Chemical Manufacture Process Automation

Group 22

November 1st, 2019

Jason Scislaw – EE
Anish Umashankar – EE
Amanda Gilliam – CPE
Ernel Reina – EE

Sponsor: Helicon Chemical Company

Table of Contents

Figure Index

List of Tables

1. Executive Summary

Making new materials at the basic chemical level is a lot like cooking. To get a recipe right, you need to be very precise in the quantity of ingredients, the order in which you add them, and the methods used to add energy to the mixture. If you need to bake two cakes, you can do it manually, one at a time, and still follow the recipe to perfection. If you need to bake 50 cakes, the task becomes much more difficult. Our sponsor, Helicon Chemical Company, has designed a manual system to increase their chemical production from the 'several cakes' level to the 'dozens of cakes' level. Our project aims to automate this system.

The problem with Helicon's system, as it's currently designed, is that its production will be limited by the presence and, more importantly, the full attention of one or more technicians. If we can work with Helicon's design team to automate parts of their process, we could free up time and energy from the technicians to work on other things.

Automating Helicon's entire chemical process is outside the scope of Senior Design, and would result in their proprietary recipe being shared with the world forever. For these reasons, we find it necessary to narrow our focus to a small section, one that is generic to most chemical processes. A diagram outlining this section is shown later in this document (fig. 1). This section calls for mixing two reagents with pