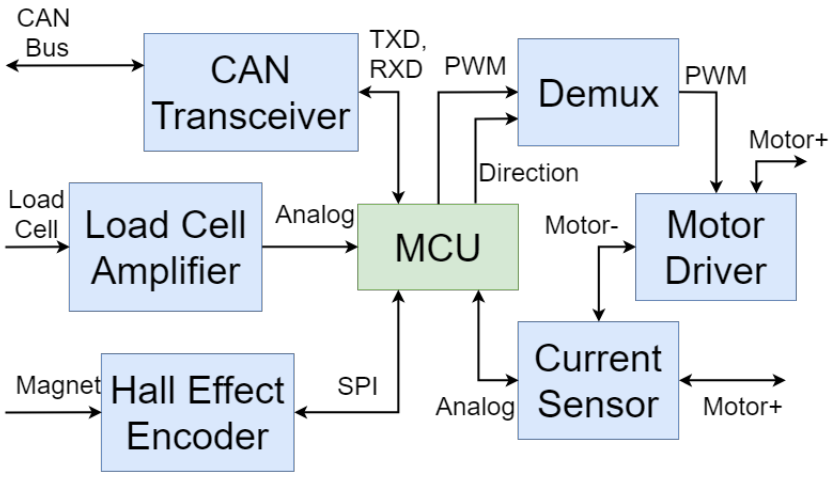

Figure 12: Block diagram of the joint control board

3.3.2 Joint Angle Sensor

As mentioned in Section 2.6, joint angle sensing is one of the main tasks needed in order to control a robot arm. Without this key sensor, the robot arm controller has no way of knowing where its links actually are in space. The type of sensor is very important since each has different design considerations that need to be taken into account.

Potentiometers seem like a good choice due to their simplicity and high accuracy capabilities. However, they do not lend themselves well to this application because of how quickly they wear out. Over time, as the joints move to different positions, the potentiometers will wear out quickly and cause inaccurate readings. Additionally, long lifespan and high resolution potentiometers can be very expensive. Furthermore, potentiometers are large and can be difficult to mount. Finally, the hard stop on the potentiometer means the joint angles will be limited to a certain range (typically about 270 °for single turn potentiometers).

The next obvious solution is to use optical encoders because they will not wear out and offer very high resolution capabilities. These sensors are not well suited for this application, however, since they are typically expensive, especially for high resolution encoders - and ones that are capable of reading absolute position. Additionally these sensors are somewhat bulky and would take up too much space in the closed environment of a joint.

This leaves us with Hall effect sensors. These sensors are very small and moderately high resolution while also being a contact-free sensor, so wearing them out will not be a concern. A main concern with Hall effect sensors is that the magnets need to be mounted to a non-metallic material in a precise manner. Traditional machining methods make this difficult to accomplish, but 3D printing allows us to easily overcome this challenge. Another concern is external electromagnetic interference, but with somewhat careful circuit board design, we should be able to minimize this issue.

3.3.3 Motor Current Sensor

As mentioned in Section 2.6, the motor current is another important factor in controlling a robot arm. Since motor current is proportional to motor torque, knowing the current lets the controller know how much force the arm is putting on a load, to give an example. There are a few different technologies that can sense the current through a motor, each with their own design considerations that must be taken into account.

A shunt resistor seems practical due to the simplicity of the design, but careful tuning must be done in order to get the noise levels down to a reasonable amount. In addition to this, the power loss when using a shunt resistor could cause the arm to stall before anticipated at the low voltages being used in this project. When the shunt resistor takes power from the motor, the whole motor curve slides inward, decreasing the maximum power output. Trace resistance would be a good alternative, but requires calibration after the circuit is constructed.

Instead of these, we decided to use a Hall effect current sensor. Hall effect current sensors are ready-made sensors that give low noise, properly calibrated outputs, are not very expensive, and are easy to integrate into a circuit design. These sensors have extremely small power losses to the motor. The main drawback of these sensors is that they have a low bandwidth, but we are using DC motors so this should not be a problem. Some care will need to be taken when placing these on the circuit, however, since they are sensitive to external magnetic fields. The hall effect current sensor chosen for this design was the ACS722 [19]. This device was selected by using Table 5.

Table 5: Current Sense IC Comparison

Part name	Supply Voltage (V)	Current Range (A)	Sensitivity (mV/A)	Output	Built in filter?
ACS722LLCTR-05AB	3.3	-5 to +5	264	Analog	y
ACS723LLCTR-05AB	5	-5 to +5	400	Analog	y
ACS724LLCTR-10AB	5	-10 to +10	200	Analog	n
ACS725LLCTR-10AU	3.3	0 to +10	264	Analog	n

3.3.4 Inter-board Communication

Inter-board communication is the heart of any arm. Without this, the joints of an arm would not know where how to move. A good, robust communication protocol that allows for simple transactions between boards is favorable for this application.

SPI and I^2C are mostly used for on-board, controller-to-peripheral communications and therefore are not a good choice for the base to control board communication. RS232 is not a good solution for this problem either because it is a single transmitter and single receiver per line, meaning a new wire needs to be run for each additional link. (Daisy-chaining the joints would work, but doing this would increase the computational overhead on the joint boards.) This leaves RS485 and CAN.

RS485 and CAN are similar in many ways, but with a few key differences that separate them. RS485 is very fast to transmit and simple to implement, but takes a lot of the controller's time to send packets. CAN has the advantage because the controller and transceiver control the transmission independent of the controller so the controller has more free time to process data. Another advantage CAN has over RS485 is the amount of error checking that goes on to ensure proper message transmission. For these reasons, we decided to use CAN to communicate between the base and control boards.

3.3.5 Preliminary Joint Control Board Designs

Preliminary joint control board design was done very early on in the project. Unfortunately, the majority of this work was done before much of the necessary research for part selection, leading to many revisions and re-designs, though the main concepts stayed in place. This section outlines the preliminary designs and revisions of those up until the selection of the TM4C123GH6PM microcontroller, at which point the design solidified and forward progress began.

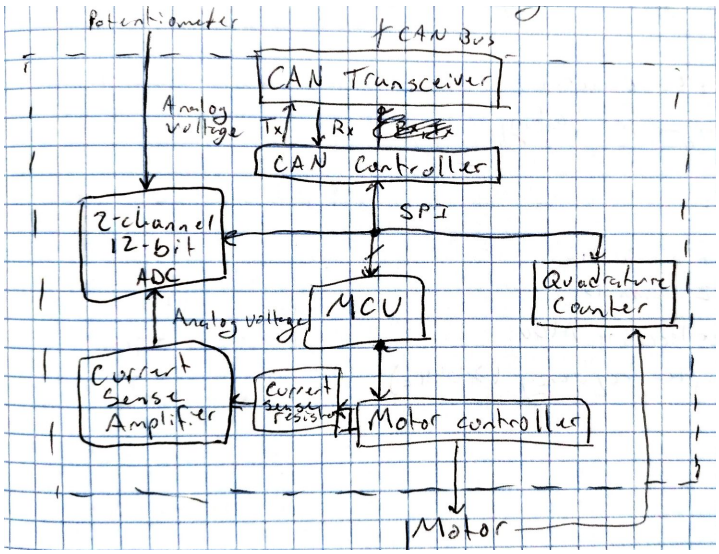

Figure 13: First draft of the complete joint board block diagram

The first complete block diagram of the joint control board Figure 13. This diagram was centered around a MSP430G series microcontroller, and focused on keeping the cost of the board down. Some component selection was solidified , including the following: MSP430G2553 (microcontroller), MCP3202 (ADC), DRV8872 (motor driver), LS7366R (quadrature counter), MCP2515 (CAN controller), and MCP2561 (CAN transceiver).

The decision to replace the potentiometer with an absolute hall effect encoder was made to lengthen the lifespan of the device. Greater detail about this decision is presented in Section 3.3.2. This change resulted in the diagram shown in Figure 14. Certain components, such as the CAN controller and transceiver, operated at 5V while others, such as the microcontroller and absolute hall effect encoder, required 3.3V. Most other components could function at either voltage, and were placed on either side of the level shifter for other reasons. The ADC, for example had a supply-voltage-dependent maximum SCLK frequency.

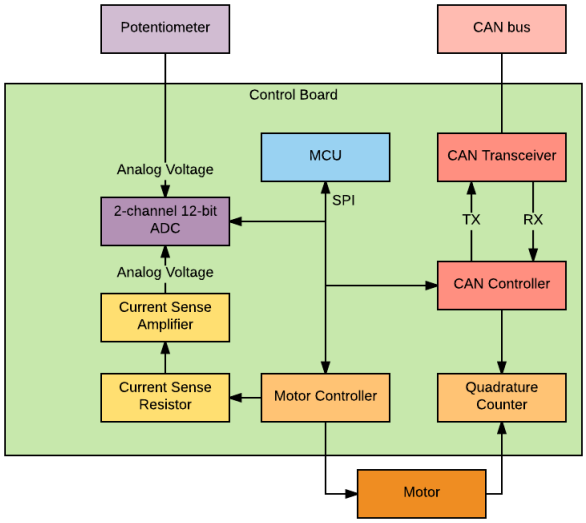

Figure 14: Iteration of preliminary joint board block diagram

Some planning was done to coordinate communication between the components, including placement of a 3.3V to 5V level shifter. This planning led to the block diagram shown in Figure 15. This diagram requires many lines to pass through the level shifter, which is not ideal and led to the decision to revise component selection.

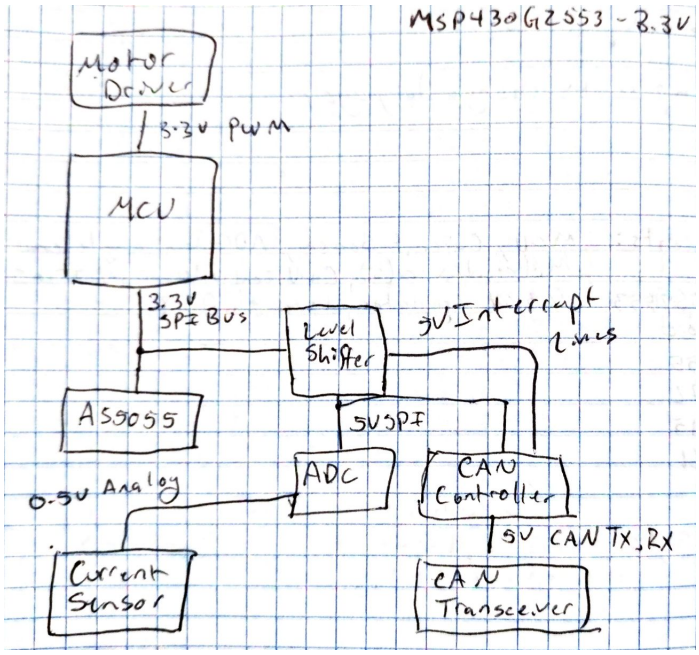

Figure 15: Mostly complete preliminary joint board block diagram

3.3.6 ADC Selection

The first design of the joint control board used a potentiometer to detect the joint angle and a current sense amplifier to detect the motor current. Both of these signals require an ADC to process, and the original microcontroller did not have a high resolution ADC on board, so an off board ADC was needed. Table 6 highlights the different ADCs that were evaluated before selecting one. The MCP3202, while not necessarily the best option from the chart, was selected since it has a very easy to use interface and was available in DIP package for easy testing. With the switch from the MSP430G2553 to the TM4C123GH6PM, the external ADC was no longer needed.

Table 6: ADC Selection

Part name	Supply Voltage (V)	Sampling rate (MHz)	SCLK Freq (MHz)	Price ($)
ADS7042	3.3	1	16	2.65
ADS7043	3.3	1	16	2.65
ADS7044	3.3	1	16	2.65
ADC121S021	3.3, 5	0.2	4	2.85
ADC121S051	3.3, 5	0.5	4	3.31
ADC121S101	3.3, 5	1	4	3.39
ADC7476	3.3, 5	1	20	3.41
MCP3202	3.3, 5	0.1	1.6	2.31
MAX11665	3.3	0.5	8	2.73

3.3.7 Preliminary Price Breakdown

Initially, one of the goals of this project was to produce a very low cost joint board. An estimate that matches the joint board block diagram from Figure 14 can be seen in Table 7.

Table 7: Summary of Parts Selection

Function	Part name	Price (qty: 1)	Price (qty: 1k)
CAN Controller	MCP2515	$1.87	$1.42
CAN Transceiver	MCP2561	$0.90	$0.68
ADC	MCP3202	$2.61	$1.98
MCU	MSP430G2553	$2.41	$1.18
Motor Controller	DRV8872	$2.17	$1.06
3.3V Voltage Regulator	LP2950-33	$0.47	$0.13
Current Sensor	ACS723LLC10AB	$5.53	$2.36
Encoder	AS5055a	$8.07	$5.48
Price (qty: 1)	$24.03		
Price (qty: 1k)	$14.30		

3.3.8 Motor Driver

The DC motor driver selected was the DRV8872. This driver takes in two inputs [20], in1 and in2, which affect the output much like inputs to an H-bridge.The exception is when both inputs are high. In this case, the inputs are pulled together as a motor brake. A truth table can be seen in Table 8.

Table 8: DRV8872 truth table

IN1	IN2	OUT1	OUT2	Description
0	0	Z	Z	Coast
0	1	L	H	Reverse
1	0	H	L	Forward
1	1	L	L	Brake

This motor driver was chosen because is provided high enough current limit for the selected motors to operate normally and low on resistance to provide more power to the motors and not generate excess heat. Another benefit of the DRV8872 is that it has a simple interface of 2 PWM inputs which is relatively easy to interface with from our microcontroller.

Table 9: Motor Driver Comparison

Part name	Peak Current (A)	Continuous current (A)	Control method
LMD18245	6	3	direction, brake
DRV8842	6	3.5	PWM
DRV8829	5	3.5	Phase, enable
L298	3	2	in1, in2, en
DRV8872	3.6	3.5	in1, in2

To measure the speed of the motor, a servo horn with 6 spokes was attached and a beam break sensor was mounted on the motor with the spoke traveling through the beam. The signal line of the beam break sensor was connected to an Arduino that measured the frequency by incrementing a count in an interrupt triggered on a pin change. The ISR just incremented a count that was printed out and reset every 5 seconds. Some issues arose with this system, however, since there was a small amount of bouncing on the rising and falling edges, leading to multiple readings for each beam break. This was solved by placing a 10nF capacitor from the signal line to ground. Since the line went high 12 times per revolution and the value was printed out every 5 seconds, the actual printed value happened to be in revolutions per minute, as shown in 1.

$$rpm = \frac{1rev}{12ticks} * \frac{1}{5} * \frac{60seconds}{1minute} \tag{1}$$

During initial testing, the motor driver was wired up with V_m of 8.4V, logic voltage of 5V, a 10kΩ pull up resistor on nFault, Isen grounded, and the motor outputs connected to a DC motor (See datasheet [20]). During this test, one of the inputs was connected to an Arduino Uno, outputting a constant PWM wave using the `analogWrite()` function with a duty cycle of approximately 50% at 490Hz. The motor turned, but very slowly and with a high pitched whine. When a 100μF capacitor was placed from V_m to ground, the motor spun up to full speed and the whining sound went away.

Further testing revealed a strange behavior when increasing the frequency of the input PWM signal. The motor spun normally at low frequencies of around 500Hz, but at around 1kHz the motor started slowing down and making a whining sound. The problem worsened with increasing frequency. Eventually, this problem was fixed by using a power supply that could output 3A and adding capacitors from in1 and in2 to ground. With these additions, the motor driver functioned as expected.

3.3.9 Demultiplexer

A demultiplexer was not necessary for the operation of the joint board. However, it simplifies the motor control signal from two PWM lines to one PWM line and a digital output, speed and direction. Since it was fairly simple to implement, the additional complexity in the circuit was added and the software was simplified by only needing to set up a single channel on the PWM controller.

The original demultiplexer selected for the joint board (SN74LVC1G19) did not output the correct values to drive the motor driver (DRV8872). The demultiplexer output can be seen in Table 10 and the motor driver inputs can be seen in Table 8. When the EN pin was pulled high, both outputs would also be driven high. This effect is undesirable since, when given a PWM signal, this would cause the motor driver to turn then brake then turn again as opposed to the desired turn then coast then turn. To solve this problem, a different chip (SN74LVC1G18 [21]) was selected. The truth table for this chip can be seen in Table 10.

Table 10: SN74LVC1G18 truth table

Inputs		Outputs	
S	A	Y0	Y1
0	0	L	Z
0	1	H	Z
1	0	Z	L
1	1	Z	H

3.3.10 INA332

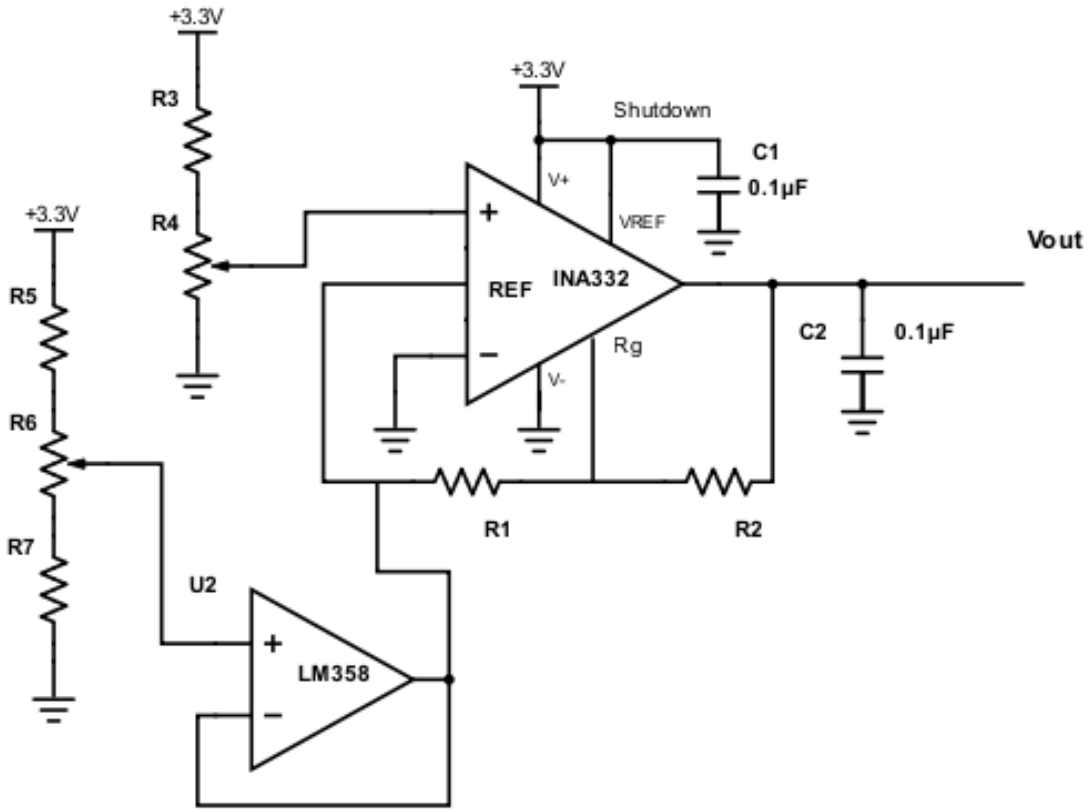

Figure 16: INA332 Test Circuit

The INA332 instrumentation amplifier is used to measure the force applied to a load cell. The amplification of this amplifier is given from the datasheet [22] as Equation 2. Since the load cell needed an amplification of at least 100 to amplify the .03V signal to 3V, the calculated resistances were $R_1 = 10\text{k}\Omega$ and $R_2 = 195\text{k}\Omega$. The actual values selected for R_1 and R_2 were 10kΩ and 200kΩ, respectively. This gives the amplifier an expected gain of 105 V/V. See Figure 16 for the circuit diagram.

$$G = 5 + 5(\frac{R_1}{R_2}) \tag{2}$$

The INA332 needs a voltage reference to use as the 0V differential output. Initially, two 1kΩ resistors were used to apply this voltage. This caused the output to change nonlinearly with the input voltage. The resistors were replaced with a LM358 dual operational amplifier, configured as a voltage buffer with the input connected to a potentiometer. The INA332

inputs were connected to ground and the LM358 buffer potentiometer was adjusted to set the 0V output to $\frac{1}{2}V_{cc}$. This voltage was 1.846V. Instead of a potentiometer, two resistors were used to create this 1.846V offset.

3.3.11 Hall Effect Encoder

An Arduino Uno was used to ensure that the AS5055a absolute Hall effect encoder was functioning as we intended. Arduino makes rapid prototyping very easy by providing many libraries and a simple interface to quickly test a device without worrying about the board not functioning. Using the Arduino SPI library, the correct packets to send to the AS5055a were verified, along with the correct speed of both the SCLK and CS lines. The AS5055a datasheet specifies [23] that the chip needs to receive a joint angle request at least every 0.6ms in order for the device to not go into low power mode. With this in mind, a sampling speed of 1kHz was selected.

In addition to the Arduino testing, tests were run with the TM4C123GH6PM and a test rig. The encoder test rig underwent significant revision to improve its reliability and accuracy in position the magnet above the hall effect sensor. The first iteration can be seen in Figure 17 and the second iteration can be seen in Figures 19 and 18.

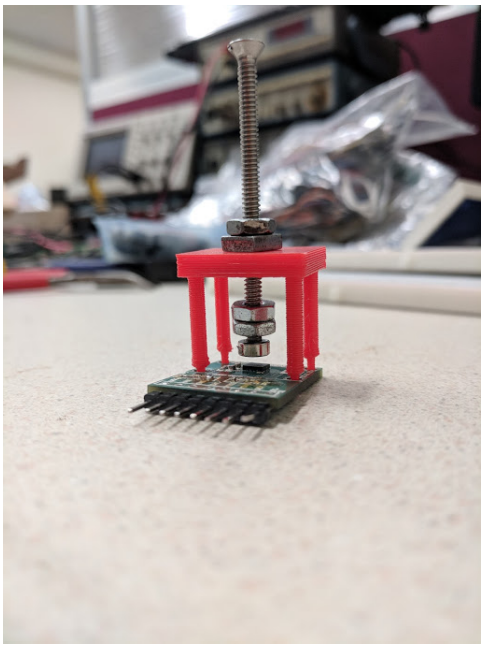

Figure 17: First Iteration of Encoder Test Rig

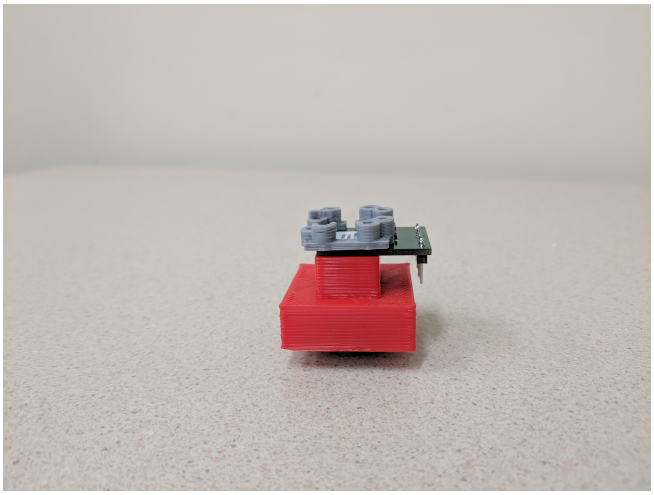

Figure 18: Second Iteration of Encoder Test Rig, Side View

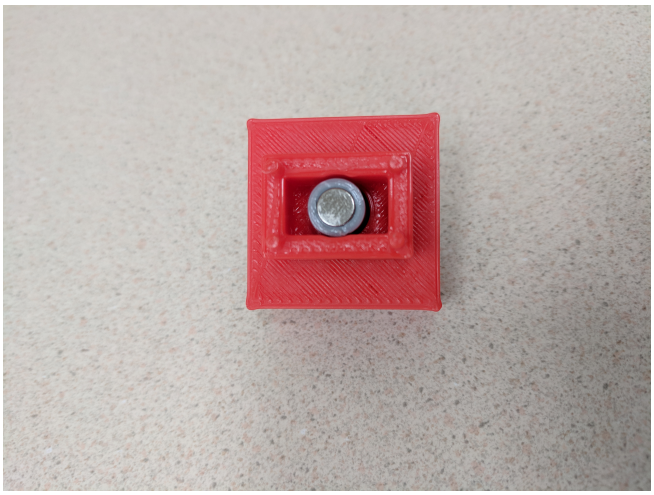

Figure 19: Second Iteration of Encoder Test Rig, Magnet View

One major problem encountered with the hall effect encoder was very noisy signals. Every so often, the encoder would read a value that was around 100 counts off of all the other values read in. This is most likely either an issue with magnet alignment or a noisy SPI line. In either case, this was largely solved in code with an 8-item circular buffer and exponential filter with alpha of 0.85. Using these two in conjunction with one another, these sporadic errors were largely evened out. The value 0.85 was decided on by testing the various values.

An alpha of 0.8 would very occasionally have erroneous readings and an alpha of 0.9 was very steady, but had a slightly delayed reaction time compared to an alpha of 0.8. An alpha of 0.85 was a compromise between steadiness and reaction time. All of the data was plotted and can be seen in Figures 20, 21, 22, 23, and 24.

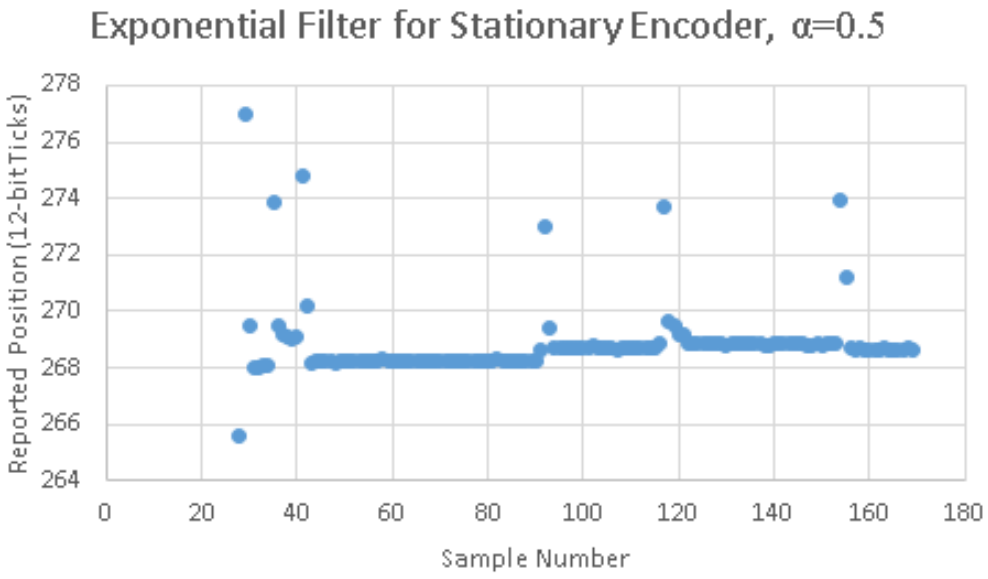

Figure 20: Behavior of the encoder value with alpha of 0.5

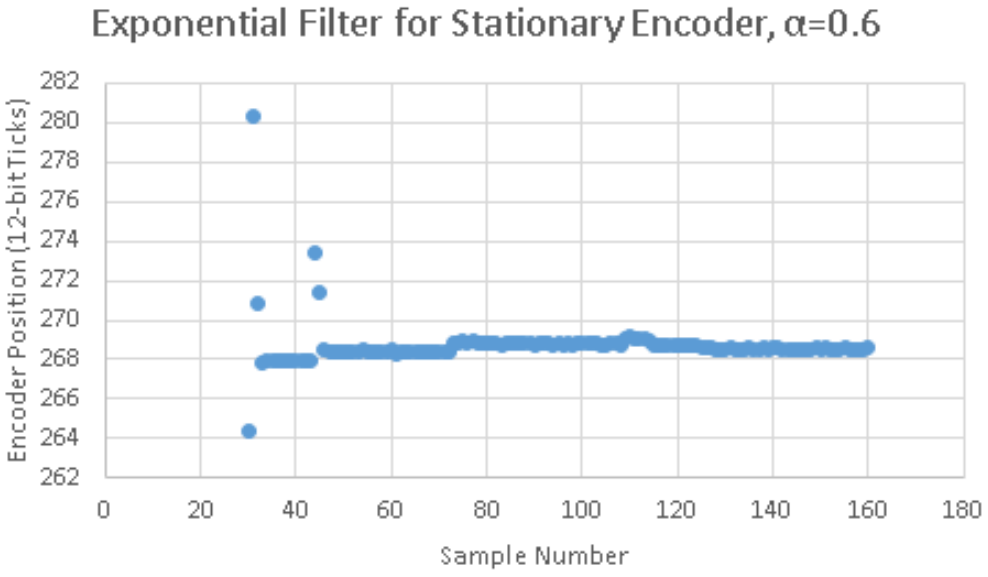

Figure 21: Behavior of the encoder value with alpha of 0.6

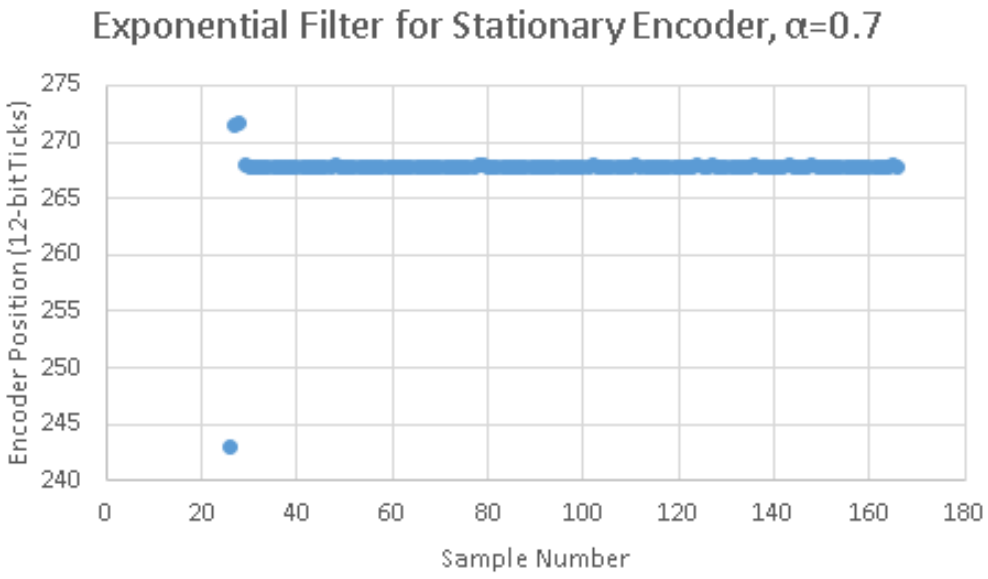

Figure 22: Behavior of the encoder value with alpha of 0.7

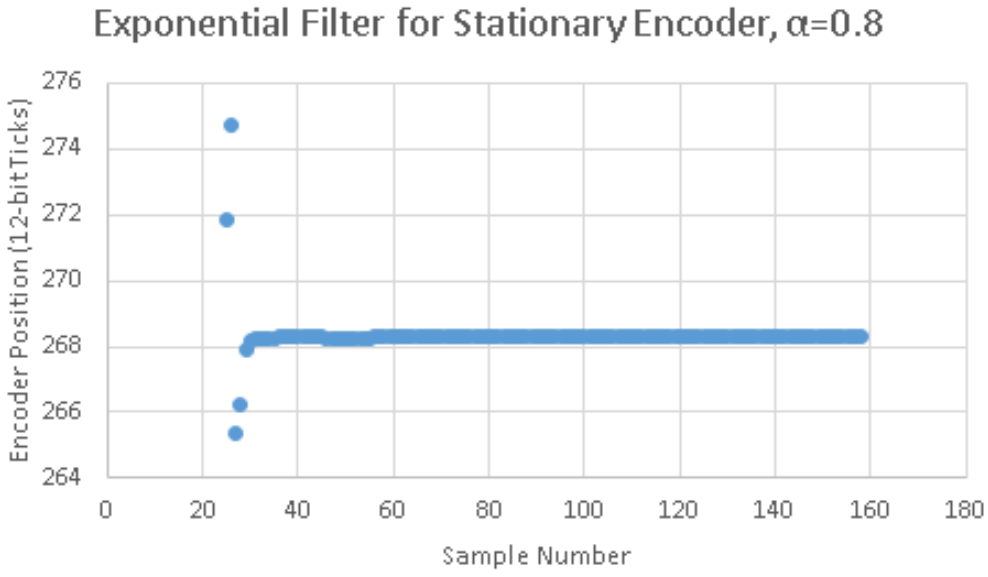

Figure 23: Behavior of the encoder value with alpha of 0.8

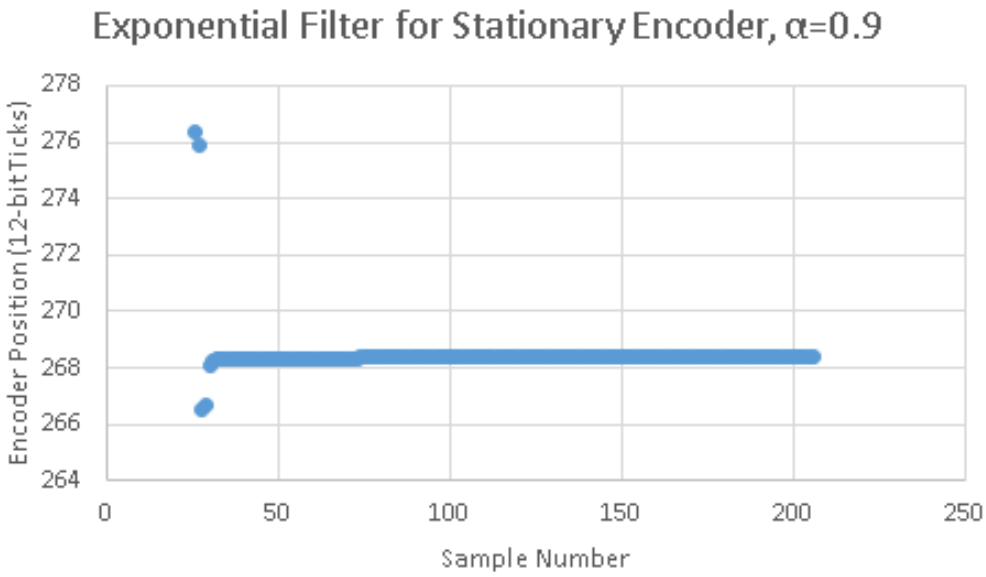

Figure 24: Behavior of the encoder value with alpha of 0.9

3.3.12 TM4C123GXL Launchpad

Selecting a microcontroller was a key part of making the joint control board. Without a capable MCU, the joint board would not be able to function as we want it to, but buying the best microcontroller on the market can be costly. The EK-TM4C123GXL [24] is an ARM Coretex M4f-based microcontroller evaluation kit from TI that has many peripherals to allow us to control the joint board without buying many external peripherals. The peripherals [25] on this chip that we will need include a CAN controller, 12-bit ADC, USB controller [26], SSI controller, PWM controller, and many GPIO. These peripherals were implemented separately with a test circuit configured to verify that each peripheral was functioning correctly.

Several peripherals were needed to achieve the desired functionality from our microcontroller. The test board consisted of a potentiometer connected to an ADC pin, an SPI controlled ADC (MCP3202), the 1:2 demultiplexer (SN74LVC1G18), CAN transceiver (TC332), and some LEDs.

As a temporary stand in for the AS5055 absolute Hall effect encoder to test the SSI peripheral (used to verify the functionality of the rest of the system), a MCP3202 12-bit, 2 channel ADC was used. Both devices use SPI to communicate their sensor data back to the MCU, and the packets are similar in structure. Some differences between the two that can be changed are a maximum sample rate for the AS5055 of 1ms as opposed to the few SCLK cycle delays for the MCP3202. The AS5055 has a maximum SCLK frequency of up to 10MHz at 3.3V [27] while the MCP3202 has a limit of 900kHz at 3.3V.

The potentiometer was connected to PE0 which was enabled at AIN3. The ADC was set to

sample at 1kHz with hardware oversampling 16x enabled. A timer was configured to start an ADC conversion every millisecond, and a GPIO pin was set to 0 every time the ADC finished a conversion and set to 1 when the ADC conversion began. The time necessary to sample once at 16x hardware oversampling was around 5μs.

3.3.13 Printed Circuit Boards

A PCB was made [28] for each of the major subsystems as a more final test. The subsystems for which a PCB was designed include the motor driver (both with and without a current sensor), load cell amplifier, CAN transceiver, and TM4C123GH6PM microcontroller. After these were verified to be working, a Boosterpack compatible design of the whole joint control board.

The motor driver PCB can be seen in Appendix A.7 and has the demultiplexer and a shunt resistor for triggering the automatic internal shutoff if the current gets too high. The motor driver with current sensor has a current sensor integrated in the board and can be seen in Appendix A.7. The load cell amplifier has the LM358 op amp as well as the INA332 instrumentation amplifier and can be seen in Appendix A.7. The CAN transceiver PCB was designed that contains the 6 necessary DIP switches, CAN transceiver, and auto-terminate circuit and can be seen in Appendix A.7. The joint control board Boosterpack has all of these components in a single PCB that mounts on top of the TM4C123GXL Launchpad and can be seen in Appendix A.7. All of these PCBs can be seen in Figure 25. A closeup of the joint control board Boosterpack attached to a Launchpad can be seen in Figure 26.

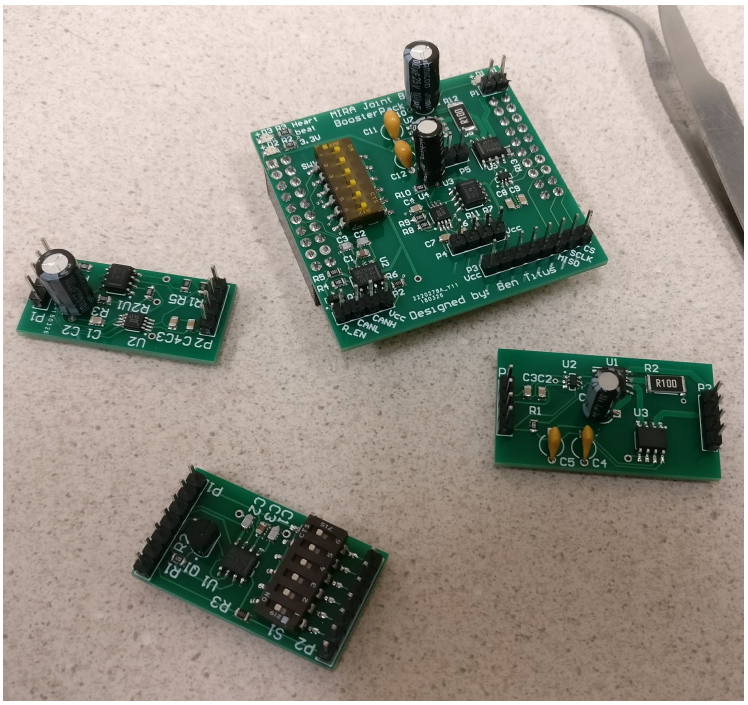

Figure 25: Assembled PCBs of each major subsystem including motor driver with current sensor (right), CAN transceiver (bottom), load cell amplifier (left), and joint control board Boosterpack (top)

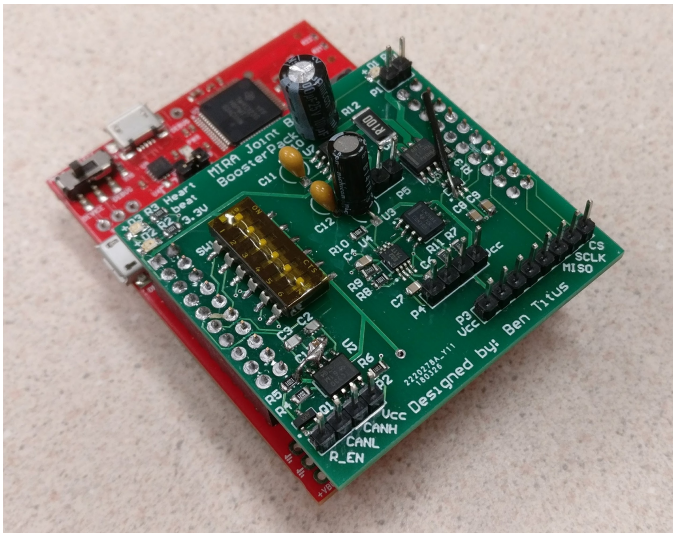

Figure 26: Joint control board Boosterpack on the TM4C123GXL Launchpad

Small problems existed in some of the PCBs during the first revision. The CAN transceiver PCB has the V_{CC} and GND pins swapped. Once this problem was identified, the traces were cut, wire soldered, and the schematic altered for another revision. The load cell amplifier PCB has a GND pin that was not connected to anything. The GND pin on the INA332

amplifier had an unconnected ground pin. This pin was connected with wire, and the PCB worked as expected.

The joint control board Boosterpack had the CAN transceiver V_{CC} and GND pins swapped as well as needing the motor direction pin to be moved from one pin to another. This change came about due to the Launchpad construction. Two of the pins were connected internally but labeled differently, so the motor direction pin was swapped in code, the trace was cut, and wire was soldered to the new pin. These changes can be seen in Figure 26. Another revision of the joint control board Boosterpack was made but not ordered due to time constraints. A repository containing all of the altium files for these PCBs can be found in Appendix A.7.

3.3.14 Joint Control Board Code

The joint control board code uses TI's RTOS for TivaWare [29]. The main reason for this decision was to gain access to the task synchronization that the RTOS provides to compartmentalize the code into different threads. A code flowchart can be seen in Figure 27. There are many threads running alongside one another, but they are all fairly separated in terms of their functionality. Each thread handles a single function, for example there is a thread for processing encoder data. This thread is released on a 1kHz software timer. The code repository can containing this code as well as a repository containing testing code can be found in Appendix A.7.

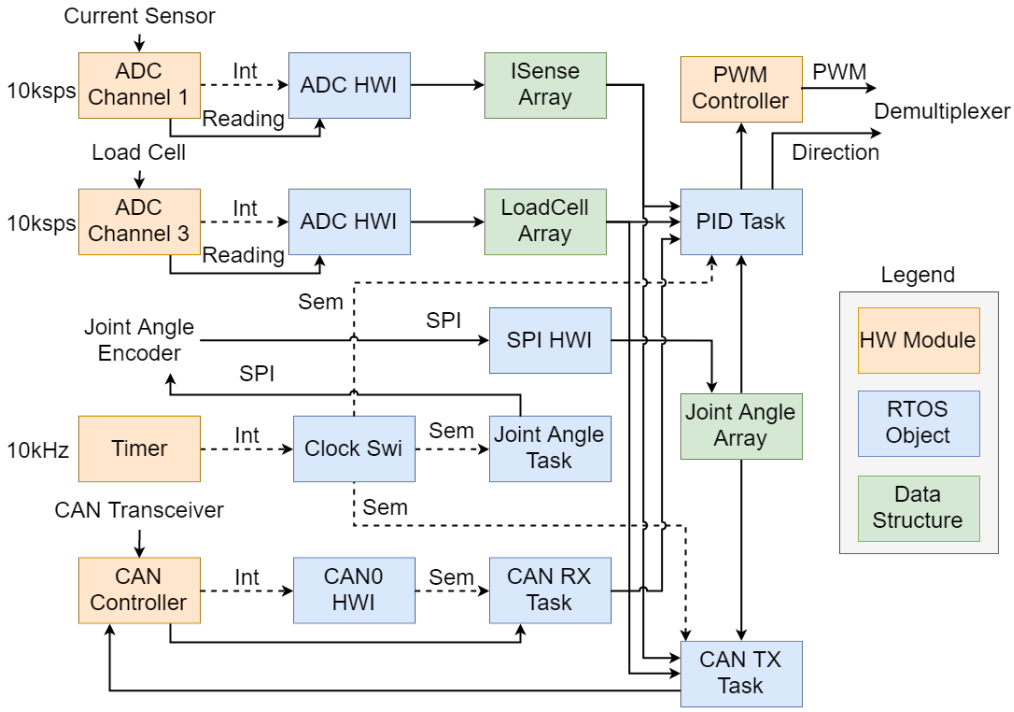

Figure 27: Joint control board code flowchart

3.4 CAN Bus

The CAN bus is the spine of the control system, carrying messages from the base to each joint and back from each of the joints to the base. Unit tests and careful integration were done to ensure that this tricky protocol was implemented properly and robustly while leaving room for future implementation of additional message types.

3.4.1 Determining Joint Placement with Message ID

CAN bus will be used to communicate between the base and joints. A limitation of CAN is that there is no way of determining the position of a module on the bus. This is important for controlling a robot arm since joint 1 is controlled differently from joint 2, etc. In order to determine the position of a joint board on the bus, another method is needed.

The CAN message ID contains 11 bits in the standard frame. We decided to use the upper 6 bits as a joint board number identifier, a number unique to that specific joint board, and the lower 5 bits as a message type. By doing this, we can tell the base the joint identifier number and position on the arm to route position update messages to the correct joint on the arm.

There were a few ways of accomplishing this. One would be to program each joint board with a different identifier number. This could get very tedious and confusing for having many joint boards, since we would have to change and track identifier numbers for each unique joint board. Another option was to use EEPROM to automatically store the identifier number and upload it via the USB cable used for programming the board. This could get complicated since we would have to write code to not only read in the identifier properly, but also store it in EEPROM properly. Instead, we decided to use DIP switches. DIP switches allow us to input the identifier number in binary and update them on the fly without reprogramming the board.

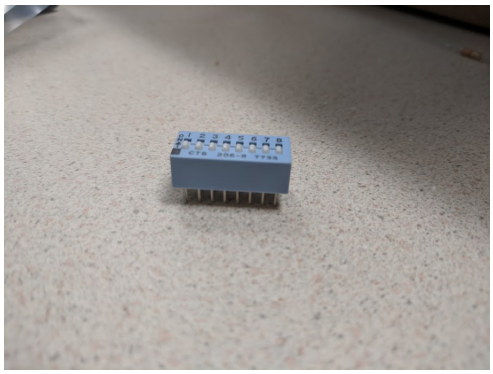

Figure 28: 8-channel DIP Switch

On startup, the joint board reads in the identifier bits and stores them in a variable. The

Table 11: CAN message ID breakdown

CAN ID	Message type
000000XXXXX	Reserved
XXXXXX00000	Init Encoder
XXXXXX00001	Init PID P constant
XXXXXX00010	Init PID I constant
XXXXXX00011	Init PID D constant
XXXXXX00100	Position update to joint
XXXXXX10000	Position update to base

identifier is then used as a mask for the CAN message receive IDs. The CAN controller compares incoming message IDs to the ID mask and ID value set when initializing the message receive object. When setting the message receive object, we can set the message ID to the joint board identifier, shifted up by 5 bits, and set the mask to only listen for messages that match the upper 6 bits. We can set the message receive object to receive messages with the joint board ID in the upper 6 bits, regardless of message type. Since there are 32 message objects available on the CAN controller of the TM4C123GH6PM microcontroller, we have a great amount of flexibility for listening to different types of messages.

3.4.2 Implementing a Simple CAN Bus

The preliminary CAN setup consisted of two TM4C Launchpads, one with transmit code and one with receive code. The transmitting board was set up for 1Mbps transmission with message ID = 2, and message length = 1 byte. The message data was a 4-bit value that incremented every time the message was successfully transmitted. A software delay was used to slow the transmission rate down to about every second. The receive board was set up for 1Mbps transmission with message ID = 0, message ID mask = 0, and message length = 1 byte. Setting both the message ID and mask to 0 signals the controller to accept any message. 4 LEDs were set up to see the CAN message data visually.

We had several problems getting this simple example to work. Initially, one of the jumper wires used for the CAN bus was broken, causing the CAN Hi lines on the transceivers to not be connected. The next problem was that the sample code provided by TI was not correct. When the CAN controller receives a valid message, it signals the processor with an interrupt. When a receive interrupt is processed by the example code, the interrupt flag is cleared and then the message data is read in. The problem is that when the interrupt flag is cleared, the new data bit that signals that there is valid data available is cleared. To fix this, the operations must be switched so that message data is read in before the interrupt flag is cleared. Once this fix was applied, CAN communications were functional.

During the debugging process, a logic analyzer (Saleae 8-channel logic analyzer) was used

to verify correct transmission of CAN packets. The logic analyzer software can be seen in Figure 29 decoding a packet with ID 2, data length 1, data byte equaling 7, and a proper ACK signal. The x marks a bit that was inserted to keep the timing consistent among the transceivers. Additionally, Figure 30 shows the ACK bit on the RXD line (channel 2) and no ACK bit on the TXD line (channel 0).

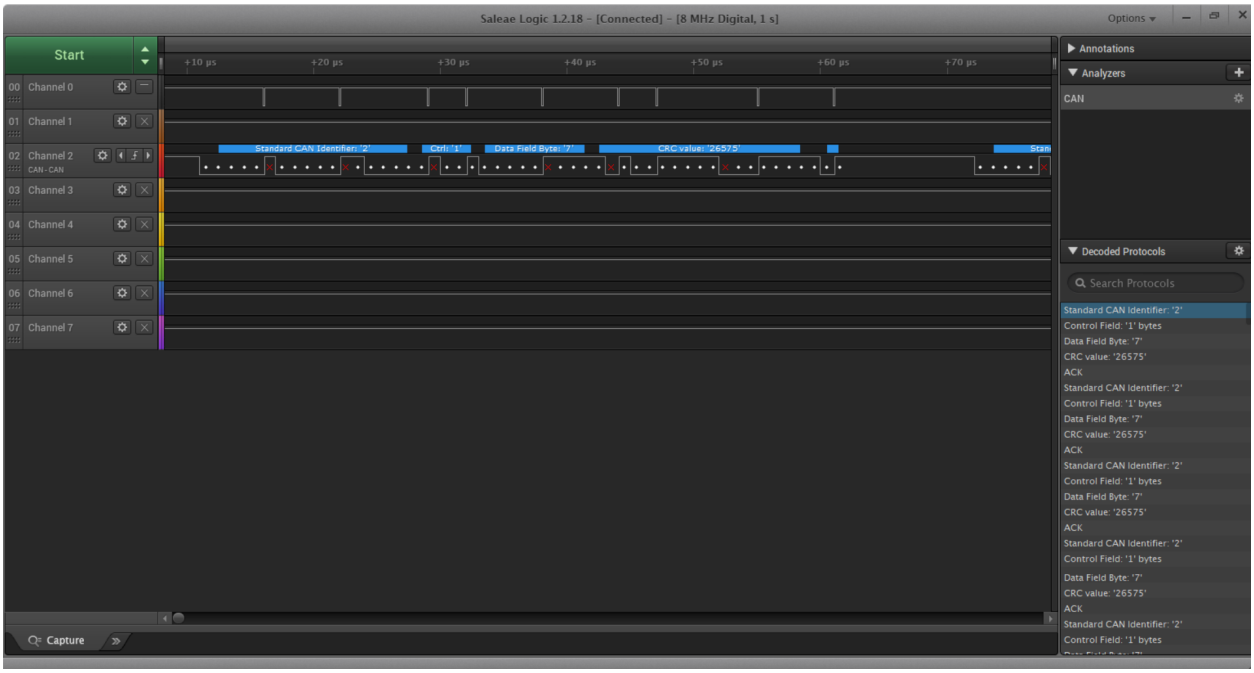

Figure 29: CAN bus as viewed in the logic analyzer software

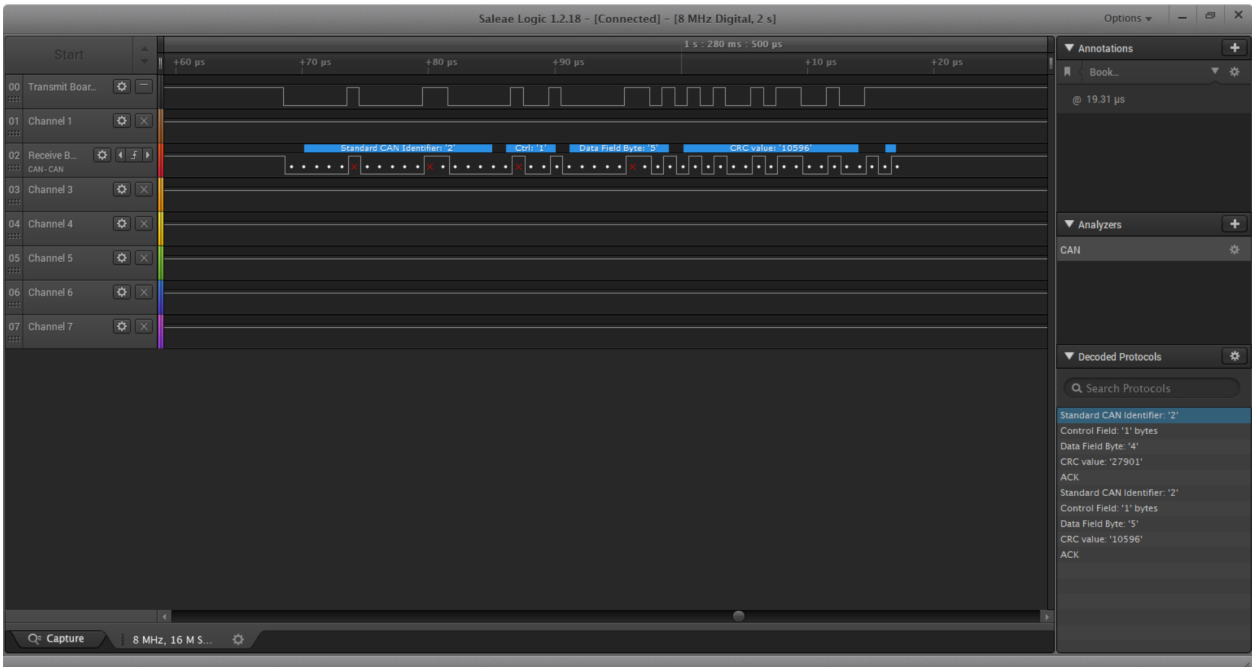

Figure 30: CAN TXD (channel 0) and RXD (channel 2) in the logic analyzer software

3.4.3 CAN Termination

CAN specification states that the CAN bus needs to be terminated on both ends by 120Ω resistors. The purpose of these resistors is to help mitigate signal reflections as well as pull the CAN Hi and CAN Lo lines together when the bus state is recessive. Since the resistors need to be at either end of the CAN bus, using normal resistors requires a static bus configuration. Our system will not necessarily have a fixed configuration, though, so a different solution was needed.

One method was to require the end user to attach a unique component, such as an end effector, at either end of the bus. This would effectively mean that an arm would always need a base module and an end effector to function properly. This rigid definition was not something we wanted to enforce on the end user, so instead we came up with a auto-disconnect circuit to disconnect the terminating resistor if another joint is added to the arm.

The way this works is through a MOSFET switch, seen in Figure 31. The MOSFET (part number BS170) drain and source connect the CAN Hi and CAN Lo lines through a 120Ω resistor and the gate is pulled up to V_{DD} through a 1lΩ resistor. When the next joint is connected, the gate is connected to ground and the MOSFET is opened, disconnecting the CAN Hi and CAN Lo lines. When there is no joint connected, the MOSFET is closed, effectively acting as a 5Ω resistor in series with the 120Ω resistor. Two of these new 125Ω resistances in parallel would lead to a resistance of $\frac{1}{\frac{1}{125}+\frac{1}{125}} = 62.5\Omega$ is within the tolerance

of the CAN specifications which states that the resistance must be between 50Ω and 70Ω.

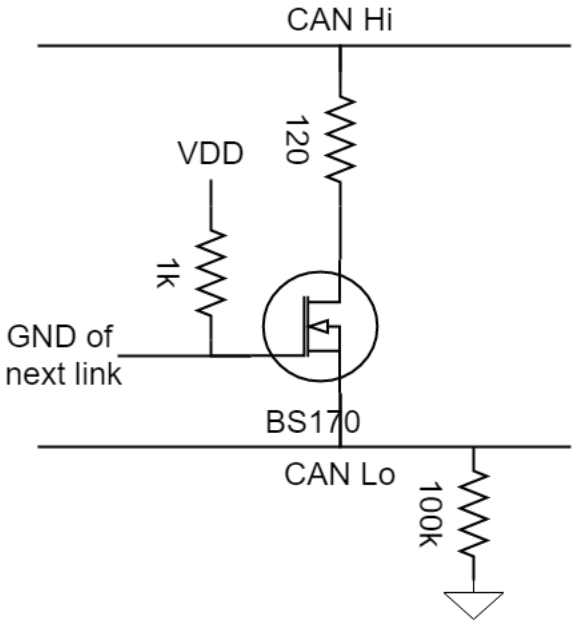

Figure 31: CAN bus auto-terminate circuit

3.4.4 Position Updates

The control system supports up to 6 joints at a time. In order to have a constant transmission speed for every combination of joints up to and including 6 joints, 6 joint update packets are sent every update. When fewer than 6 joints are used, the remaining packets are transmitted but never received. This is one place where using the CAN bus helps, since these messages don't need to be received by a specific receiver. Only the ACK bit needs to be sent when no errors are detected with packet structure, but this can be sent by any received.

Since all 6 joints were being updated every time, the minimum update time is the time it takes to transmit 6 CAN packets. This time was measured to be about 0.8ms with a transmission speed of 500kHz using the logic analyzer and can be seen in Figure 32. The joint update and response were both sent at a rate of 20Hz, 50ms apart, and can be seen in figure 33.

Figure 32: Position update from the base module

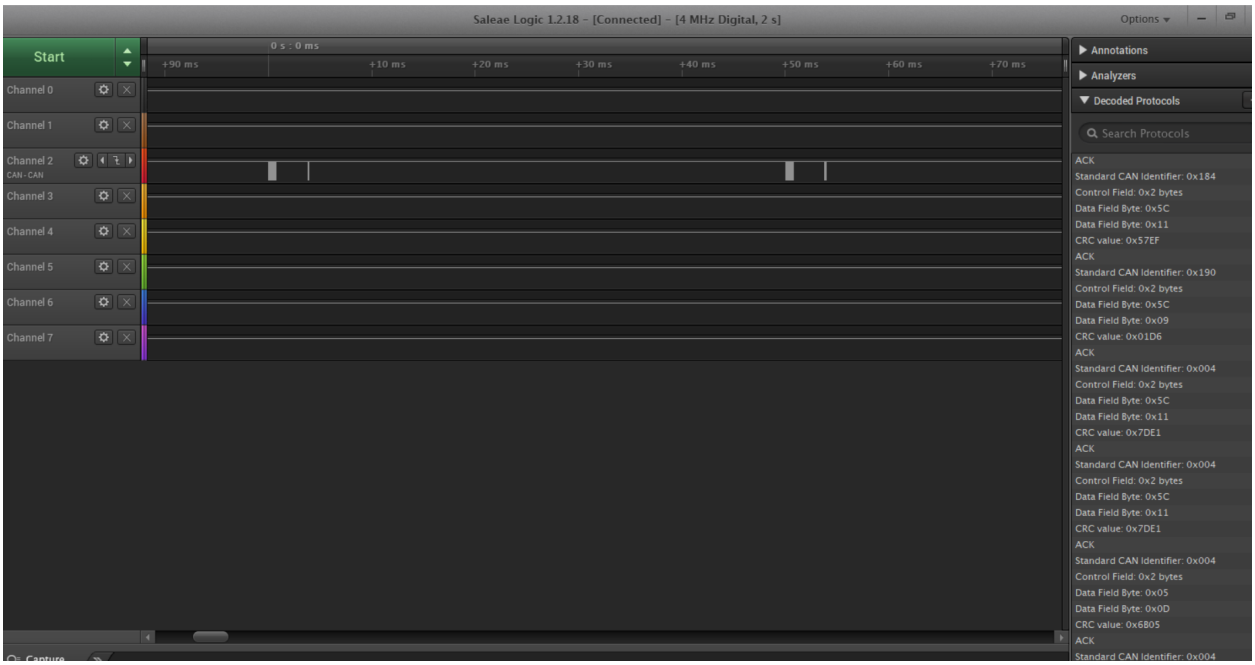

Figure 33: Position update from the base module with response from a joint

3.5 Base Module

The base module serves two main purposes. The first is to translate serial UART packets from the computer to the CAN packets for the joints. The second is to provide power and communication buses for the entire system. These two parts are crucial to the functionality of the arm.

3.5.1 Overview of Base Module Design

The base module is pretty simple from a hardware standpoint. It mostly consists of two parts, the microcontroller and the power supply. A CAN transceiver is needed, in addition to the microcontroller, but after that there is no more hardware except connectors. A block diagram can be seen in Figure 34.

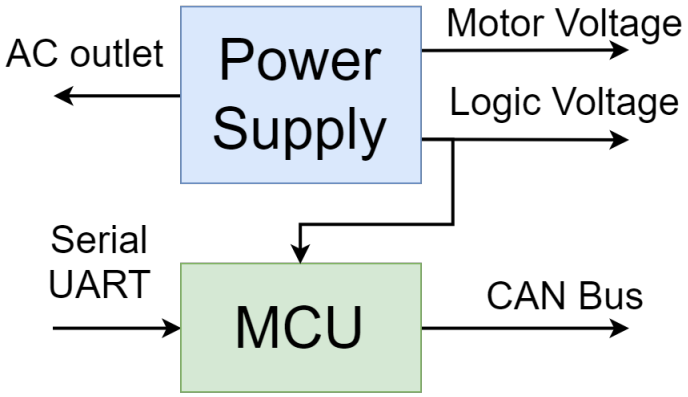

Figure 34: Block diagram of the base module

3.5.2 Base Module Code

The base module is also running TI RTOS for TivaWare boards. The main reason for this was to multithread the code to simplify programming [30]. By having multiple threads, the code can be compartmentalized into sections that handle different parts. Figure 35 shows the code flowchart. There are two main states, initialization and runtime. During initialization, all of the constants for each joint are loaded in. These include CAN ID, encoder offset, and PID constants. Once all 6 joints have been initialized, the code transmits that information on the CAN bus and switches into the runtime state.

In the runtime state, the base module reads in joint angle updates from the computer, then sends them along to the corresponding joint. The joints then send back their actual joint angle, which is parsed back into serial and sent to the computer. The code repository containing all of this code can be found in Appendix A.7.

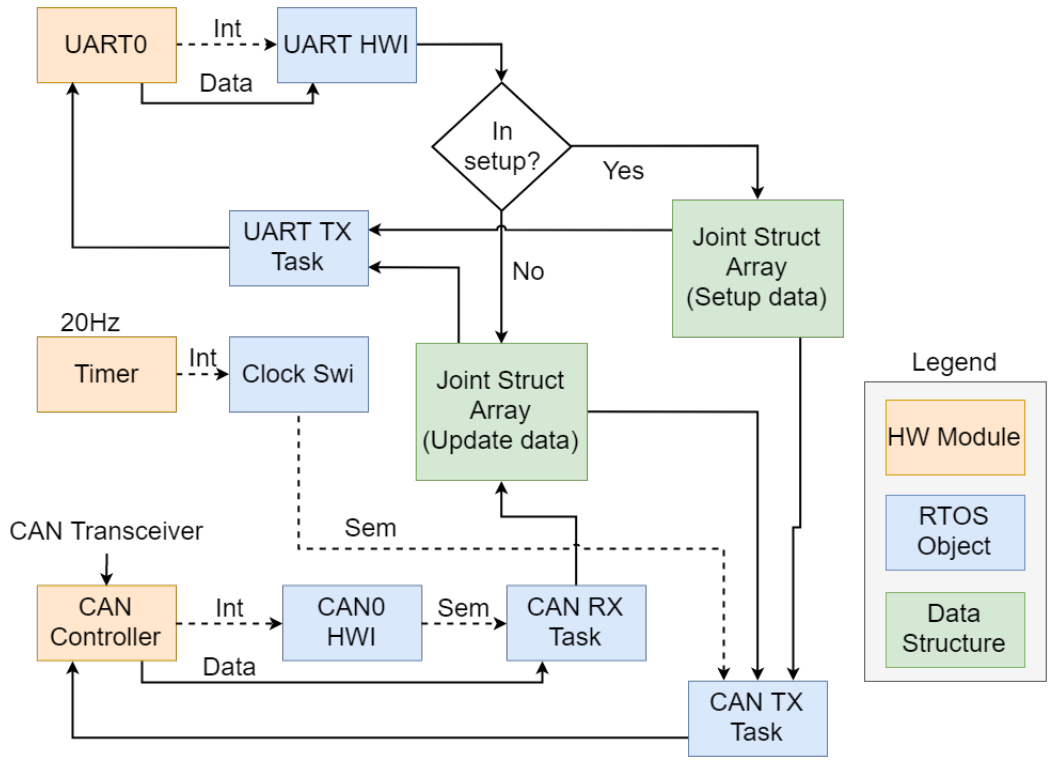

Figure 35: Base module code flowchart

3.5.3 Power Rails

Since the arm is of a variable size, the power rail must also be modular enough to accommodate this. To solve this issue, a modular power rail was designed, and a diagram can be seen in Figure 36. The main rail is made up of 16 AWG wire and the connectors are rated for 15A continuous current draw. The power rail and CAN bus are very similar, but with smaller amperage JST connectors and thinner, 22 AWG wire.

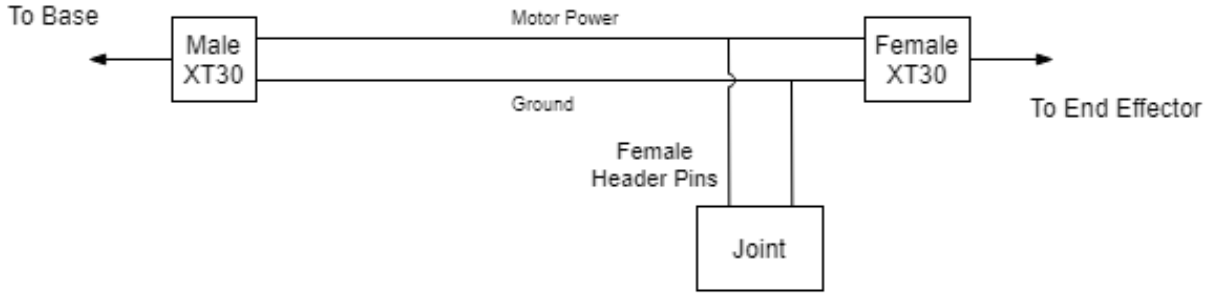

Figure 36: Modular power rail for motor power

3.6 Arm Structure

Arm structure is not something we wanted to fully define, since the end user is supposed to create their own arms, but there were some basic components that needed clarification. The first of these is that every arm must begin with a base module and have some combination of up to five additional joints connected. This allows the end-user flexibility in how they want to construct the arm without allowing them to add too many joints.

3.7 Software Application

The code library is another important part of what we did to make our arm work. It controls all of the electrical components via the sending of packets out to the base module over a Serial UART line. It handle a lot of the more involved calculations for controlling the arm like the forward and inverse kinematics. The repository containing all of the software code can be found in Appendix A.7.

3.7.1 Maven

Maven is a utility for Java-based projects that seeks to provide a uniform build system for the project. It accomplishes this by defining a project object model and a set of plugins that each Maven project shares. Therefore, Maven can provide a streamlined build environment for every instance of the project, allowing users to have the same build process across multiple different devices and environments. Maven could be compared to a flexible template for how a project should be arranged and what files should be included. We chose Maven for our Java project because it makes it much easier for our team to collaborate on the front-end side of the code. It also allows us to package into our program libraries that we used in our project so that there are fewer dependencies that the end-user must download in order to use our software [31].

3.7.2 Program flow

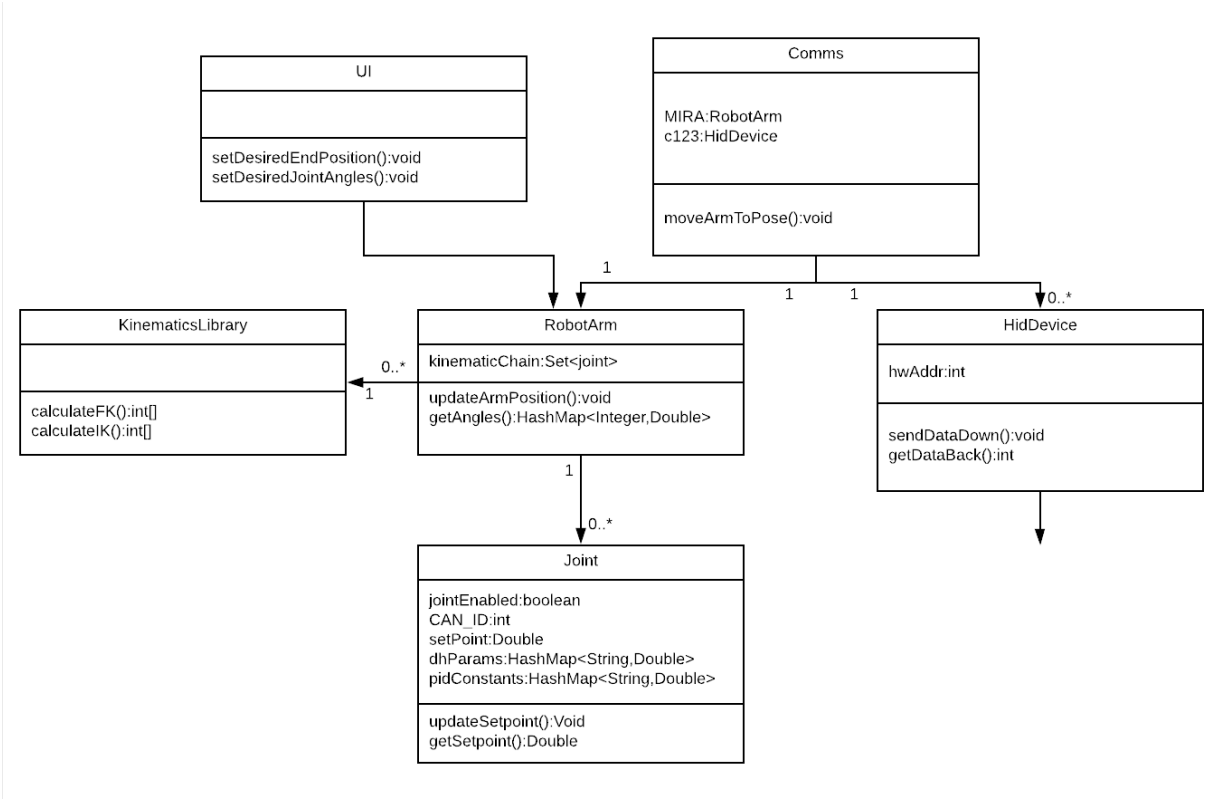

Figure 37: UML Diagram showing different classes and their relations

The front-end program running on the PC is written in Java. We used JavaFX to make the GUI. We chose to write this part of the program in Java because of the speed and reliability of a language that has many libraries and excellent Interactive Development Environments.

The main class starts the JavaFX Project. The JavaFX Project holds an list of joints which lists all the joints, which contains data about how the arm is configured. Each Joint object makes sure to tell the Comms object that it's time to send a message to the actual joint whenever new information about itself comes in. When the user enters new information about the arm into the GUI, the GUI's controller tells the joint object to change that information about itself. For example, if the user changes the setpoint of Joint 2 from 90 degrees to 112 degrees, the GUI will tell Joint 2 that its position has been updated to 112. Joint 2 will, upon seeing that its position has been updated, ask the Comms object to convey the new position information to the arm's base board.

The Joint object does not contain a Comms object inside itself. Rather, there is a single Comms object for the entire program to use. The Comms object follows the Singleton design pattern. A singleton is a class which can only ever be instantiated one time. Singletons are often used to hold configuration information about a program because they guarantee that if

one object makes changes to the singleton's settings then any other object that subsequently asks for those settings will get back the most up-to-date version.

In this case, it makes sense to use a singleton because we want to guarantee that there's only a single place in the code which tries to access the serial port at any given time. Another way to accomplish this same goal would have been to move all of Comms's functions to inside the JavaFX controller. There is only one controller object. Arranging the program this way would have violated Java's design principles and would have made writing the code a battle rather than an art form.

Each joint would need to hold a reference to the JavaFX controller inside itself. Referencing such an architecture-specific piece of the program within the core of the program's logic would be bad for future portability of the code.

3.7.3 Serial Communication

Serial communication is a very common protocol used to transmit data between a maximum of two devices over two lines, Rx and Tx. Serial communication is already available on our microcontroller through its universal asynchronous receiver/transmitter (UART). This device translates the Tx and Rx line into a parallel data bus that can interface with our microcontroller autonomously. The Java code interfaces directly with this UART over a USB line connected to both the Tiva board and the computer. The Java code holds a class called Comms which opens a specified serial port upon instantiation. This is implemented through the use of NRJavaSerial [32], a library created by Kevin Harrington used for serial communications over USB. Upon instantiation of the Comms singleton, we open the specified serial port on the computer and begin polling it at a baud rate of 115200 bits/second. The enables the port to send and receive data so that the computer can send and receive packets from our base board. The Java code holds a buffer which acts as a First in First Out (FIFO) queue that is constantly updated when new data is received so that we can read in data that is on the serial line [33]. Since we are only able to send individual bits at a time across the serial line, we need to encode and decode the data that we send. We have to encode the data from ASCII strings into their decimal equivalents before sending them out over the serial line. Then upon receipt of data we must decode the data before it is able to be built into a string which represents one packet. Once we have individual packets available to us as strings, we can easily use Java's string comprehension functionality to update the necessary parts of the code and properly encode data for sending [33].

3.7.4 Mutltithreading

Multithreading is the process of creating new threads in order to run code in parallel. Creating a new thread in Java involves instantiating a new thread object from Java's standard libraries and passing in the relevant information to the Thread through the constructor [34].

The reason that we needed to use a thread is the need for a mode where the computer communicates over Serial, writing out and reading in data that it receives at a constant rate. The problem with this style of coding is that it requires a while loop which continuously executes code that would normally block all other pieces of code from running. Since we need to concurrently run our GUI and modify the values that we are sending to the arm based on GUI inputs, we cannot have the processor locked up all the time sending and receiving serial communications. Therefore, we created a new thread where serial communications could be handled on an entirely separate process than the GUI. We create this thread after the initialization of our arm has been completed and start it when the arm has acknowledged that it is ready to begin communications. Therefore, we are able to uphold constant serial communications without causing our GUI to crash mid-operation. Doing this allows us to move the joint sliders pictured below and have them constantly send out new messages to the arm telling the selected joint to turn to the new position.

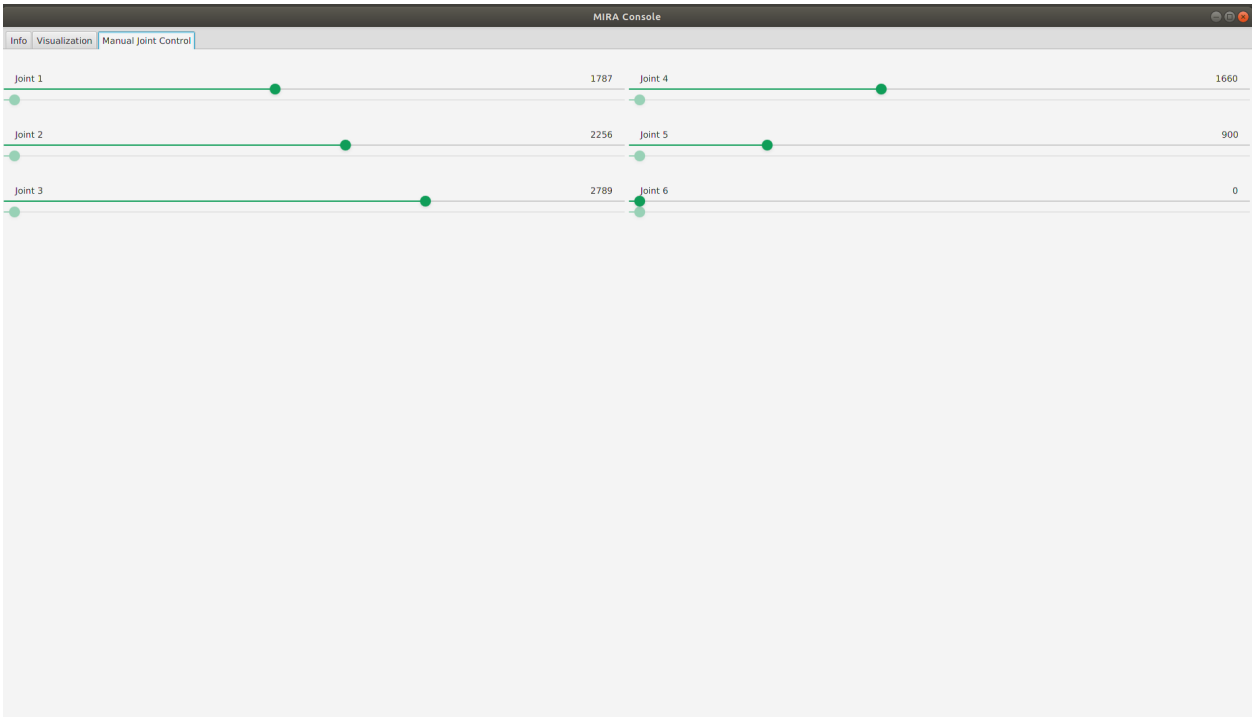

Figure 38: Image of the GUI tab which adjusts the setpoint for joints

3.7.5 Saving of Configuration

In order to store the information that we need to be persistent between different instances of the application, we used a library called Gson made by Google [35]. This library's primary use is to take data and store it in a .json file, a kind of simplistic database. We used this library to save the joint object which were the data containers about the configuration of the arm. All of the saving and loading is done via the GUI so that the user can either load information about an arm themselves, or they can change the values using the GUI text fields

and save a new configuration when it needs updating. Doing so allows the user to always be able to either modify and update the constants inside the data container that is the Joint object and have those objects be saved and loaded during each runtime of the application. Pictured below is the GUI tab which stores all of the constants for the arm such as encoder home values and PID constants, as well as has buttons to interface with the saving, loading and startup of our program.

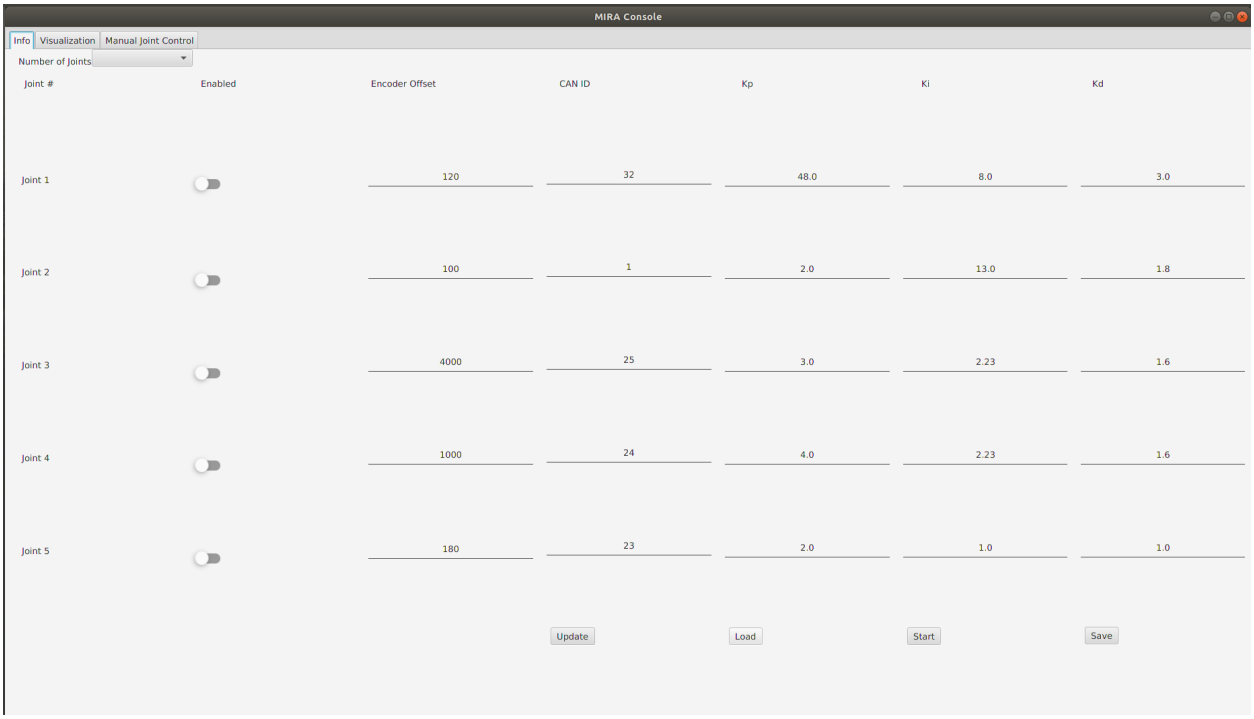

Figure 39: GUI config tab with save and loading

4 Testing

4.1 Acceptance Criteria

Acceptance criteria for this project will be broken into 5 major categories: Joints, End effector, Base, Software application, Code library

4.1.1 Joint Board

- Receive initialization information and joint angles from base
- Moves joint to angles
- Send position updates back to base
- Pass power and signal buses
- Capable of powering logic without powering motors
- Control board is the same for each joint
- Has CAN Auto-Terminate Circuit

4.1.2 Modify RBE3001 Arm

- Remove control system and replace with our own
- Add a link to the existing arm
- Replace currently implemented servo motors with brushed DC motors

4.1.3 End Effector

- Receives power and signal buses
- Keyed connection
- One input connector
- Terminate CAN bus
- Uses a joint board

4.1.4 Base

- Sends and receives joint angles to/from Personal Computer (PC)
- Receives initialization information from PC, then sends it to all joints on signal bus
- Outputs power and signal buses
- Converts AC wall power to system power bus
- Power supply and arm on/off switch
- Capable of powering logic without powering motors
- Array of indicator LEDs
- Terminate CAN Bus

4.1.5 Software Application

- Sends configuration information to the Code Library
- Sends individual joint angles or pose commands to robot through the Code Library
- GUI to adjust current arm configuration parameters
- Record and play back sequence of poses
- Acts as a front-end for code library
- Stretch goal: 3D model of arm moving in real-time

4.1.6 Code Library

- Receive configuration information from user, selects control constants, sends to base
- Able to control the robot: Receive joint status, send joint angles
- Calculate joint angles using kinematics
- Stretch Goal: Written so that it can interface with multiple languages

4.2 Motor Driver

In order to determine whether the performance of the motor driver was dependent on input frequency or other factors, the input frequency was increased again from 200Hz to 10kHz. This time, the motor performed much better than previous times where it stayed at a constant RPM at different frequencies. The RPM actually increased with an increase in frequency, as can be seen in Table 12. With this in mind, we decided to use a PWM frequency of 1KHz as a compromise between the higher resolution that comes about from lower frequencies (there are more timer counts between periods) and the higher rotation speed that would come from a high-frequency PWM system according Table 12.

Table 12: PWM frequency input at 50% duty cycle vs motor speed

PWM Frequency (Hz)	Motor RPM (rpm)
200	80
400	79
1k	81
2k	82
5k	86
10k	90

5 Conclusion

The goal of this project was to create a proof of concept for a platform that can prototype multiple different robot arms. We feel that we accomplished the most of the goals defined in the Project Deliverables section through our proof of concept. This project spanned a broad swathe of engineering disciplines and as such required lots of work in each of the three "pillars of robotics". Mechanical engineering, electrical engineering and computer science all played important roles in the design and implementation of our robotic arm. We made use of our respective backgrounds in this area in addition to a large amount of self-study required to fully realize our goal.

From the start of this project, we knew that it would not be a trivial accomplishment to make a modularly reconfigurable robot arm. We defined the goals that we wanted to achieve and tried to clearly lay them out in the Project Deliverables section. Very few of our ideas came from external sources because we wanted to create something novel. It was clear to us at the beginning of the project that if we were to meet this goal, we would have to approach many complex engineering problems and solve them using innovative and creative methods. In the end, we were able to reach most of our goals, but not without running into many errors which can be expected with a project of this scope. There were a few roadblocks which impeded progress and led to executive decisions to change how we approached certain problems. Principal amongst these was a project redesign in response to a change in project administration early B term. This redesign was not just a test of our dedication to the project but our dedication to our goals. It turned out that due to the shortened time span which we had to complete this project, we had to reevaluate some goals and scale others back so that they were achievable.

5.1 Future Work

In possible future iterations of this project, we urge teams who decide to take up this project to follow our goals outlined in the Project Deliverables section. We put great thought into these goals in order to make them fully encapsulate what we want as an end-deliverable for the project. All of this careful planning gave us a good outline for what we should be accomplishing in the project and therefore should be utilized by subsequent iterations to begin working more quickly. Overall, we would urge teams to expand more in the mechanical aspect of the arm. We believe it would be more beneficial to future projects to use the current electrical system or complete the design and fully integrate the TM4C123GH6PM microcontroller with the joint board, rather than start over with a new control system. There is also quite a bit of room to expand in the software application area, implementing more types of controls and making the GUI more intuitive and easy to use. Additionally, integrating inverse kinematics with the software would be good. By avoiding our mistake of spending so much time defining what we wanted out of the project, subsequent teams should be able to make use of the work we did over this past year.

References

[1] . F. STU, "Robotick rameno abb irb 120," February 8, 2016.

[2] RobotWorx, "Robotworx: Expert industrial robot integrator." www.robots.com. Accessed on September 11, 2017.

[3] Traclabs, "Reconfigurable modular manipulator (rmm)." https://traclabs.com/projects/rmm/. Accessed on Sep 11, 2017.

[4] D. Calzada-mariaca, M. Preston, and Y. Zhou, "Modular robotic arm," tech. rep., April 26, 2015.

[5] igus, "robolink robot components." www.igus.com/robolink/robot. Accessed on September 6, 2017.

[6] P. Cain, "Pot vs. sensor," *Electronic Products*, pp. 44,46, 2010.

[7] B. Sensors, "Choosing the right sensor technology." http://www.beisensors.com/customer-resources/bei-choosing-the-right-sensor-technology.html. Accessed on October 13, 2017.

[8] M. Howard, "Choosing the right position sensor." http://www.zettlex.com/articles/choosing-right-position-sensor. Accessed on September 17, 2017.

[9] S. Ziegler, R. C. Woodward, H. H. C. Iu, and L. J. Borle, "Current sensing techniques: A review," *IEEE Sensors Journal*, vol. 9, no. 4, pp. 354–376, 2009.

[10] J. Patrick, "Serial protocols compared," *Embedded Systems Programming*, 2002.

[11] N. Murphy, "Can we talk?," *Embedded Systems Programming*, 2003.

[12] C. Watterson, "Controller area network (can) implementation guide," tech. rep., Analog Devices, February, 2012.

[13] S. Corrigan, "Controller area network physical layer requirements," tech. rep., Texas Instruments, January 2008.

[14]

[15] D. Systems, "Solidworks." https://www.solidworks.com/.

[16] Ultimaker, "Ultimaker cura software." https://ultimaker.com/en/products/ultimaker-cura-software.

[17] Lulzbot, "Lulzbot taz 6." https://www.lulzbot.com/store/printers/lulzbot-taz-6.

[18] Texas Instruments, *Tiva TM4C123GH6PM Microcontroller Data Sheet*, 2014. SPMS376E.

[19] Allegro Microsystems, *High Accuracy, Galvanically Isolated Current Sensor IC*, 2015. Rev. 2.

[20] Texas Instruments, *DRV8872 3.6-A Brushed DC Motor Driver With Fault Reporting*, 7 2016. Rev. SLVSCZ0C.

[21] Texas Instruments, *1-OF-2 NONINVERTING DEMULTIPLEXER WITH 3-STATE DESELECTED OUTPUT*, 7 2012. SCES406K.

[22] Burr-Brown Products from Texas Instruments, *Low-Power, Single-Supply, CMOSIN-STRUMENTATION AMPLIFIERS*, 9 2001. SBOS216B.

[23] austria micro systems, *AS5055A Low Power 12-Bit Magnetic Position Sensor*, 10 2014. Rev. v2-06.

[24] Texas Instruments, *Tiva C Series TM4C123G LaunchPad Evaluation Board User's Guide*, 4 2013. SPMU296.

[25] Texas Instruments, *TivaWare Peripheral Driver Library*, 7 2016. SPMU298D.

[26] Texas Instruments, *TivaWare USB Library User's Guide*, 7 2016. SPMU297D.

[27]

[28] Purdue University, *PCB Design Specifications*, 7 2011. Rev. 1.0.

[29] Texas Instruments, *SYS/BIOS (TI-RTOS Kernel) User's Guide*, 5 2017. SPRUEX3T.

[30] Texas Instruments, *TI-RTOS 2.16 for TivaC Getting Started Guide*, 2 2016. SPRUHU5D.

[31] A. M. Project, "Maven." https://maven.apache.org/what-is-maven.html. Accessed on March 15, 2018.

[32] NeuronRobotics, "Nrjavaserial." https://github.com/NeuronRobotics/nrjavaserial. Accessed on April 20, 2018.

[33] Sparkfun, "Serial communication." https://learn.sparkfun.com/tutorials/serial-communication/all. Accessed on April 16, 2018.

[34] GeeksforGeeks, "Multithreading in java." https://www.geeksforgeeks.org/multithreading-in-java/. Accessed on April 15, 2018.

[35] Alphabet, "Gson." https://github.com/google/gson. Accessed on April 15, 2018.

Appendix A Early Project Iteration

A.1 Introduction

The goal of this project is to create a cost-effective, modular kit of parts that can be used to create a robotic arm. In this paper, we will be using the word "Joint" to refer to a piece of the arm that has a motor, and we will use "Stick" to refer to the part of an arm that connects two joints. The joints provide degrees of freedom for the arm while the sticks space out the joints. Joints can connect to joints and sticks, but sticks can only connect to joints. End-of-arm tools can be swapped out, but not during operation. In addition to a physical kit, we will create a GUI for easy configuration and basic control of the arm. The base will communicate with a computer running control code either through the software application or code library.

We aim to construct our kit with smart joints and dumb sticks. This will be accomplished by designing a controller board that has all the necessary components to control one motor. This controller board will be placed on each joint and connected to a main processing unit in the base that handles control for the entire arm. The full set of components for this kit are outlined in Appendix A.7.

We will create a software application to interface with a constructed arm. The user will input how they have constructed their arm into this application and then be able to do some simple control. Another feature of this application will be the ability to record a series of poses for the arm to perform. In addition to this software, we will also create some programming libraries to allow users to control the arm with an actual program.

In this document, we will outline some existing robot arms and highlight the differences between these arms and our arm kit. Next, we discuss what work there is to be done on this project. After discussing the work to be done, we will state how this work will satisfy the capstone design requirements for each of the three disciplines represented by our group members. Then, we state the constraints we expect going forward with this project. Next, the acceptance criteria for any deliverables at the end of this project will be outlined. Finally, we will state an estimated timeline for this project.

A.2 Modular Robotic Arm

This project aimed to close the market gap between inexpensive toy robot arms and expensive professional grade industrial arms. The group aimed to do this by designing a single joint that could be used to assemble a robot arm. Ultimately, a single DOF joint that was heavy, difficult to manufacture, and expensive to produce was designed and constructed. In their future recommendations section, the group stated that the goal of designing a modular robot arm was possible but their design was not the solution [4].

A.2.1 Our Robotic Arm System

Our modular robotic arm kit aims to offer a completely different experience compared to existing products and projects. The system maintains a low cost while providing a versatile platform for beginner engineers or rapid prototyping professionals. This is achieved by avoiding expensive proprietary software and subtractive manufacturing; favoring off-the-shelf parts, 3D-printed structures, and freely available software. Providing custom-built software for controlling the arm creates a plug-and-play environment suitable for most any skill level.

A.2.2 Control board

The control board is meant to be implemented as an independent module that interfaces with a main controller module. Its tasks are to send and receive data from the main controller and control the position of a single motor. As such, the main factors that must be taken into account when designing the control board are methods of measuring joint position and motor torque, as well as communicate with an off-board controller. Motor torque is proportional to motor current. Therefore, the motor torque will be calculated from the measured current through the motor.

A.2.2.1 Joint Position Detection

Angular position sensing must be used to determine the joint angle of the motor. There are several commonly used methods of determining angular position, including potentiometers, optical encoders, and Hall effect sensors [6–8]. A comparison of the different angular sensors can be seen in Table 13.

Potentiometers are very commonly used to measure angular position due to their simple implementation and low cost. In addition to being low cost, potentiometers provide high linearity and accuracy [8]. Although generall robust, these sensors do not lend themselves well to many, rapid adjustments or mechanical vibrations. Both of these significantly reduce the lifespan of the sensor [6,8]. The situations potentiometers excel in are those that require an easily adjustble voltage at low to medium adjustment frequencies, such as settings nobs on control panels or analog reference voltages as trim potentiometers [6].

Hall Effect sensors are less commonly used, and consist of a bipolar magnet rotating above a Hall effect sensor with the axis of rotation perpendicular to the plane of the sensor. Since there is no contact between the rotation and the sensor, these types of sensors have very long lifespans [6]. Unfortunately, these sensors do not provide high resolution since they are susceptible to electromagnetic interference and temperature, and also have some hysteresis [8].

Optical encoders are another method of measuring angular position. These sensors consist of a beam of light that shines on a slotted disk so that as the disk rotates, the slots break the light beam. These sensors can have very high resolutions and are resistant to

shock and vibrations [7]. Like magnetic sensors, these sensors have very long lifespans since there is no mechanical connection on the sensor [6]. Unfortunately, these sensors are susceptible to foreign particles blocking the light beam from sensing the slots and causing incorrect readings. The most common kind of optical encoder, the Quadrature encoder, does not sense absolute position; it can only read relative position, meaning that a quadrature encoder would need to be combined with some other sensor in order for the robot to be able to sense its joint angles correctly. Other encoders called Absolute Encoders do not have trouble reading absolute position, but they are prohibitively expensive. [8].

Table 13: Comparison of different angular position sensors

Sensor	Cost	Linearity	Accuracy	Lifespan	Notes
Potentiometer	$	Depends on ADC	Moderate	Short	Repeated motion at the same angle can lead to failure
Encoder	$$$	Very High	Very High	Long	Cheap ones can't sense absolute position
Hall Effect Sensor	$$	High	High	Very Long	Requires special attention to surrounding magnetic fields when mounting

A.2.2.2 Current Sensing

Current sensing can be done in many ways. The most common way is by using a shunt resistor and an amplifier. A variant of this method is to use the resistance inherent in the wires or traces as a shunt resistor. Another common method of current sensing is to use a Hall effect sensor [9].

Shunt resistors are used in either high side or low side configuration. They are simple to integrate, low cost, and capable of measuring both AC and DC currents. The downsides to this method are relatively large insertion loss that increase exponentially with current, large thermal drift that must be compensated for, as well as large system noise from amplification. There are two main implementations of shunt resistors, high side and low side [9].

Low side current sensing means that the shunt resistor is placed in the return current path. This method is simpler to implement since the voltage on the shunt resistor is with respect

to ground, so it can simply be amplified. Some problems exist with this, however, since the resistor separates the current path from ground. In this configuration, the circuitry used to measure the voltage on the shunt resistor will not report a fault if the system experiences a short circuit [9].

High side current sensing means that the shunt resistor is placed on the forward current path. This configuration is able to detect short circuit faults, an advantage to using this configuration over low side current sensing. An additional advantage is that the return current path is directly connected to ground. The downside to high side current sensing is that it requires a differential amplifier since the voltage across the shunt resistor is very close to supply voltage. [9].

Trace resistance sensing is very similar to using a shunt resistor, but there are some slight differences. Since there isn't a way to control the resistance of a copper trace, the system must be calibrated after being assembled. Another key difference is the amount of amplification needed. Copper traces have very low inherent resistance, so a very large amplification must be used. This large gain imposes a limitation on the maximum measurable bandwidth set by the gain bandwidth product of the amplifier [9].

Hall effect sensors are commonly used to measure current as well. These sensors can measure current intrusively or non-intrusively, as well as in open loop or closed loop configurations. Non-intrusive devices measure current by wrapping wire around a toroid that focuses the magnetic field on a sensor in a break in the ring of the toroid, or placing the Hall effect sensor on top of the current to be measured. These work fairly well, but are very susceptible to noise from magnetic fields upwards of 10cm away. Methods of shielding these sensors exist, but are complicated and expensive to implement. Intrusive sensors route current through the device and measure the generated magnetic field with a Hall effect device near the current path. Open loop applications take the voltage generated on the Hall effect sensor and condition it to whatever output is needed. Closed loop sensors reroute the sensed current to a secondary coil that is used to generate a proportional current to the measured current. This proportional current is then used as feedback to reduce error [9].

Insertion loss caused by these sensors is very small. Since these sensors measure current by induction, they can only measure current in a specific frequency band, and high currents at high frequencies can cause these devices to overheat. Most of these frequencies are DC to some upper limit determined by the physical characteristics of the sensor, usually around 100kHz. These sensors cannot be used on their own, since they have an inherent voltage offset, called misalignment voltage, and suffer from high thermal drift. Integrated ICs that compensate for these factors are fairly widespread, allowing for very easy integration [9].

A.2.2.3 Off-Board Communication

There are many types of communication protocols that could be used to communicate with the main controller. Common protocols include SPI, I^2C, RS232, RS485, and CAN. Of these, SPI and I^2C are meant mostly for chip to chip communication while RS232, RS485,

and CAN are all meant for module to module communication [10]. A comparison of these protocols can be seen in Table 14.

SPI is a full duplex, synchronous serial link consisting of 3 lines, SCLK, MOSI, MISO, and an additional line for every peripheral, CS. Data rates of up to 10MHz or more are possible due to the elimination of addressing with the CS lines and dedicated clock line [10]. Using SPI for controller-to-controller communication presents a problem, however. Since the data transfer rate is controller by the master, the slave could fall behind on processing data. This can be avoided by only transmitting data one direction at a time. Typically, SPI is limited to onboard communications since its signal degrades fairly quickly over distance [11].

I^2C is a half duplex, synchronous, multi-master bus consisting of a clock and data line. Data rates of up to 3.4MHz can be reached, and each device has a unique address or multiple addresses to avoid overlap. An interesting aspect of I^2C is clock stretching. Clock stretching is when a slave pulls the clock low to stall the master until it has enough time to process information. Typically, I^2C is limited to onboard communication since its signal degrades fairly quickly over distance [10].

RS232 is a common full duplex interface that consists of two transmitter/receiver pairs. The protocol limits communication to 1 sender and 1 receiver per line. Data rates of up to 115.2KHz are possible at a range of up to 200ft. Data is typically sent in 8N1 format with 8 data bits, no parity bit, and 1 stop bit or 7E1 format with 7 data bits, even parity bit, and 1 stop bit [10].

RS485 is a full duplex multi-master protocol that consists of up to 32 transceivers on the bus. Data transmission rates of up to 10Mbps and distances of up to 4000ft are possible. Transmission can be reduced to half duplex by removing one transceiver at each node. Data is sent much the same as in RS232 with either 8N1 or 7E1 being common formats [10].

CAN is a half duplex multi-master bus protocol that allows for many nodes to connect and send data on the two transmission lines. Messages are sent with unique addresses that also act as arbitration for bus priority. Packets are fully defined with 11 or 29 bit addresses, 0-8 bytes of data, and some additional control and verification bits [12, 13]. Data rates of up to 1MHz and distances of up to 3000ft are possible. Multiple error checks are implemented at the hardware level since packets are predefined, allowing the controller to load a transmit buffer and let the transceiver send a message or wait until a receive buffer is full before reading the message [11].

Protocol	Max Distance	Max Speed	Wires needed	Notes
SPI	Within circuit board	10MHz	SCLK, MOSI, MISO, + 1 CS for each node	No addresses needed
I^2C	Within circuit board	3.4MHz	2	Address included in message
RS232	200 feet	115.2KHz	4	Can include parity bit
RS485	4000 feet	10Mbps	4	Can transmit fast or far but not at same time
CAN	3000 feet	1MHz	2	Resilient signal

Table 14: Comparison of off-board communication protocol performance

A.3 Description of Work

There are many different ways to accomplish the goals we set. We decided to design several "smart" joints, "dumb" sticks of various lengths that passes signals through from joint to joint, changeable end of arm tools, a base for routing messages to each joint as well as providing power to the entire system, and a computer for performing complex real-time calculations. We plan to design every component listed in the kit of parts found in Appendix A.7.

We decided on designing two different joints (twist and rotation), with keyed connectors so they can connect in one of four orientations. Two kinds (straight and right angle) and three different lengths (75mm, 150mm, 225mm) of sticks will be designed as well. Joints can connect to sticks and joints, but sticks can only connect to joints. As such, the connectors will have to be designed with this in mind. Each connector will have to make a strong physical connection as well as a solid electrical connection to send power and data through to each Joint.

Each joint will have a control board that allows it to connect to the main communications line running through the system and control the motor on the joint. This board will have all of the necessary components for controlling one motor and communicating with the base. The base will act as an interface between the computer that is performing all of the complex calculations and the joints that are controlling their positions. The computer will

need to have a USB port and be able to run Python programs.

We will develop some software for controlling the arm with a GUI that will run on the user's computer. This software will have a simple control interface for moving the arm, and some configurable settings to act as inputs for the kinematics equations. In addition to this we will develop a code library for end users to interface with in their own code.

A.4 Methodology

A.4.1 Kit Components

We decided to break a robot arm down into its component parts. We came up with the main parts of our kit: sticks, joints, a base, and end of arm tools. This breakdown was to try to maximize modularity while keeping the pieces relatively simple. By separating the joint from the stick, we can have multiple sticks, which are easy to manufacture, of different types and lengths to offset a few types of joints, which are difficult to manufacture. The base is necessary to send and receive computation control from a computer. Multiple end of arm tools are needed to provide functionality to the arm besides movement.

A.4.2 Connectors

The connectors are vitally important to the functionality of this project. A good connector will need to make solid mechanical and electrical connection between parts while also providing the ability to quickly connect and disconnect parts. In addition, the type of connection we choose will affect the modularity of the system as a whole. The important things to note while deciding on criteria for the connector are how/if they will be keyed, how they will pass electrical signals, where exactly the connector will be on the joints, and how the connector will be secured.

A.4.2.1 Connector Position on Joints

Connector position is the first major design choice we had to make. They can be positioned either on the axis of rotation of the joint or off the axis of rotation of the joint, and selecting one method versus the other vastly changes the way that the connector would work. Putting the connector ON the axis of rotation means that the connector would connect to the joint axis directly, while putting it OFF the axis of rotation means the connector would connect to a piece that is connected to the joint axis.

The advantage that placing the connector off the axis of rotation has over placing the connector on the axis of rotation is that the joint will be a solid unit. Having the joint be a solid unit seems a better design choice than splitting it in half, so we decided to go with putting

connectors off the axis of rotation.

A.4.2.2 Securing the Connection

One of the important aspects of a modular system is how easy it is to connect or disconnect parts to or from that system. The main options for quick connections are requiring no additional hardware, requiring a single screw, and requiring slots and pins.

The most obvious solution is to connect parts with no additional hardware required. This creates a very complicated design challenge since using no tools means the user would have to secure any connection with just their hands. This can weaken the joint mechanically. The advantage to this method has is that it is fairly quick.

A step down in the simplicity solution is to require a single screw to join two pieces together. This is still simple and pieces can be connected somewhat quickly, but does require a tool to connect pieces. The main advantage is that screws hold parts together very well and the mechanical integrity of the connection should be held.

Another option is to design the joints and sticks in such a way that they slot together and are held in place with pins. This requires additional hardware, but no tools. This should keep pieces together fairly well while still allowing connections to be made quickly and easily.

A.4.2.3 Keying the Connector

Keying the mechanical connection between joints, sticks, and the base changes how modular the system is overall as well as how many unique components will be needed in each kit. Not keying the connection is not an option since this would allow the user to connect the pieces together in any orientation and the orientation needs to be known in order to accurately control the arm. This leaves two main options for the connections: keying for 1 orientation and keying for 4 orientations.

Keying the connectors for 1 orientation means that three different kinds of revolute joint must be design to fully represent the ways a revolute joint can move in 3D space. Essentially, this would mean each different joint would rotate about a different axis relative to the connector axis. The modularity of the kit is impacted quite negatively by doing this, since each joint can only connect in one way and therefore cannot be used where another type of rotation is needed. This design is quite simple, however, since the connectors don't need to be rotationally symmetric about any axis.

Keying the connectors for 4 orientations presents a slightly more challenging design problem, however. The connectors would need to be evenly rotationally symmetric 4 ways about the axis of connection in order for this design to work. 4-way keyed connectors will bring the

number of unique joints down from 3 to 2. Doing this does help with the modularity of the design, though, since the rotational joint can be implemented to rotate in either axis perpendicular to the connector axis. A disadvantage of this configuration is an increase in complexity.

Since additional complexity when designing is less important that the overall modularity, we decided to go with a 4-position keyed connection. This allows a single rotational joint and only 1 right angle stick design so that the user can construct many different kinds of arms from these simple parts.

A.4.2.4 Passing Signals

Connectors also need to pass the power and signal buses through from joint to stick or joint. This can be done in one of a few ways, including: rigid mechanical connectors, loose wires running along the outside of parts, wires running inside of parts, and wires connecting internal bus bars.

Rigid mechanical connectors for passing signals would make connection when parts are connected together. These connections would have to be evenly rotationally symmetric 4 ways about the axis of rotation since the mechanical connectors are. A disadvantage of these connectors is that they rely on the integrity of the mechanical connection to pass electrical signals properly. If the connection flexes or bends too much then the electrical connection could break even though the mechanical connection is still mostly intact. Another disadvantage is that this is the most costly option for passing electrical signals, requiring 4 connectors per connection.

Loose wires along the outside of the parts have several advantages over rigid mechanical connectors. The first of which is that they only require one set of connections per connector. This reduces the cost of each connector significantly. The main disadvantage of this kind of electrical connection is that the wires could get snagged on something since the system is supposed to be active and moving. Another disadvantage is that the wires need to have enough slack to move with the arm without limiting the arm's movement.

Wires running through the parts that pop out at the connectors is another option or passing signals along the system. This option is practically the same cost as external wires, but doesn't have the problem of wires snagging on the environment. Unfortunately this doesn't solve the problem of wires needing lots of slack to allow for movement of the whole system.

Short wires that connect some internal bus bars provide a more expensive solution to this problem. This would remove the problem of wires needing slack for the entire system. Instead, wires would only have enough slack for one joint. Doing this does bring some complexity issues, however, since the bars would have to be designed into the system and not added on at the end.

For our design we decided to use internal wires running the length of the system. The

low cost and simplicity of this solution outweighs the negatives of having to add lots of extra wire to account for movement of the system.

A.4.3 Sticks

Sticks are the things that connect joints together and space joints out. They do not have any electronics on board; they simply pass power and communication wires along to the rest of the arm. They need to be strong, light weight, and inexpensive.

Sticks have an input side and an output side. Two kinds of sticks will need to be created: one will be straight, and one will have a right angle at the input side.

Table 15: Comparison of materials to construct sticks

Stick Material	Cost/kg	Cost/ 20mm	Rigidity	Complexity	Weight
3D-printed PLA[1]	\$20/kg	\$3	Might break	Low: Very few constraints on possible designs	150g [1]
PVC[2]	\$7	\$0.70	Bends over time	Connect/ Disconnect easily	106g
Carbon Fiber[3]	\$66	\$6	Strong, but possibly too thin		58g
80/20[4]	\$2.46	\$2	Not going anywhere	Nice connecting options	154g

We choose to use 3D-printed PLA. While it's not the lowest cost option, nor is it the lightest one, its high configurability makes it the ideal material for our needs - especially given its availability for potential customers; anybody with a 3D-printer would be able to make one of our arms. Also, while aesthetic concerns should not be the only factor, we're allowed to consider the way the final product would look. An arm made from PVC would reflect poorly on all the involved parties.

[1]This number assumes 100% infill. The actual number will almost certainly be lower.

[2]http://www.homedepot.com/p/Formufit-1-in-x-5-ft-Furniture-Grade-Sch-40-PVC-Pipe-in-White-P001FGP-WH
205171542?cm_mmc=Shopping%7cTHD%7cG%7c0%7cG-BASE-PLA-D26P-Plumbing%7c&gclid=
Cj0KCQjwx8fOBRD7ARIsAPVq-Nlw_xbuCOf-QHORvUW4gQ4Dx7SiZt_vqQ3OvxBdTW-eckQhdp5WWFYaAs9DEALw_
wcB&gclsrc=aw.ds&dclid=CIagtKLB09YCFUuraQodN_MAOQ

[3]https://www.rockwestcomposites.com/45552?gclid=Cj0KCQjwx8fOBRD7ARIsAPVq-NmCNUg6ULxgd9udG-xSuPtJuHKgC
zXPDgRr2CmKU0tSXX-waAgb9EALw_wcB

[4]https://8020.net/1010.html

A.4.4 Motor Selection

Motor with Encoder	Voltage(V)	Speed No Load (rpm)	Current No Load (A)	Stall Current (A)	Stall Torque (oz-in)	Gear Ratio	Weight (oz)	Price ($)
Actobotics Planetary Gear Motor	12	26	0.21	4.9	583	455:1	???	49.99
Lynxmotion Brushed DC Motor	12	10.5	0.22	2.5	500	264:1	???	55.76
HD Premium Planetary Gearmotor	12	313	0.52	20	416.6	27:1	11.64	59.99
HD Premium Planetary Gearmotor (2)	12	12	0.54	20	8110.2	721:1	13.4	59.99
HD Premium Planetary Gearmotor (3)	12	84	0.53	20	1347.1	100:1	???	59.99
Motor Without Encoder								
Precision Planetary Gearmotor	12	26	0.21	4.9	583	455:1	3.53	27.99
Premium Planetary Gearmotor	12	32	0.21	4.9	472.1	370:1	3.53	27.99

Table 16: Comparison of Possible Motors

To choose our motors, we looked for high-torque, low-cost DC motors. We chose to go with DC Brushed motors to control our arm because they are quiet, low-cost, vibration free and fairly efficient. We also considered using Dc Brushless motors as well as Stepper Motors, but each had their own pros and cons. Brushless motors cost much more than comparable brushed motors, and require complicated control logic to operate. Stepper motors were a good option due to their ability to be backdriven and their built in discrete steps for controlling. But, they do not operate well under conditions where the load changes significantly in a short period of time and also require external control to keep track of the position.

Once we decided to use brushed DC motors, our next step was to find suitable motors that fit our criteria of high torque and low cost. We found 2 categories of motors that seemed to fill these requirements, planetary gearbox motors and spur gearbox motors. Planetary gearboxes work by having multiple "planet" gears revolving around a central "sun" gear that rotates in place. They are named for their resemblance of the planets orbiting around the sun. All of the "planet" gears are held in place by an outer "ring" gear that acts to keep the "planet" gears in contact with both the "sun" and the "ring". By having these idler gears rotating around a central axis, you can have torque transferred linearly, without the need for offset shafts, greatly reducing the total size of the gearbox. With multiple gears transferring the torque load at one time, the individual load on each tooth is lowered making these perfect for high torque applications. Spur gearboxes on the other hand use linear offset shafts that transfer the entire torque from one gear to the next until the output shaft in a direct chain. This means that they wear out much faster since the torque load is much higher on individual gears and teeth. Therefore, since we need a reliable high torque motor, we decided to go with planetary gear motors.

Once we had made the decision to go with a planetary gearbox brushed DC motor, we made a chart as seen in Table 5 of possible motors that had high stall torques. One final decision that we had to make was whether or not to purchase a motor with a rotary shaft encoder. Rotary shaft encoders provide easy control over DC motors by relaying the position of the shaft before the gearbox on the motor. This allows for high resolution control in the case of high gear reductions but also costs a fair amount extra to purchase with the motor. Considering that the motor is just a part of the joint and we care more about the position

of the overall joint rather than the motor itself, we decided to lower cost and go with a less expensive non-encoder motor. By doing this we are moving the point at which we control the joint system from the motor to the joint if we use an absolute encoder on the joint shaft. This results in a closed loop control system which is better suited to controlling the arm in our situation and is more cost-effective.

A.4.5 Control Board Part selection

Selecting the types of sensors to use for the control board was a very important step of the control board design. There are many different types of sensors to accomplish each major goal that the control board must accomplish.

A.4.5.1 Joint Angle Sensor

Potentiometers seem like a good choice due to their simplicity and high accuracy capabilities. However, they do not lend themselves well to this application because of how quickly they wear out. Over time, as the joints move to different positions, the potentiometers will wear out quickly and cause inaccurate readings. Additionally, long lifespan and high resolution potentiometers can be very expensive. Furthermore, potentiometers are large and can be difficult to mount. Finally, the hard stop on the potentiometer means the joint angles will be limited to a certain range (typically about 270 °for single turn potentiometers).

The next obvious solution is to use optical encoders because they will not wear out and offer very high resolution capabilities. These sensors are not well suited for this application, however, since they are typically expensive, especially for high resolution encoders - and ones that are capable of reading absolute position. Additionally these sensors are somewhat bulky and would take up too much space in the closed environment of a joint.

This leaves us with Hall effect sensors. These sensors are very small and moderately high resolution while also being a contact-free sensor, so wearing them out will not be a concern. A main concern with Hall effect sensors is that they need to be mounted somewhat precisely and carefully. Traditional machining methods make this difficult to accomplish, but 3D printing allows us to easily overcome this challenge. Another concern is external electromagnetic interference, but with somewhat careful circuit board design, we should be able to minimize this issue.

A.4.5.2 Motor Current Sensor

A shunt resistor seems practical due to the simplicity of the design, but careful designing must be done in order to get the noise levels down to a reasonable amount. In addition to this, the power loss when using a shunt resistor could cause the arm to stall before anticipated. When the shunt resistor takes power from the motor, the whole motor curve slides inward, decreasing the maximum power output. Trace resistance would be a good alterna-

tive, but requires calibration after the circuit is constructed.

Instead of these, we decided to use a Hall effect current sensor. Hall effect current sensors are ready-made sensors that give low noise, properly calibrated outputs, are not very expensive, and are easy to integrate into a circuit design. These sensors have extremely small power losses to the motor. The main drawback of these sensors is that they have a low bandwidth, but we are using DC motors so this should not be a problem. Some care will need to be taken when placing these on the circuit, however, since they are sensitive to external magnetic fields.

A.4.5.3 Off-board Communication

SPI and I^2C are mostly used for on-board, controller-to-peripheral communications and therefore are not a good choice for the base to control board communication. RS232 is not a good solution for this problem either because it is a single transmitter and single receiver per line. This leaves RS485 and CAN.

RS485 and CAN are similar in many ways, but with a few key differences that separate them. RS485 is very fast to transmit and simple to implement, but takes a lot of the controller's time to send packets. CAN has the advantage because the controller and transceiver control the transmission independent of the controller so the controller has more free time to process data. Another advantage CAN has over RS485 is the amount of error checking that goes on to ensure proper message transmission. For these reasons, we decided to use CAN to communicate between the base and control boards.

A.4.6 Arm Structure

Arm structure is not something we wanted to define, since the end user is supposed to create their own arms, but there were some basic things we needed to define. The first of these is that every arm must begin with a base and end with an end effector. This is because the central CAN bus must be terminated with resistors at both ends. An alternative to this is to have every piece terminate the CAN bus if it is the last piece in the chain, but this creates unnecessary complexity in each piece. The second constraint placed on arm construction is that sticks cannot connect to other sticks. This is because we didn't want the user to construct an arm with ridiculous length that would be impossible to lift.

A.4.7 Arm Base

A.4.8 End-of-Arm Tooling

A.5 Constraints

Budget will likely be the biggest constraint with this project. The stipend given to students by WPI may not cover all of the costs incurred when constructing this robot, and the rest will be paid out of pocket by the student team. Another large constraint will be access to a 3D printer for prototyping. It will be crucial to start prototyping early to accomplish all of the design goals. Time will also be a major concern, since there are many time consuming aspects to this project.

A.6 Project Goals

The end deliverables for this project will be broken into 5 major categories: Sticks, Joints, End effector, Base, Software application, Code library

A.6.1 Sticks

- Pass power and signal buses
- Strong mechanical connection
- 4-position keyed connection
- 1 input and 1 output connector
- Two kinds of sticks: Straight and Right Angle
- Quick to connect and disconnect
- Connect to joints but not sticks
- Limits exposed wires

A.6.2 Joints

- Receive initialization information and joint angles from base
- Moves to joint angles

- Send position updates back to base
- Pass power and signal buses
- Capable of powering logic without powering motors
- Control board is the same for each joint (address is selected with DIP switch or in firmware)
- 4 position keyed connection
- 1 input and 1 output connector
- Quick to connect and disconnect
- Connects to joints and sticks
- Limits exposed wires

A.6.3 End Effector

- Receives power and signal buses
- 4 position keyed connection
- 1 input connector
- Quick to connect and disconnect
- Connects to joints but not sticks
- Limited exposed wires

A.6.4 Base

- Sends and receives joint angles to/from personal computer
- Receives initialization information from computer, then sends it to all joints on signal bus
- Outputs power and signal buses
- Converts AC wall power to system power bus
- Power supply and arm on/off switch
- Capable of powering logic without powering motors

- Array of indicator LEDs
- 4 position keyed connection
- 1 output connector
- Quick to connect and disconnect
- Connects to joints and sticks
- Limits exposed wires

A.6.5 Software Application

- Sends configuration information to Code Library
- Sends individual joint angles or pose commands to robot through Code Library
- GUI to adjust current arm configuration parameters
- Record and play back sequence of poses
- Acts as a front-end for code library
- Stretch goal: 3D model of arm moving in real-time

A.6.6 Code Library

- Written so that it can interface with multiple languages
- Receive configuration information from user, selects control constants, sends to base
- Able to control the robot: Receive joint status, send joint angles
- Calculate joint angles using kinematics

A.7 Kit Components

- 3x Twist Joints (axis of rotation parallel to input Stick)
- 6x Rotational Joints (axis of rotation perpendicular to input Stick)
- 3x 75mm Straight Sticks
- 3x 150mm Straight Sticks

- 3x 225mm Straight Sticks
- 3x 75mm Right Angle Sticks
- 3x 150mm Right Angle Sticks
- 3x 225mm Right Angle Sticks
- 1x Claw Gripper End-of-Arm Tool
- 1x Hook End-of-Arm Tool
- 1x Capacitive Stylus/Pointer End-of-Arm Tool
- 1x Arm Base
- Stretch Goals:
  - Prismatic Joint(s)
  - 1x Universal Gripper End-of-Arm Tool

Appendix B Solidworks Drawings

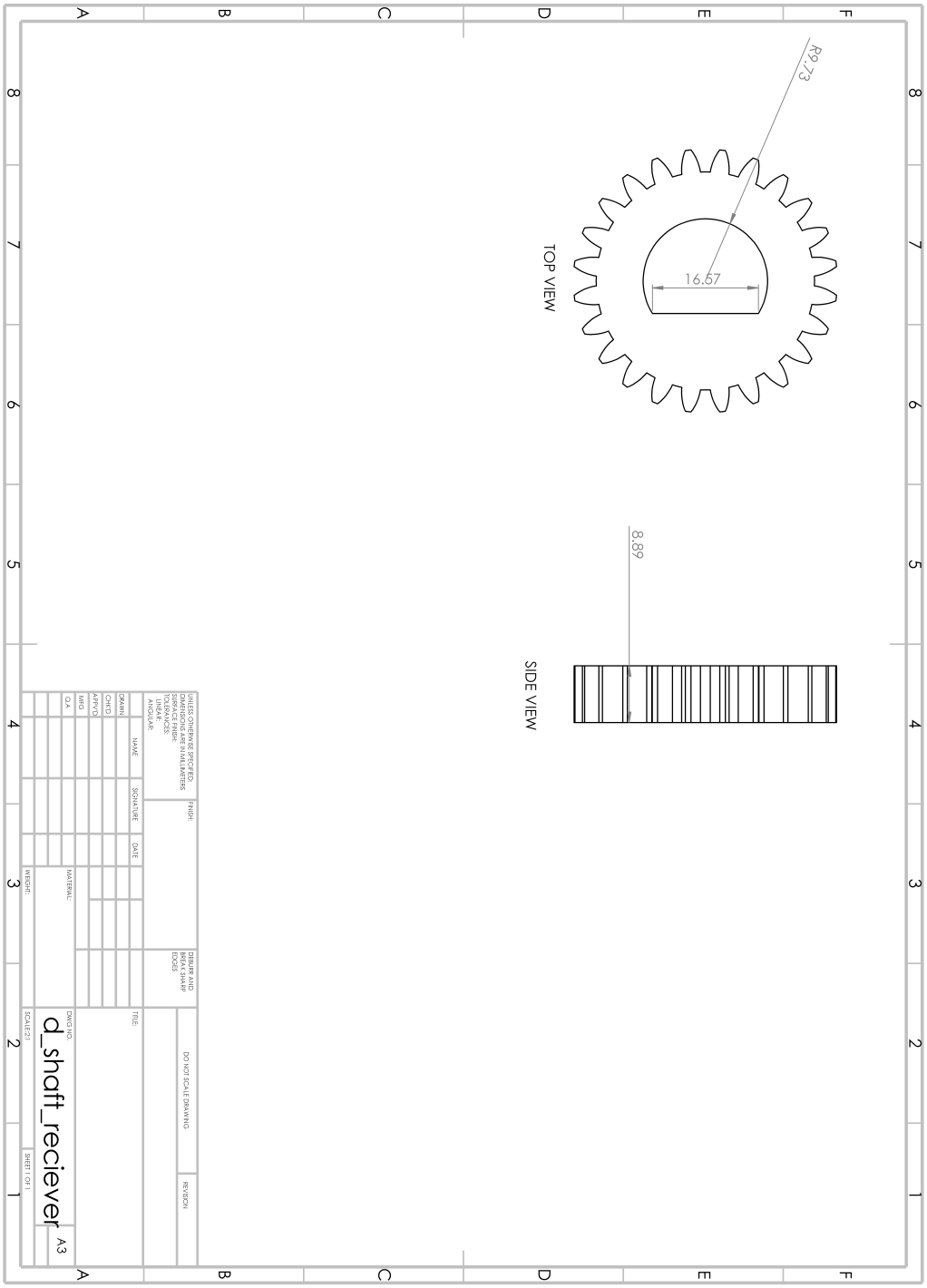

Figure B.1: D shaft receiver

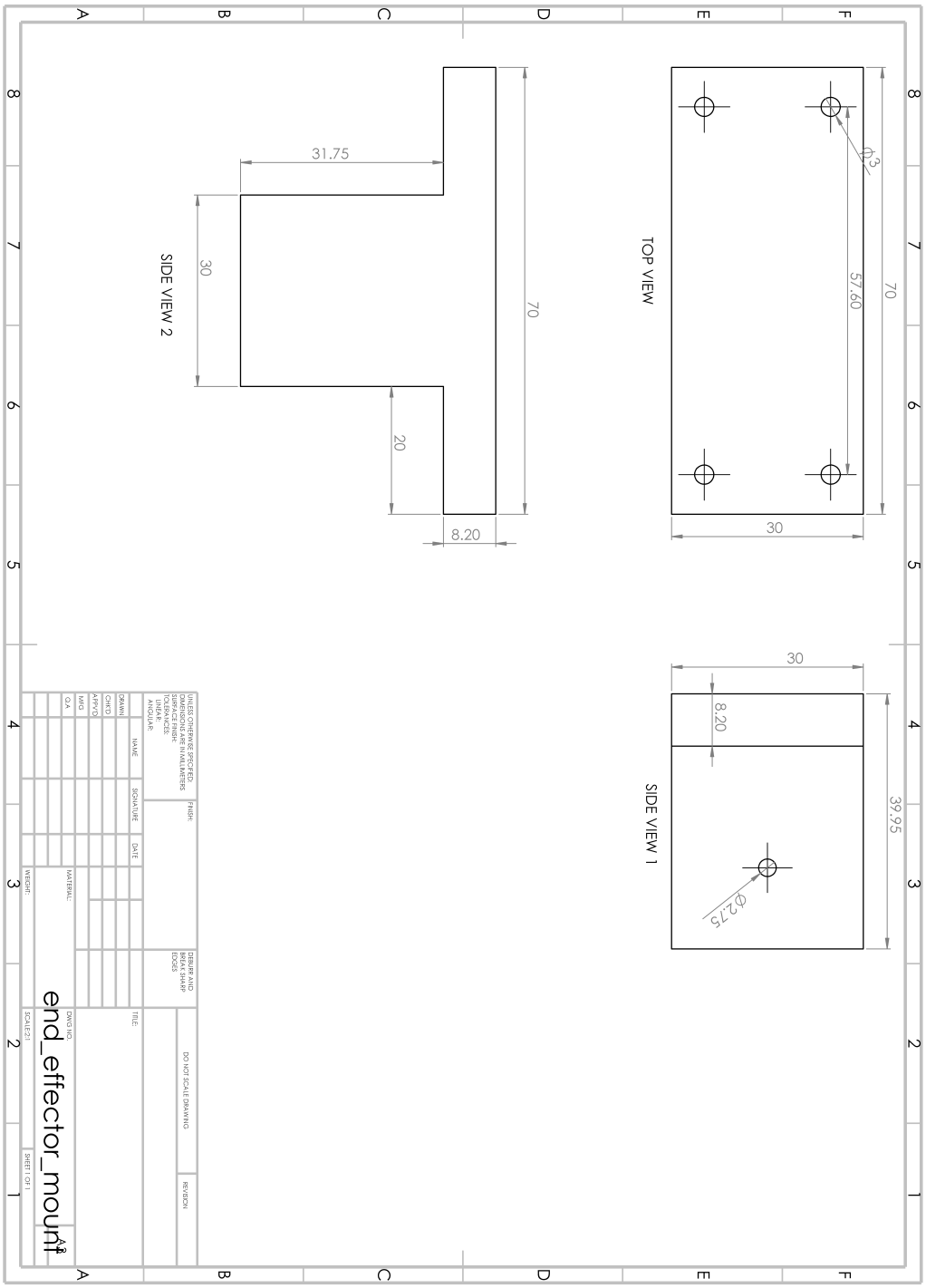

Figure B.2: End effector mount

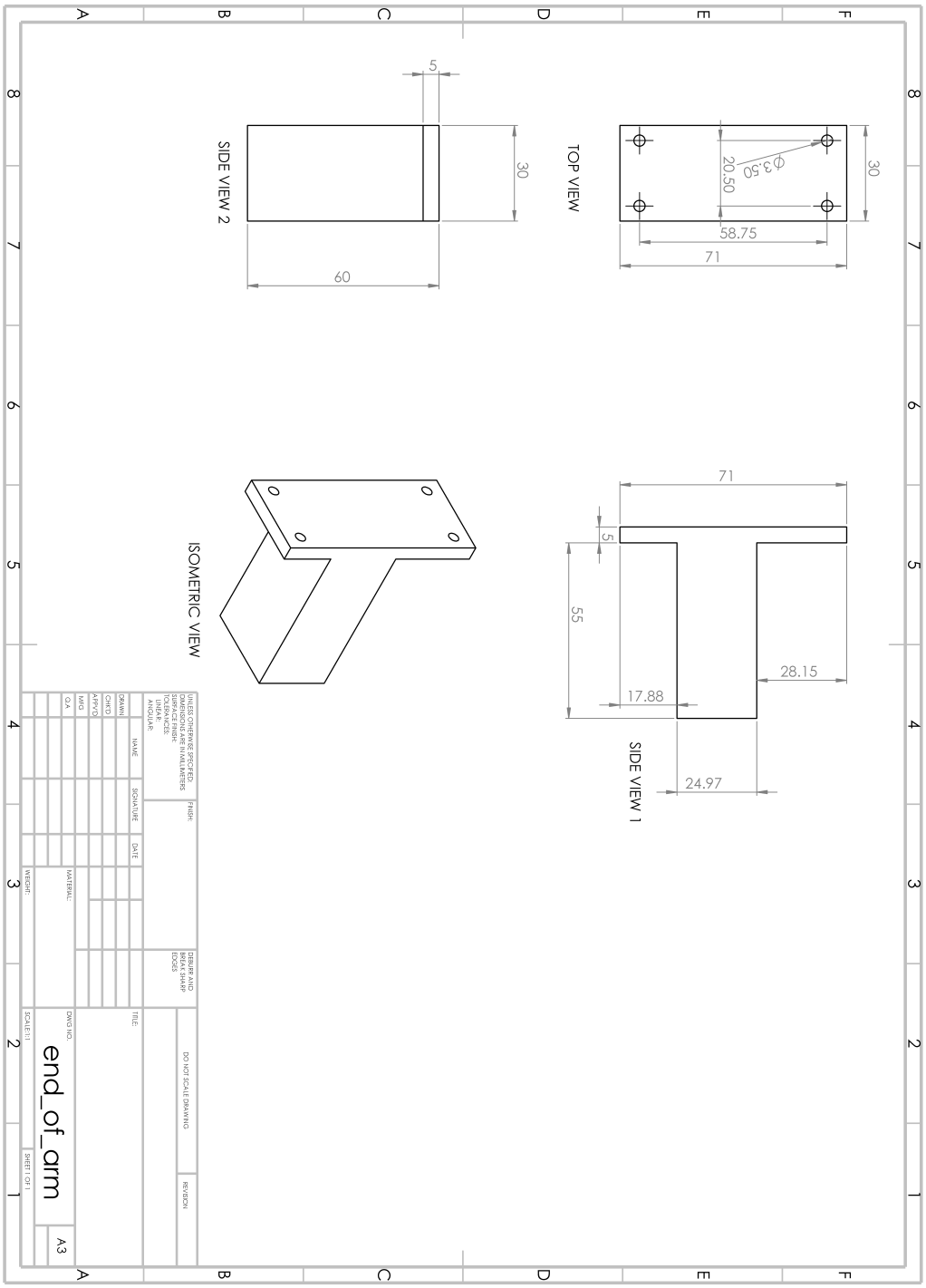

Figure B.3: End of Arm

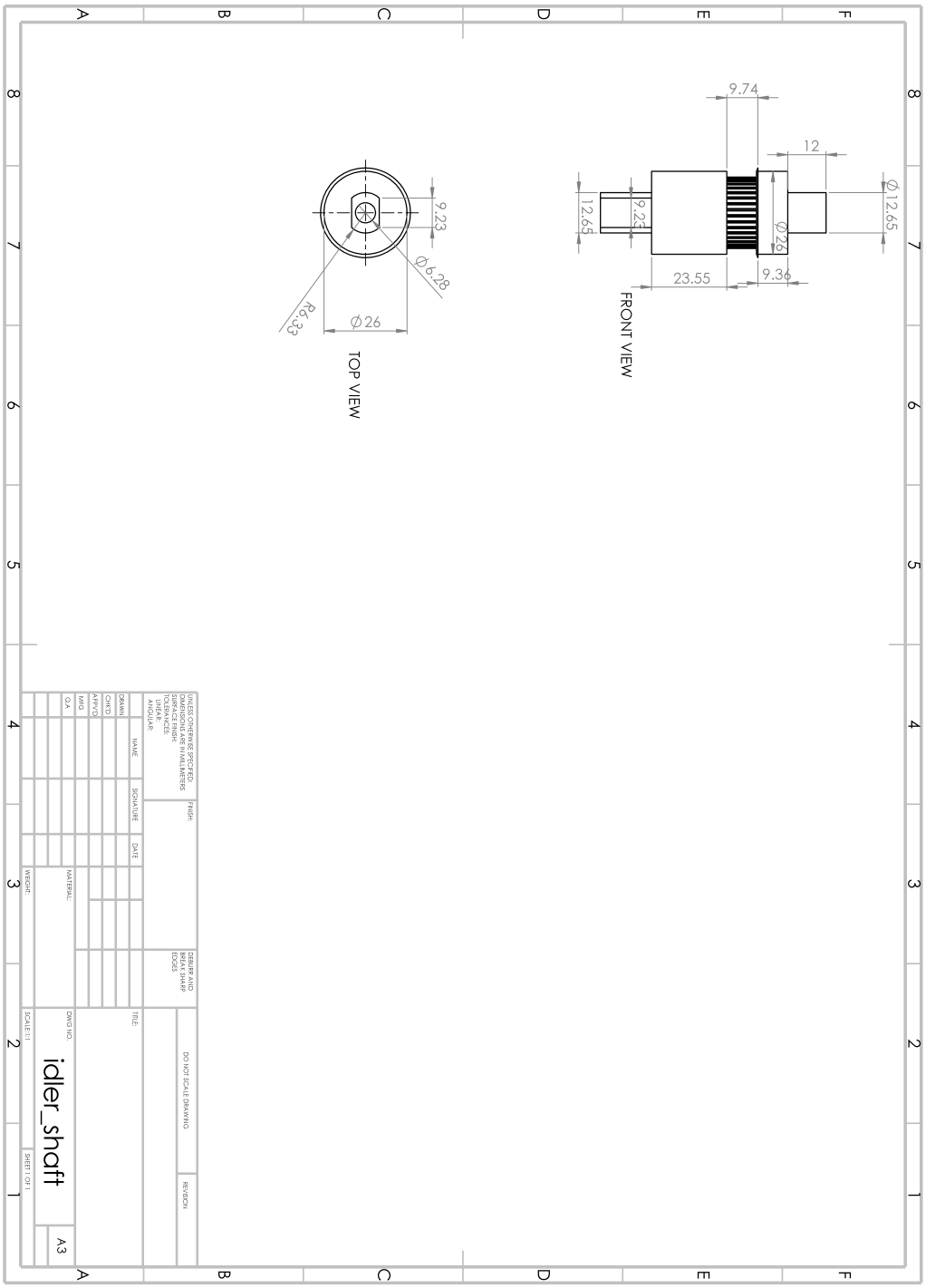

Figure B.4: Idler shaft

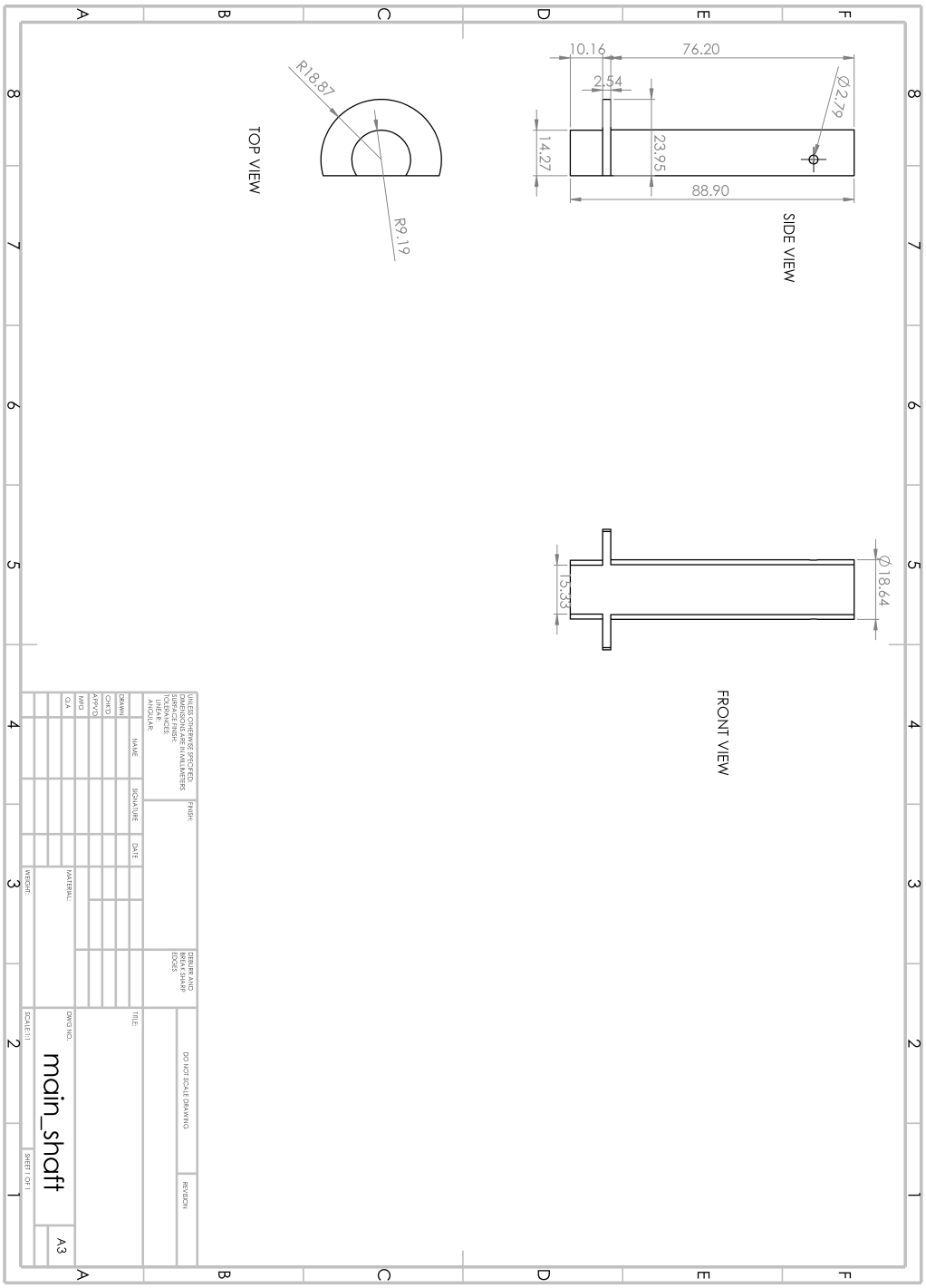

Figure B.5: Main shaft

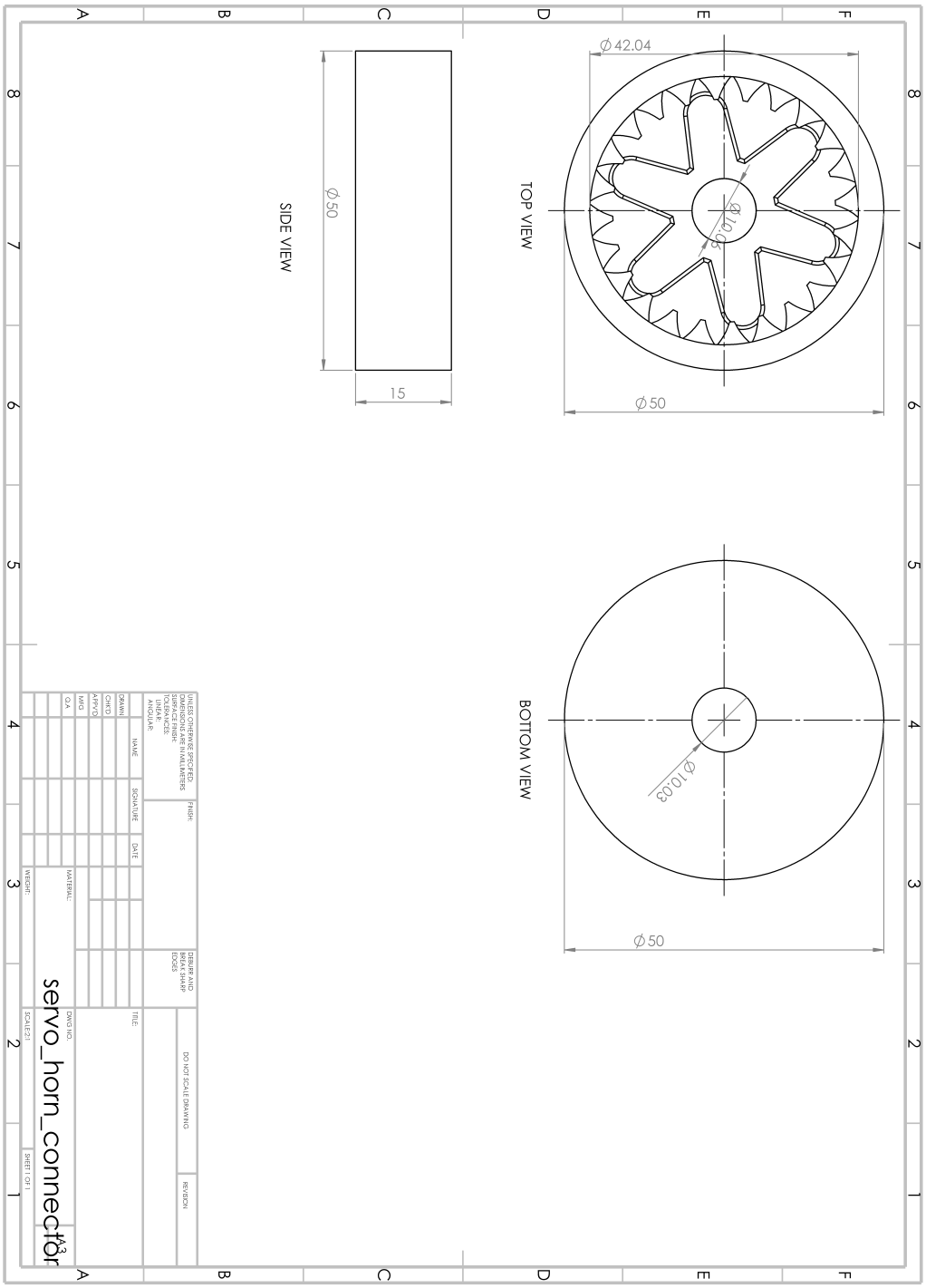

Figure B.6: Servo horn connector

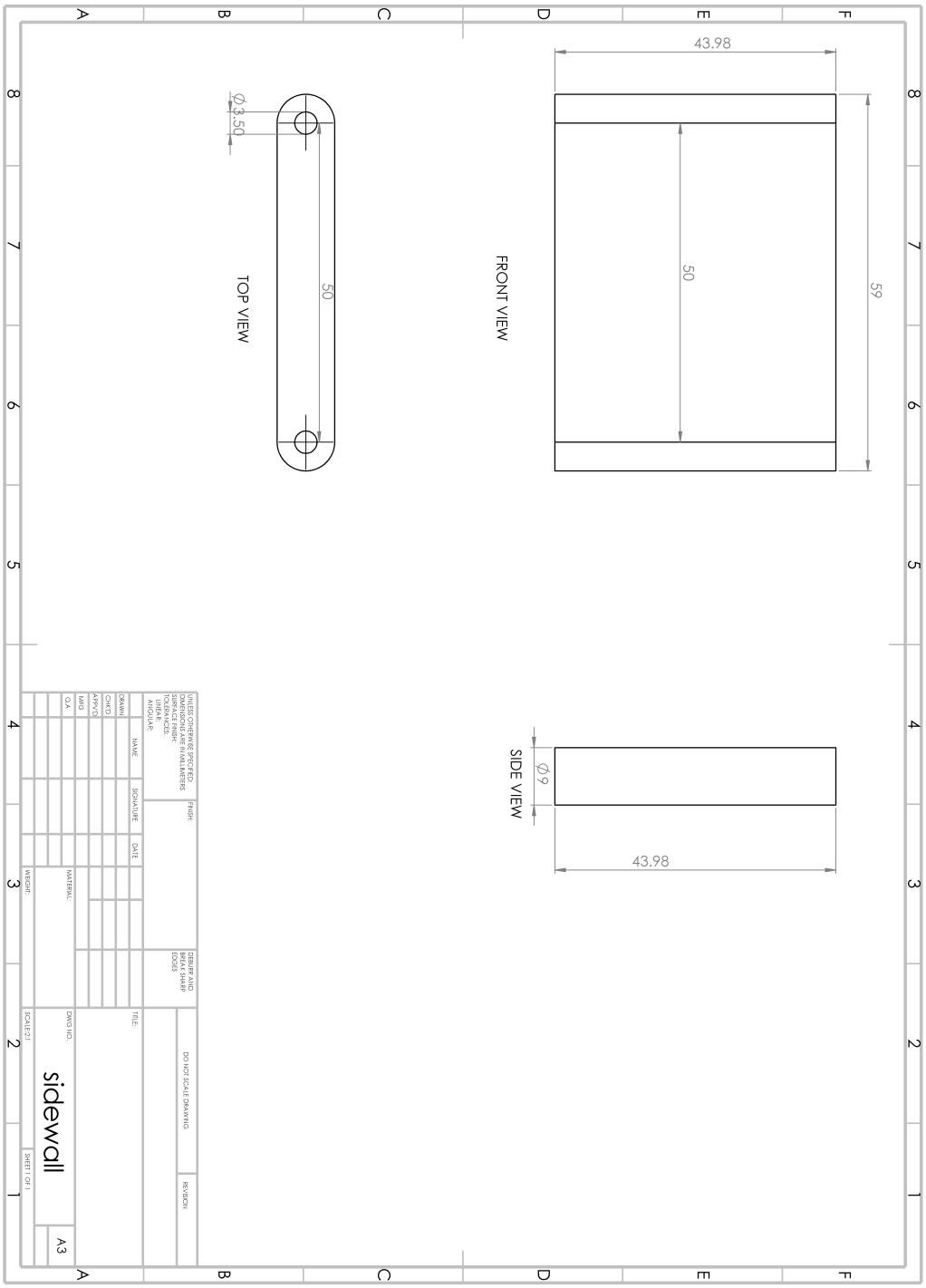

Figure B.7: Sidewall

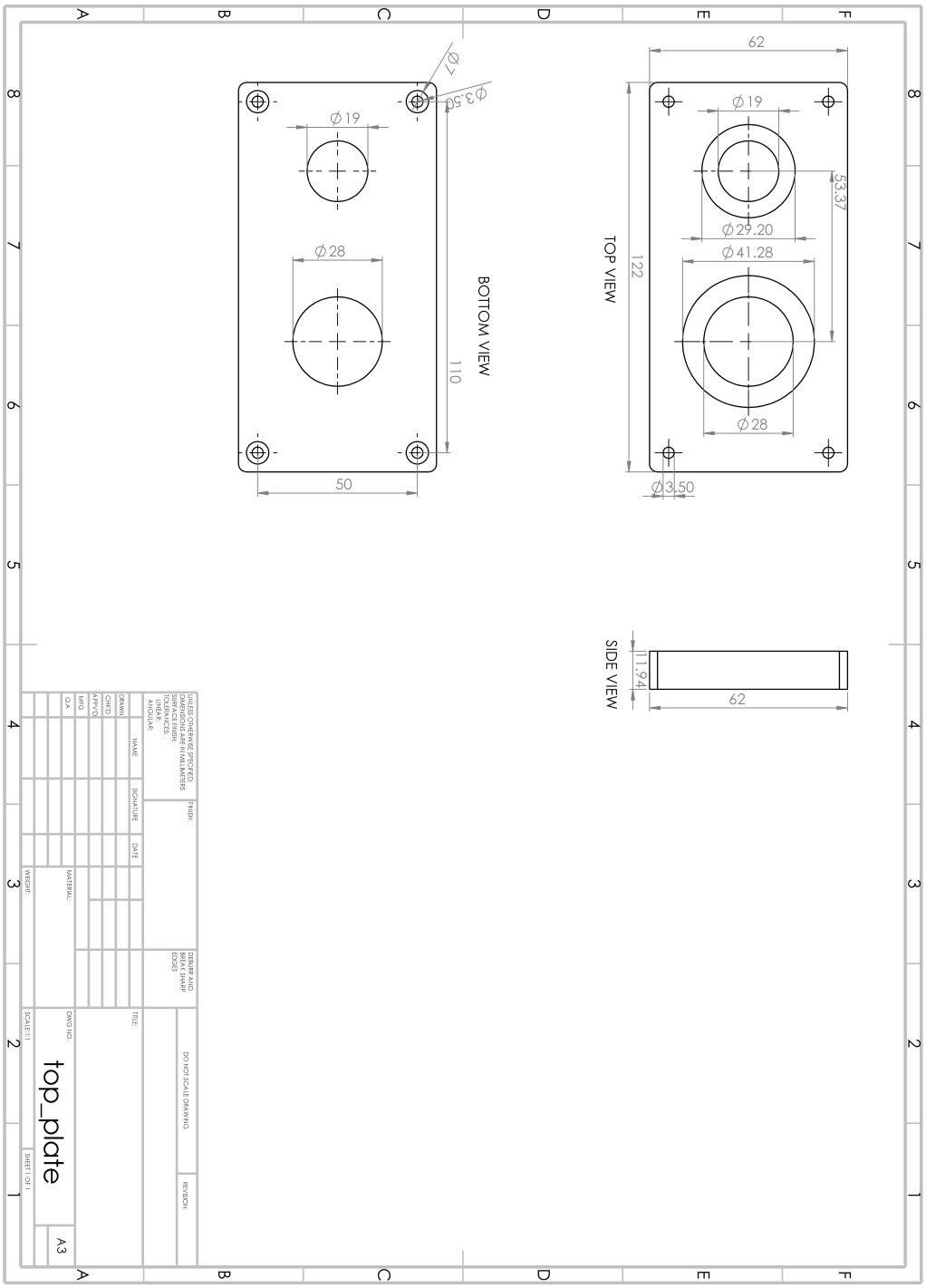

Figure B.8: Top plate

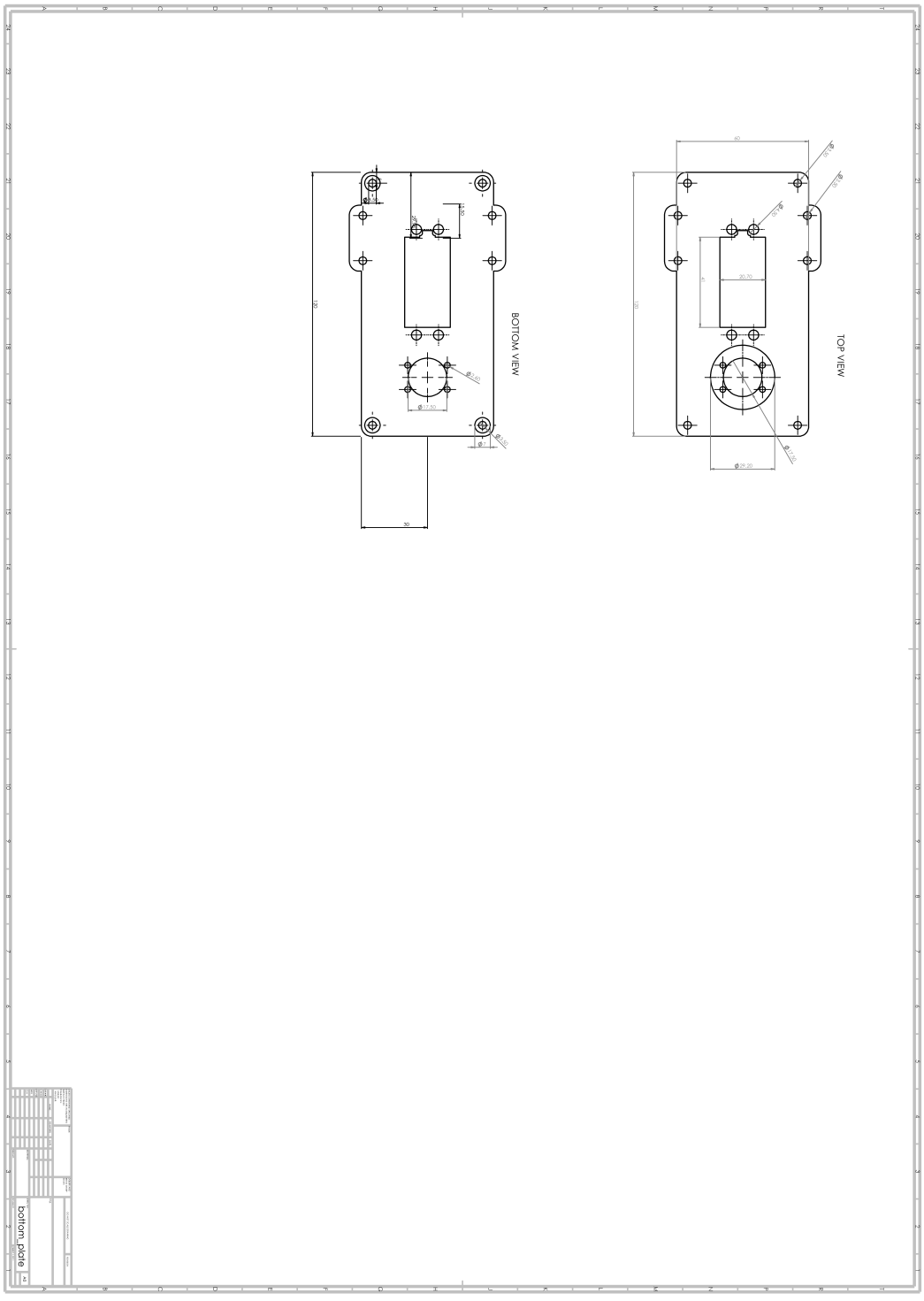

Figure B.9: Bottom plate

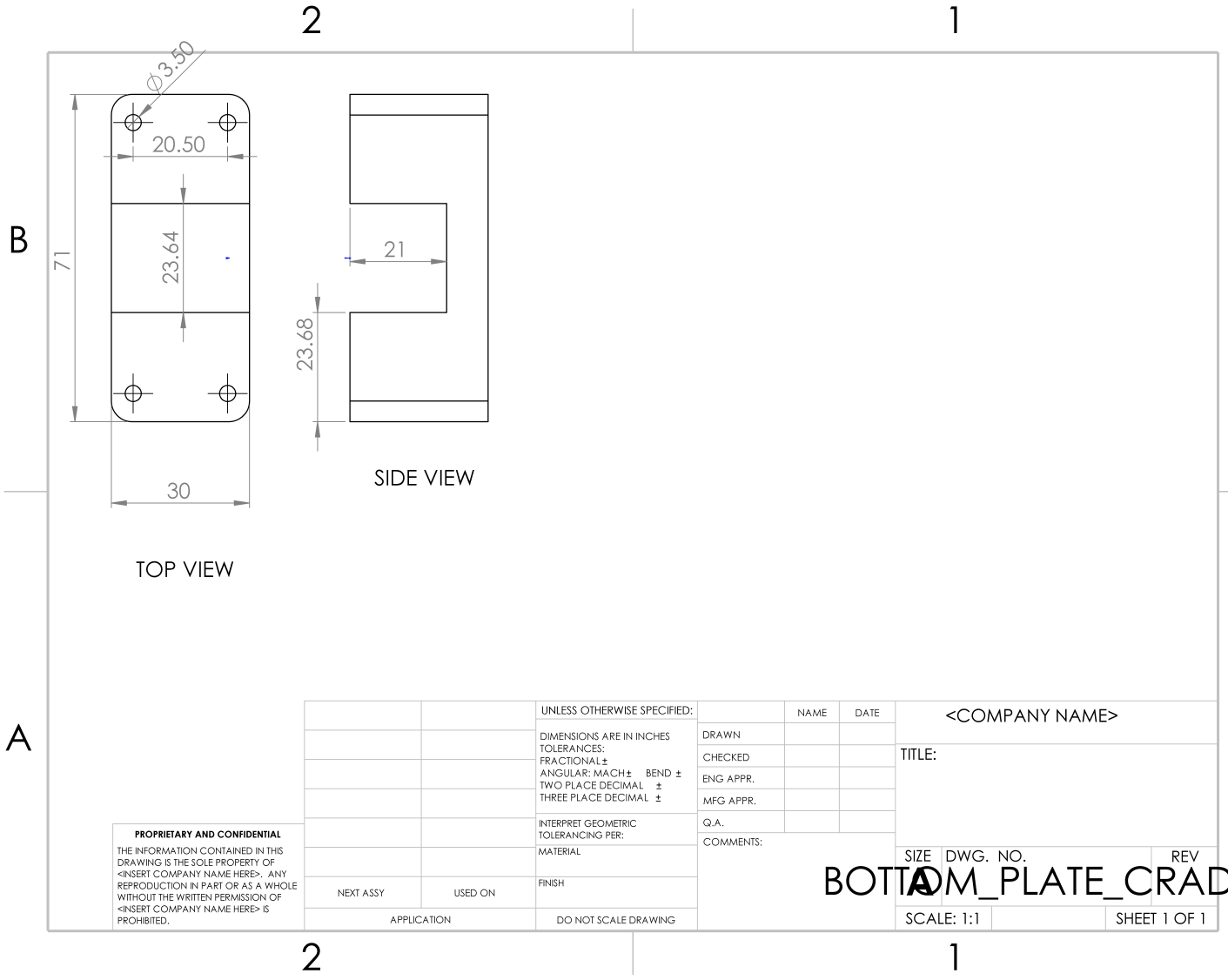

Figure B.10: Bottom plate cradle

Appendix C Motor Driver

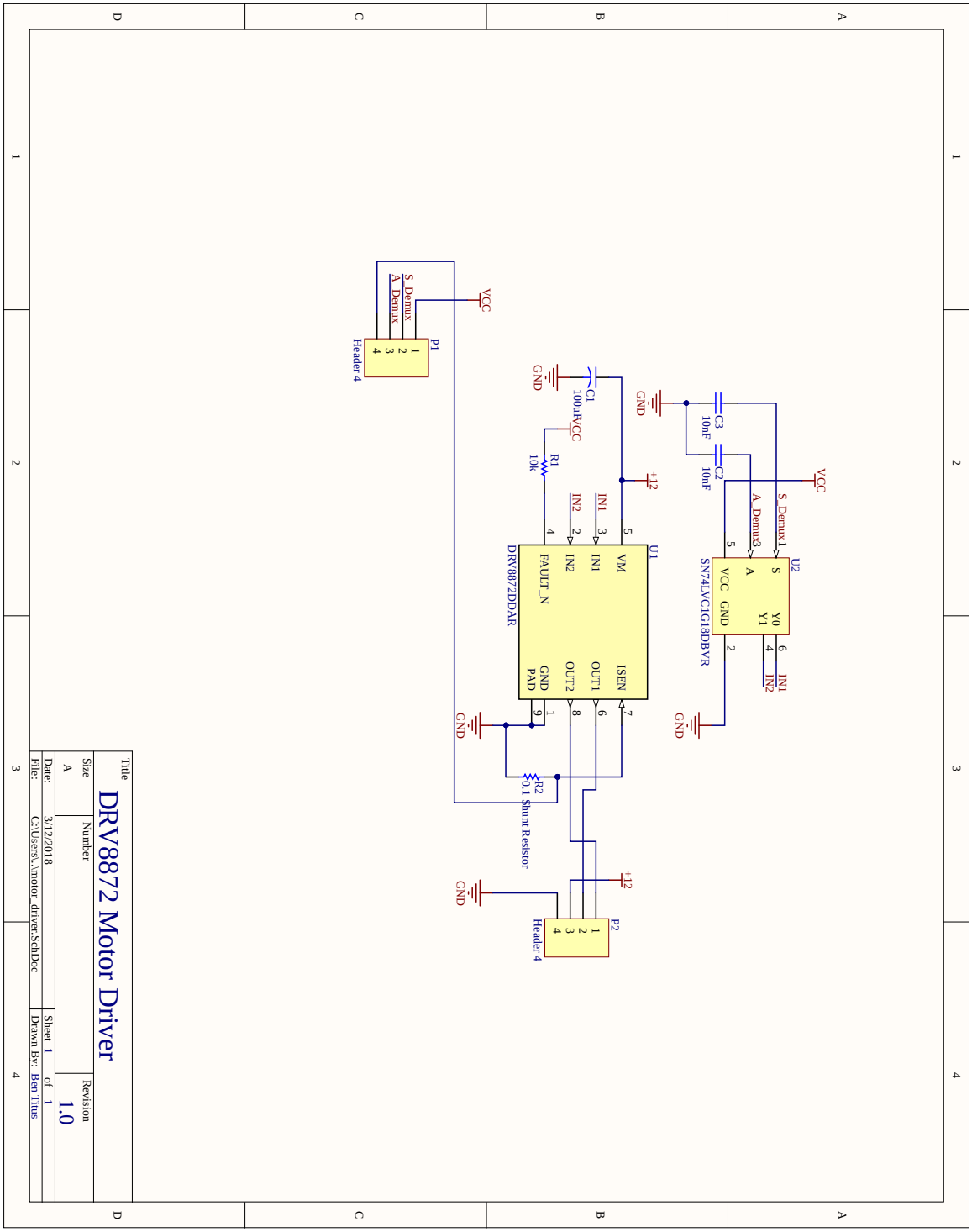

Figure C.1: Circuit diagram for motor driver PCB

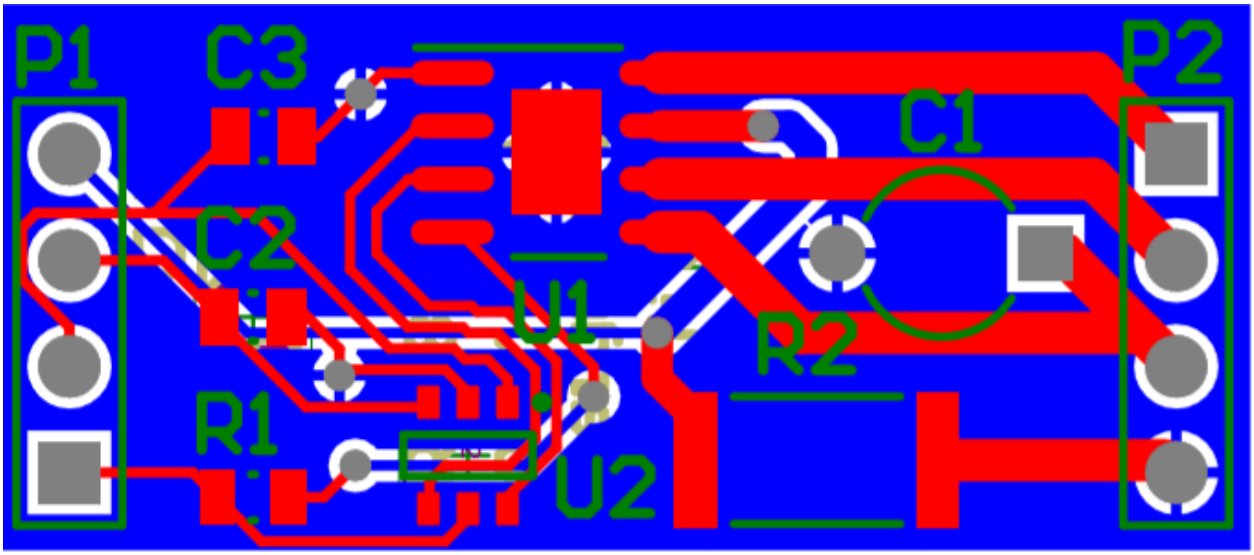

Figure C.2: PCB composite for the motor driver PCB

Comment	Description	Designator	Footprint	LibRef	Quantity
100uF	Capacitor	C1	CAPR5-4X5	Cap2	1
CAP0805	0805 (2012 Metric) Chip Capacitor	C2, C3	CAPC0805(2012)145_L	CMP-1590-00003-1	2
Header 4	Header, 4-Pin	P1, P2	HDR1X4	Header 4	2
RES0805	0805 (2012 Metric) Chip Resistor	R1	RESC0805(2012)_L	CMP-1591-00002-1	1
RES2512	2512 (6432 Metric) Chip Resistor	R2	RESC2512(6432)_L	CMP-1591-00007-1	1
DRV8872DDAR	Imported	U1	DDA0008H_N	DRV8872DDAR	1
SN74LVC1G18DBVR	One of Two Noninverting Demultiplexer with 3-State Deselected Output, DBV0006A, LARGE T&R	U2	DBV0006A_L	CMP-0859-00184-3	1

Figure C.3: Bill of materials for the motor driver PCB

Appendix D Motor Driver with Current Sensor

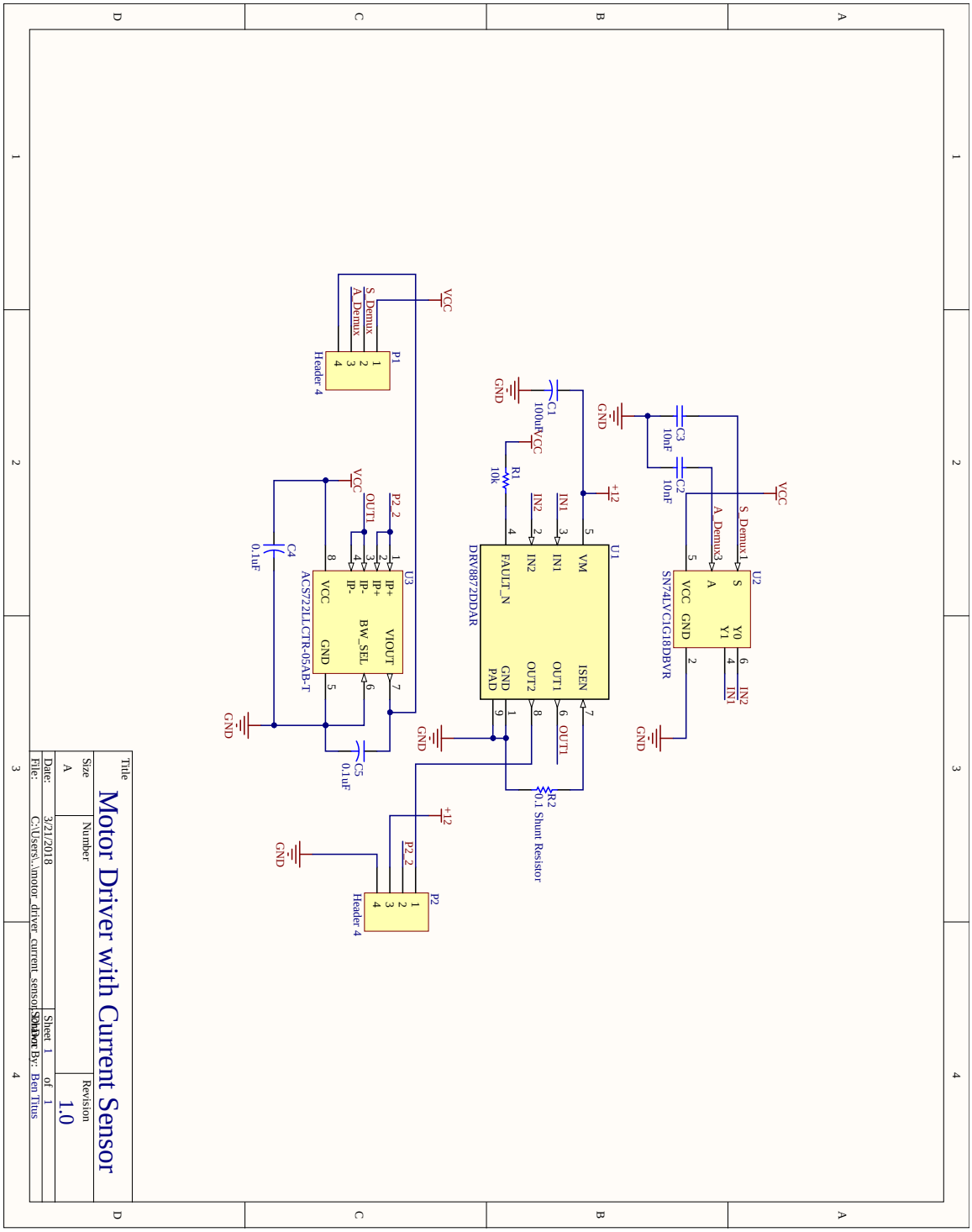

Figure D.1: Circuit diagram for motor driver with current sensor PCB

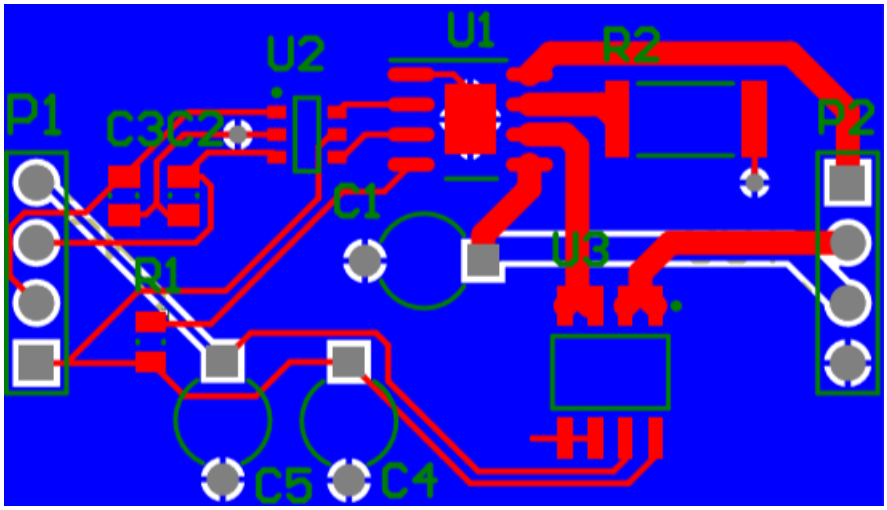

Figure D.2: PCB composite for the motor driver with current sensor PCB

Comment	Description	Designator	Footprint	LibRef	Quantity
100uF	100uF Filter Capacitor	C1	CAPR5-4X5	Cap2	1
CAP0805	0805 (2012 Metric) Chip Capacitor	C2, C3	CAPC0805(2012)145_L	CMP-1590-00003-1	2
0.1uF	Capacitor	C4, C5	CAPR5-4X5	Cap2	2
Header 4	Header, 4-Pin	P1, P2	HDR1X4	Header 4	2
RES0805	0805 (2012 Metric) Chip Resistor	R1	RESC0805(2012)_L	CMP-1591-00002-1	1
RES2512	2512 (6432 Metric) Chip Resistor	R2	RESC2512(6432)_L	CMP-1591-00007-1	1
DRV8872DDAR	Imported	U1	DDA0008H_N	DRV8872DDAR	1
SN74LVC1G18DBVR	One of Two Noninverting Demultiplexer with 3-State Deselected Output, DBV0006A, LARGE T&R	U2	DBV0006A_L	CMP-0859-00184-3	1
ACS722LLCTR-05AB-T	High Accuracy, Galvanically Isolated Current Sensor IC, 3 to 3.6 V, -5 to 5 A IP, -40 to 150 degC, 8-Pin SOIC (LC), RoHS, Tape and Reel	U3	ALEG-LC-8_V	CMP-1557-00006-1	1

Figure D.3: Bill of materials for the motor driver with current sensor PCB

Appendix E Load Cell Amplifier

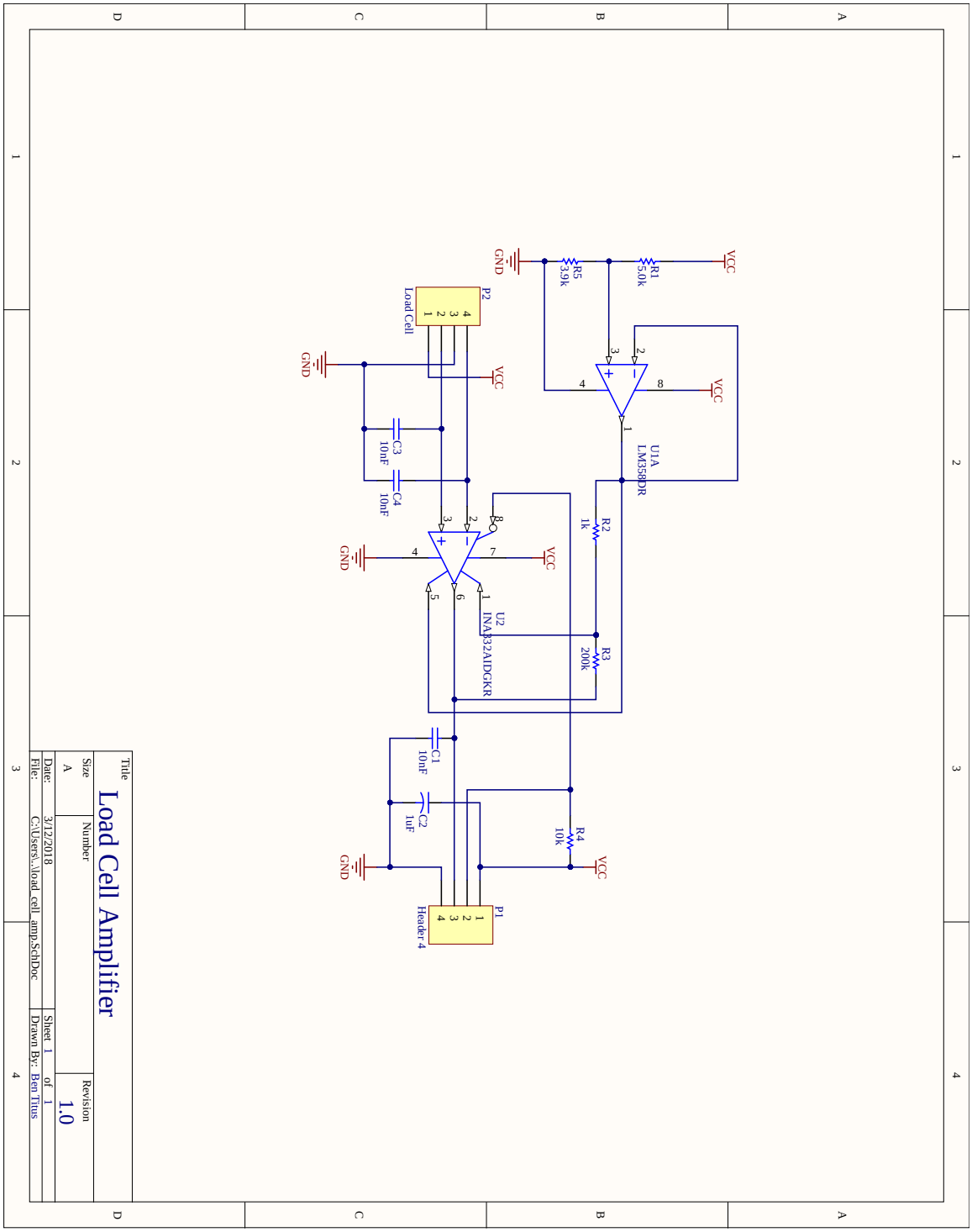

Figure E.1: Circuit diagram for load cell amplifier PCB

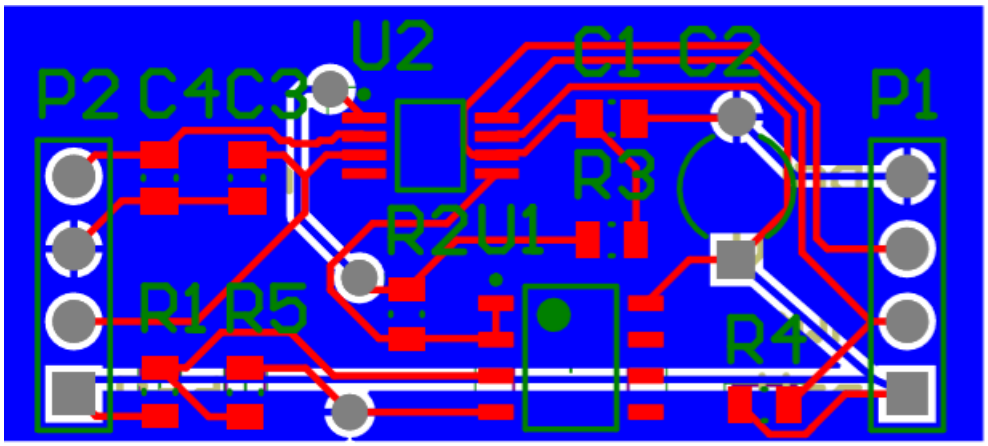

Figure E.2: PCB composite for the load cell amplifier PCB

Comment	Description	Designator	Footprint	LibRef	Quantity
CAP0805	0805 (2012 Metric) Chip Capacitor	C1, C3, C4	CAPC0805(2012)145_L	CMP-1590-00003-1	3
1uF	Capacitor	C2	CAPR5-4X5	Cap2	1
Header 4	Header, 4-Pin	P1	HDR1X4	Header 4	1
Load Cell	Header, 4-Pin	P2	HDR1X4	Header 4	1
RES0805	0805 (2012 Metric) Chip Resistor	R1, R2, R3, R4, R5	RESC0805(2012)_L	CMP-1591-00002-1	5
LM358DR	Dual Operational Amplifier, 3 to 32 V, 0 to 70 degC, 8-Pin SOIC (D), Green (RoHS & no Sb/Br), Tape and Reel	U1	D0008A_L	CMP-1685-00009-1	1
INA332AIDGKR	Low-Power, Single Supply, CMOS, Low Cost, Instrumentation Amplifier, -55 to 125 degC, 8-pin SOP (DGK8), Green (RoHS & no Sb/Br)	U2	DGK0008A_M	CMP-0944-00077-2	1

Figure E.3: Bill of materials for the load cell amplifier PCB

Appendix F CAN Transceiver

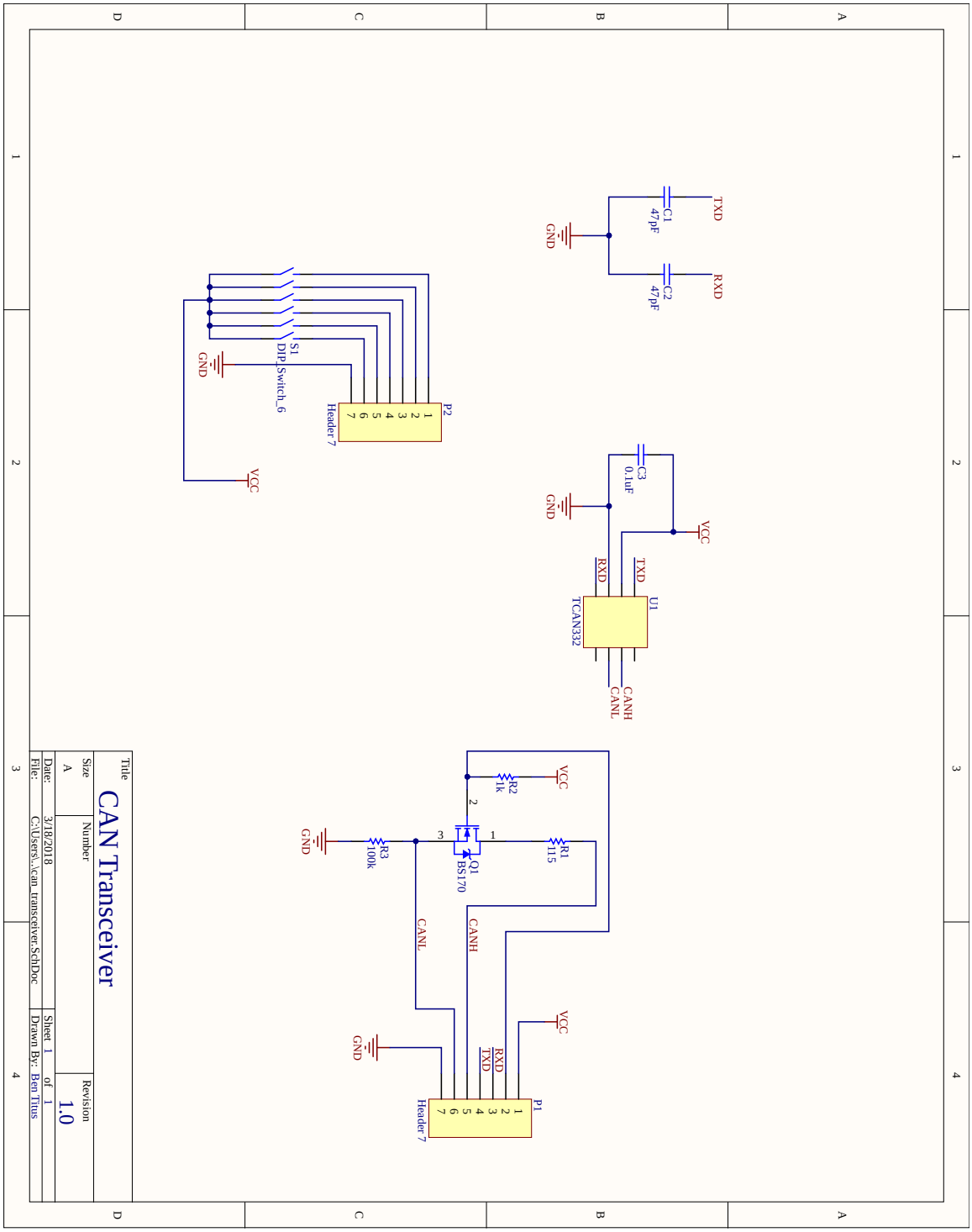

Figure F.1: Circuit diagram for CAN transceiver PCB

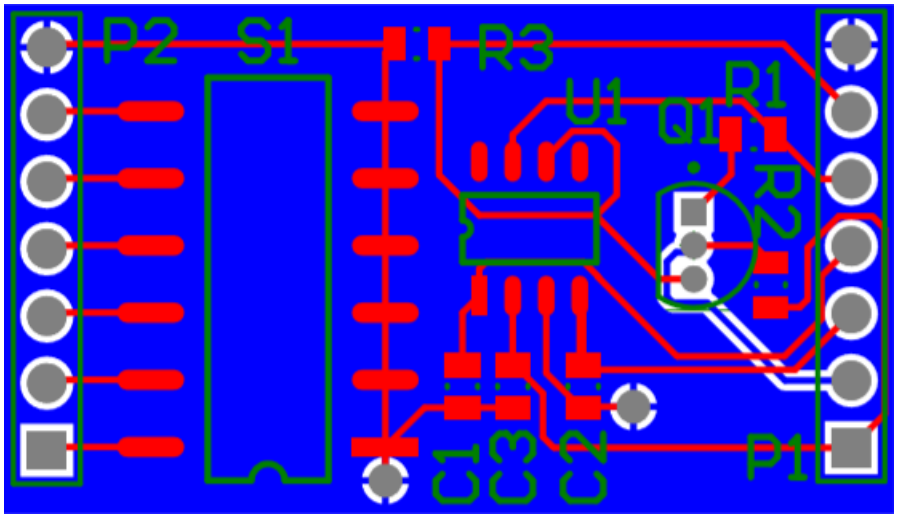

Figure F.2: PCB composite for the CAN transceiver PCB

Comment	Description	Designator	Footprint	LibRef	Quantity
CAP0805	0805 (2012 Metric) Chip Capacitor	C1, C2, C3	CAPC0805(2012)145_L	CMP-1590-00003-1	3
Header 7	Header, 7-Pin	P1, P2	HDR1X7	Header 7	2
BS170	Small Signal MOSFET, 500 mA, 60 V, N-Channel, 3-Pin TO-92, Bulk Box	Q1	ONSC-TO-92-3-29-11	BS170	1
RES0805	0805 (2012 Metric) Chip Resistor	R1, R2, R3	RESC0805(2012)_L	CMP-1591-00002-1	3
DIP_Switch_6	6 pin DIP switch	S1	SOP12	DIP_Switch_6	1
TCAN332	3.3V CAN Transceiver	U1	SOP8	TCAN332	1

Figure F.3: Bill of materials for the CAN transceiver PCB

Appendix G Joint Board Boosterpack

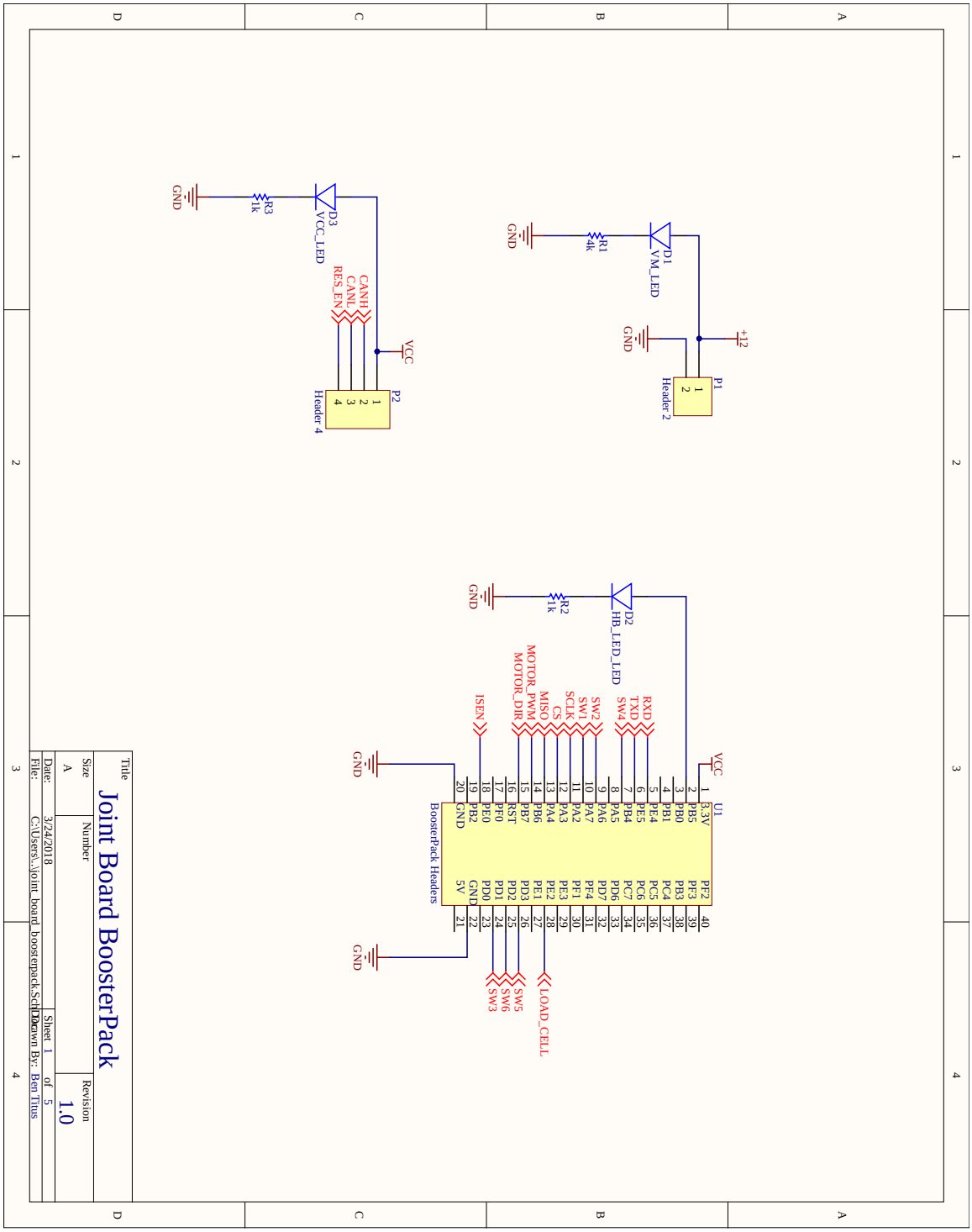

Figure G.1: Circuit diagram for joint board Boosterpack PCB

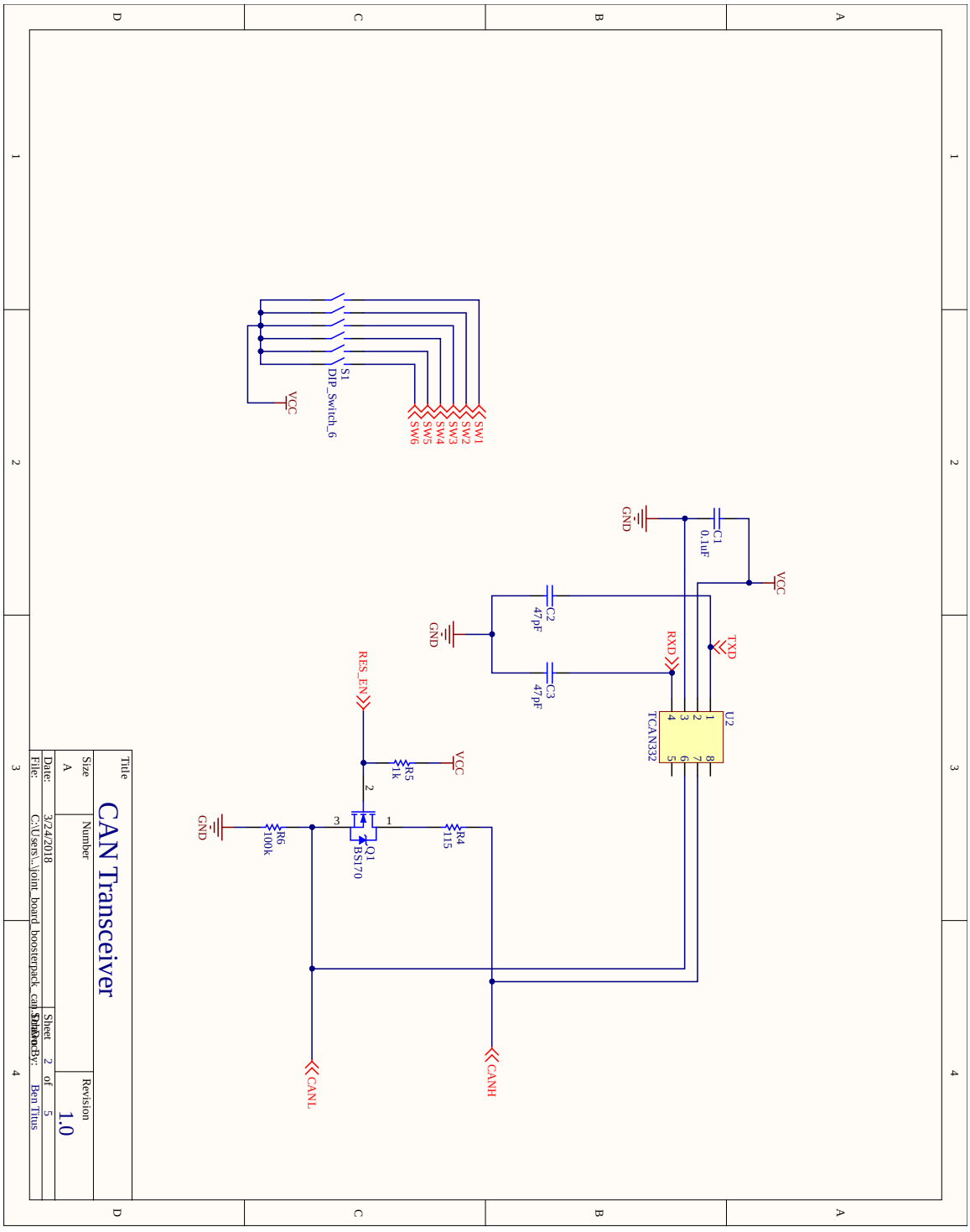

Figure G.2: Circuit diagram for joint board Boosterpack PCB

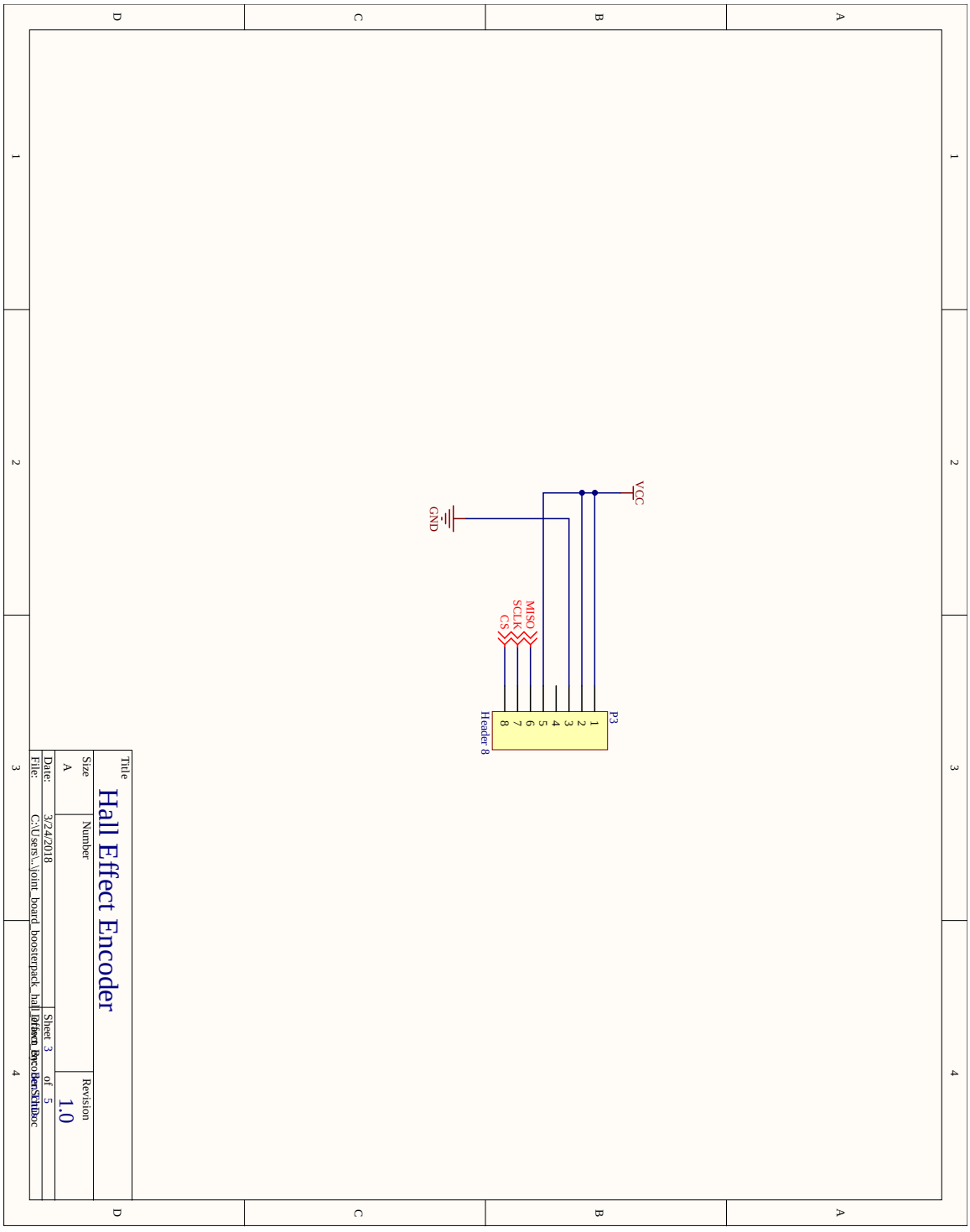

Figure G.3: Circuit diagram for joint board Boosterpack PCB

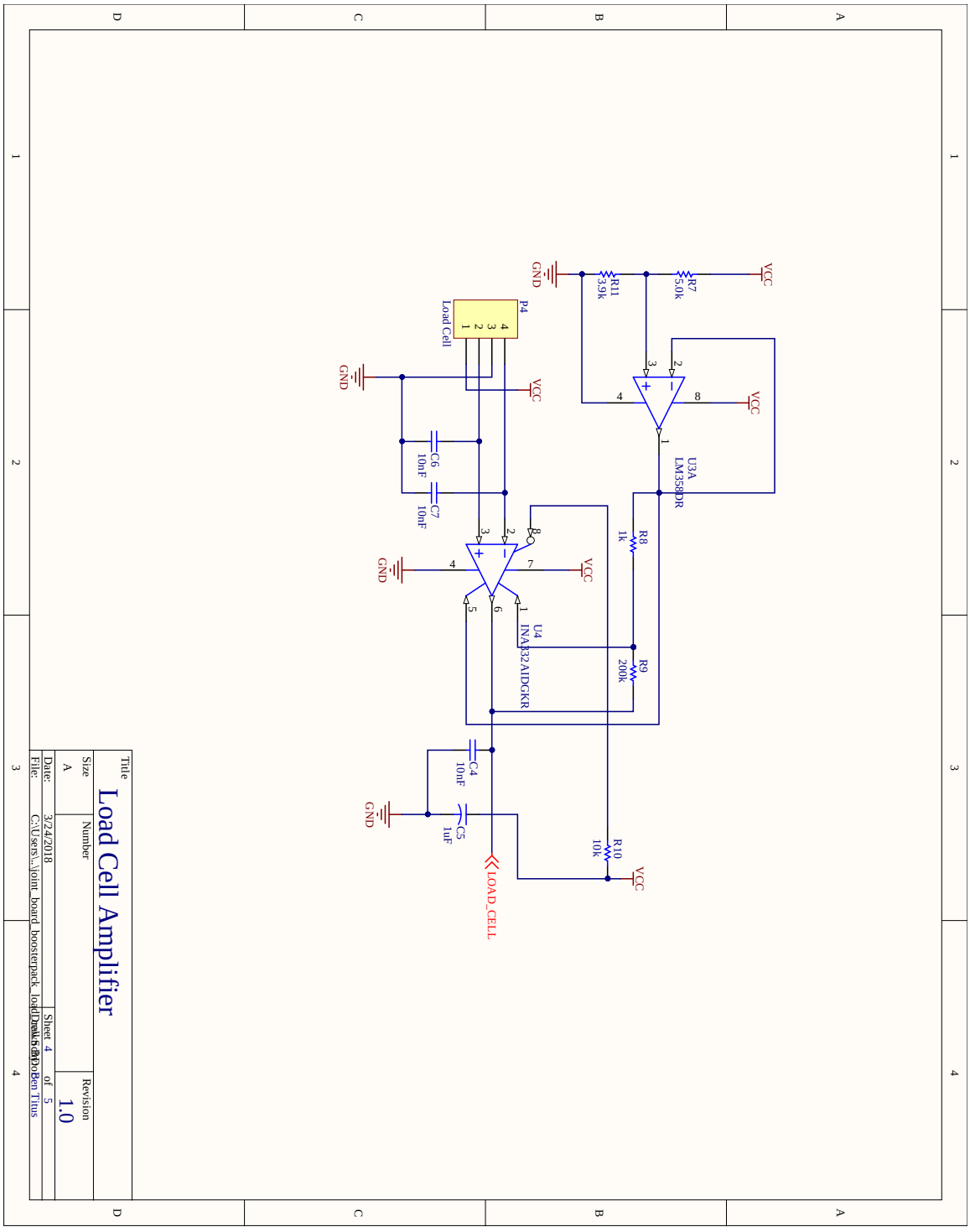

Figure G.4: Circuit diagram for joint board Boosterpack PCB

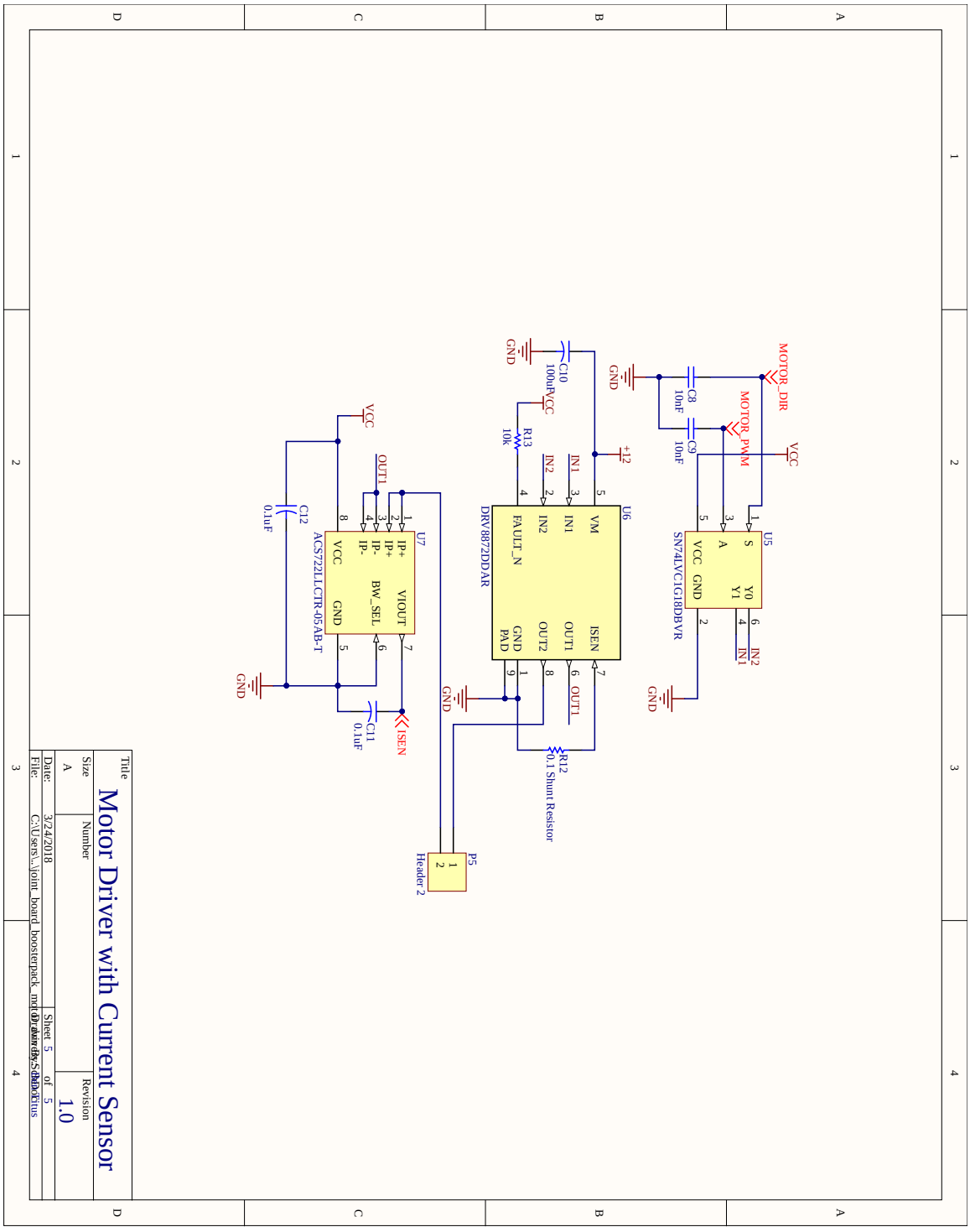

Figure G.5: Circuit diagram for joint board Boosterpack PCB

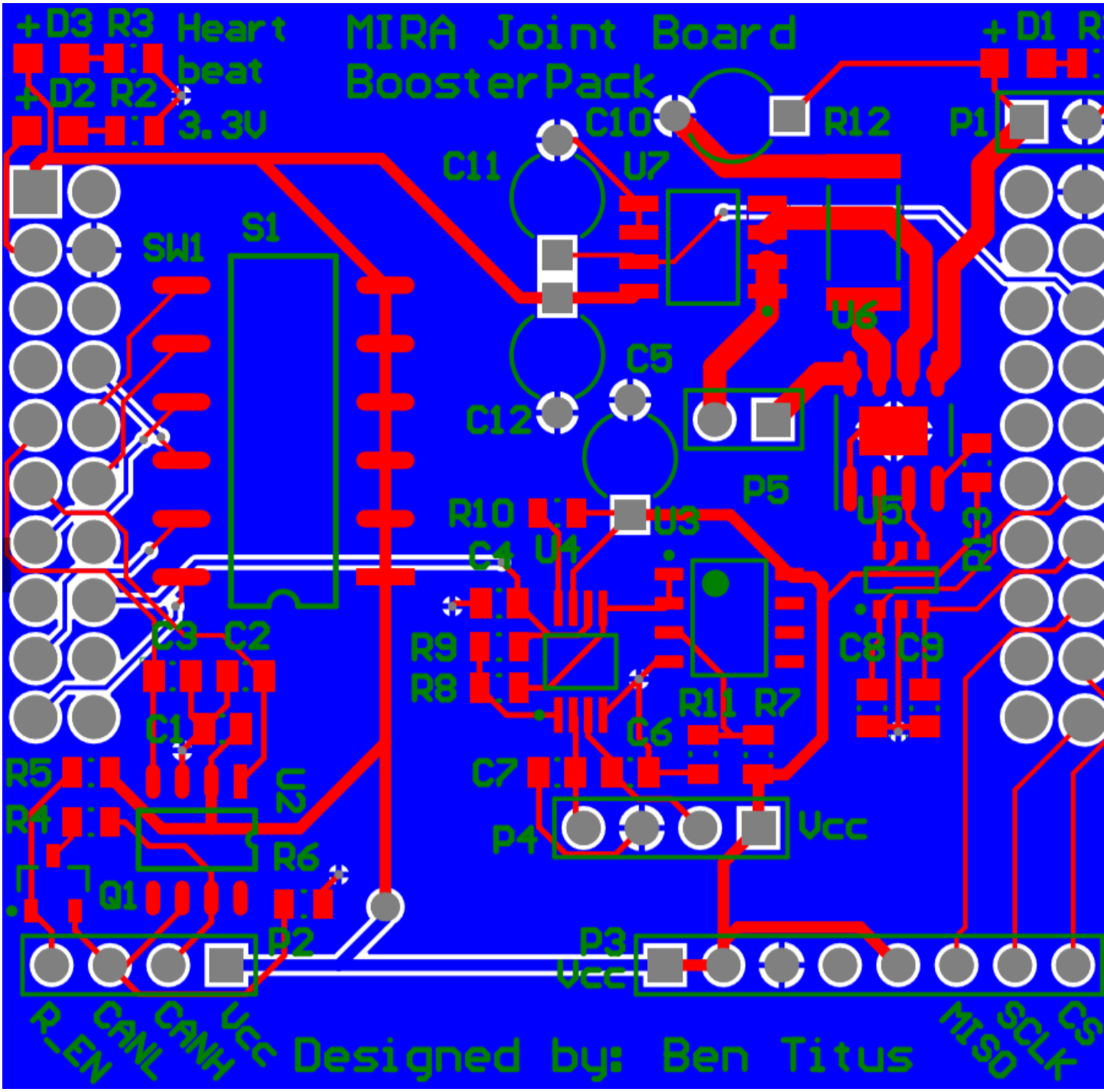

Figure G.6: PCB composite for the joint board Boosterpack PCB

Comment	Description	Designator	Footprint	LibRef	Quantity
CAP0805	0805 (2012 Metric) Chip Capacitor	C1, C2, C3, C4, C6, C7, C8, C9	CAPC0805(2012)145_L	CMP-1590-00003-1	8
1uF	Capacitor	C5	CAPR5-4X5	Cap2	1
100uF	100uF Filter Capacitor	C10	CAPR5-4X5	Cap2	1
0.1uF	Capacitor	C11, C12	CAPR5-4X5	Cap2	2
VM_LED		D1	0805 Diode	SMD_LED	1
HB_LED_LED		D2	0805 Diode	SMD_LED	1
VCC_LED		D3	0805 Diode	SMD_LED	1
Header 2	Header, 2-Pin	P1, P5	HDR1X2	Header 2	2
Header 4	Header, 4-Pin	P2	HDR1X4	Header 4	1
Header 8	Header, 8-Pin	P3	HDR1X8	Header 8	1
Load Cell	Header, 4-Pin	P4	HDR1X4	Header 4	1
BS170	Small Signal MOSFET, 500 mA, 60 V, N-Channel, 3-Pin TO-92, Bulk Box	Q1	SOT23-3	BS170	1
RES0805	0805 (2012 Metric) Chip Resistor	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R13	RESC0805(2012)_L	CMP-1591-00002-1	12
RES2512	2512 (6432 Metric) Chip Resistor	R12	RESC2512(6432)_L	CMP-1591-00007-1	1
DIP_Switch_6	6 pin DIP switch	S1	SOP12	DIP_Switch_6	1
BoosterPack Headers		U1	boosterpack_headers	BoosterPack Headers	1
TCAN332	3.3V CAN Transceiver	U2	SOP8	TCAN332	1
LM358DR	Dual Operational Amplifier, 3 to 32 V, 0 to 70 degC, 8-Pin SOIC (D), Green (RoHS & no Sb/Br), Tape and Reel	U3	D0008A_L	CMP-1685-00009-1	1
INA332AIDGKR	Low-Power, Single Supply, CMOS, Low Cost, Instrumentation Amplifier, -55 to 125 degC, 8-pin SOP (DGK8), Green (RoHS & no Sb/Br)	U4	DGK0008A_M	CMP-0944-00077-2	1
SN74LVC1G18DBVR	One of Two Noninverting Demultiplexer with 3-State Deselected Output, DBV0006A, LARGE T&R	U5	DBV0006A_L	CMP-0859-00184-3	1
DRV8872DDAR	Imported	U6	DDA0008H_N	DRV8872DDAR	1
ACS722LLCTR-05AB-T	High Accuracy, Galvanically Isolated Current Sensor IC, 3 to 3.6 V, -5 to 5 A IP, -40 to 150 degC, 8-Pin SOIC (LC), RoHS, Tape and Reel	U7	ALEG-LC-8_V	CMP-1557-00006-1	1

Figure G.7: Bill of materials for the joint board Boosterpack PCB

Appendix H TM4C123GH6PM Dev Board

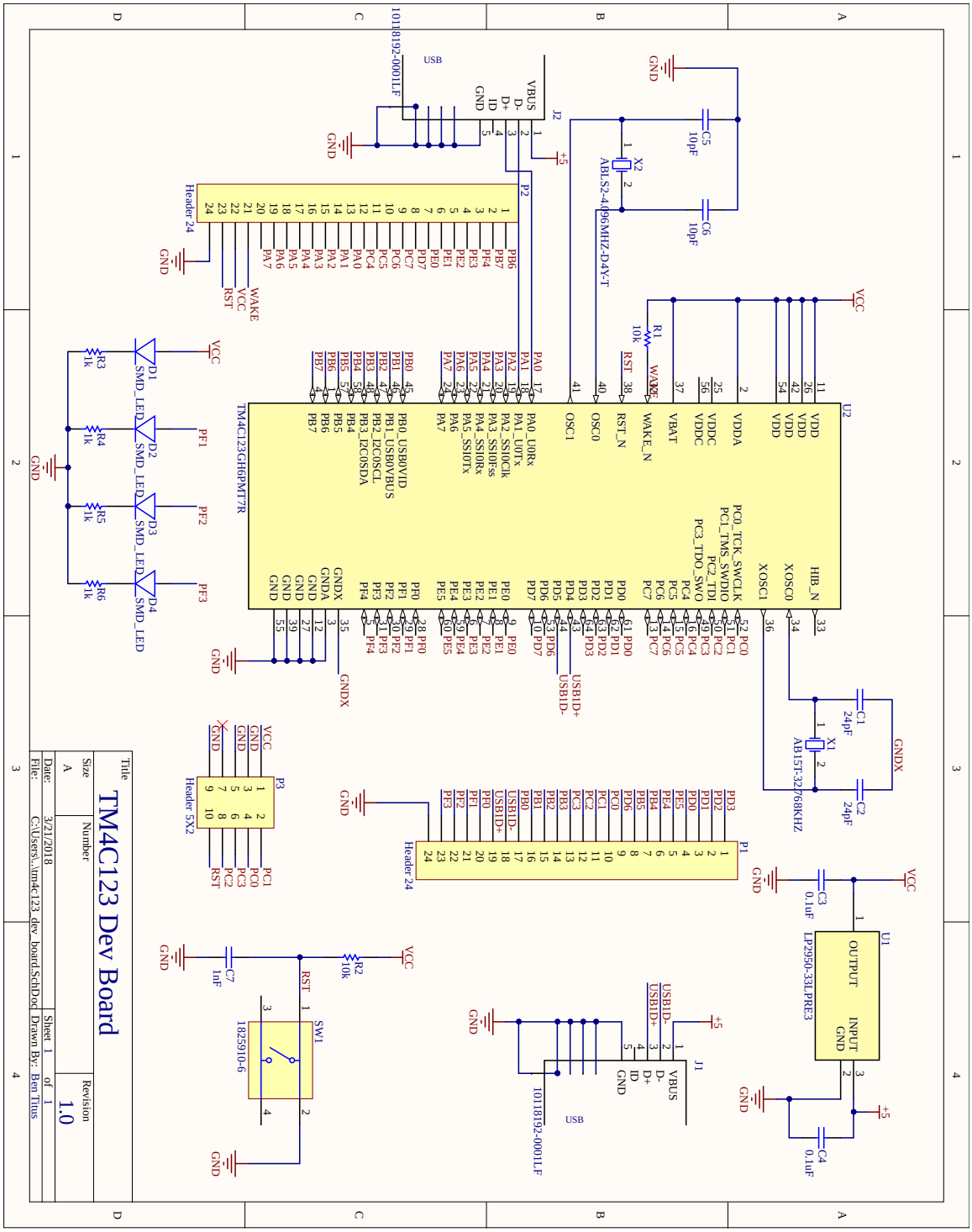

Figure H.1: Circuit diagram for tm4c123 dev board PCB

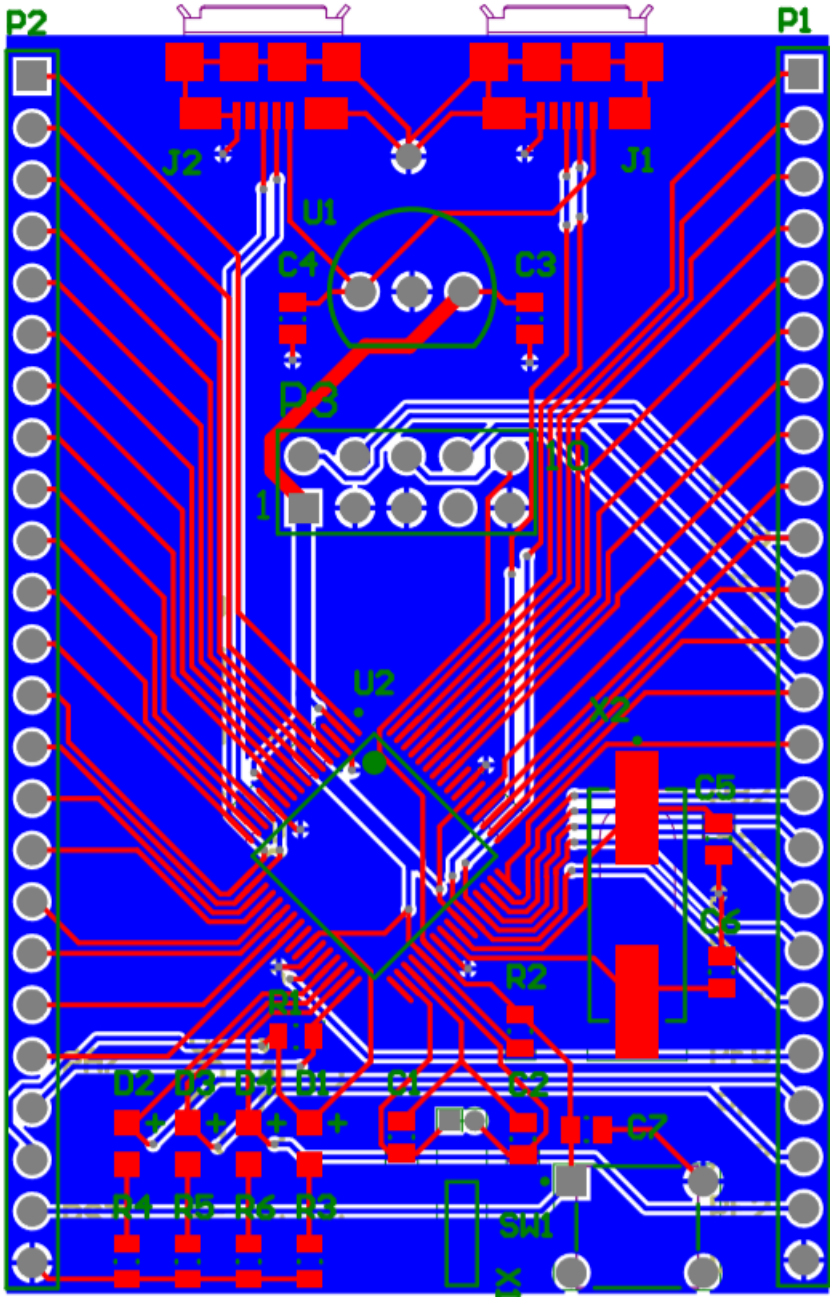

Figure H.2: PCB composite for the load cell amplifier PCB

Comment	Description	Designator	Footprint	LibRef	Quantity
CAP0805	0805 (2012 Metric) Chip Capacitor	C1, C2, C3, C4, C5, C6, C7	CAPC0805(2012)145_L	CMP-1590-00003-1	7
SMD_LED		D1, D2, D3, D4	0805 Diode	SMD_LED	4
10118192-0001LF	CONN USB MICRO B RECPT SMT R/A	J1, J2	10118192-0001LF_10118192-0001LF(Primary)	10118192-0001LF	2
Header 24	Header, 24-Pin	P1, P2	HDR1X24	Header 24	2
Header 5X2	Header, 5-Pin, Dual row	P3	HDR2X5	Header 5X2	1
RES0805	0805 (2012 Metric) Chip Resistor	R1, R2, R3, R4, R5, R6	RESC0805(2012)_L	CMP-1591-00002-1	6
1825910-6	Tact Switch, SPST-NO, 0.05 A, -35 to 85 degC, 4-Pin THD, RoHS, Bulk	SW1	TECO-1825910-6_V	CMP-1684-00021-1	1
LP2950-33LPRE3	Imported	U1	LP3	LP2950-33LPRE3	1
TM4C123GH6PMT7R	Imported	U2	PM0064A_N	TM4C123GH6PMT7R	1
AB15T-32.768KHZ	Low Frequency Cylindrical Watch Crystal, 32.768 kHz, -20 to 70 degC, 2-Pin 5 x 1.4 x 1.5 mm THD, RoHS, Bulk	X1	ABRA-AB15T-2_V	CMP-1326-00001-1	1
ABLS2-4.096MHZ-D4Y-T	Low Profile Surface Mount Microprocessor Crystal, 4.096 MHz +/-30 ppm, 180 Ohm, -40 to 85 degC, 2-Pin 11.4 x 4.7 x 3.3 mm SMD, RoHS, Tape and Reel	X2	ABRA-ABLS2-2_V	CMP-0277-00002-1	1

Figure H.3: Bill of materials for the load cell amplifier PCB

Appendix I Code Repositories

Joint control board and base code repository:
`https://github.com/bentitus13/MIRA_Joint_Board_Code.git`

Joint board testing repository:
`https://github.com/bentitus13/MQP_TivaWare_Tests.git`

Software repository:
`https://github.com/atags22/MavenMira.git`

PCB repository with Altium files:
`https://github.com/bentitus13/MQP_PCBs.git`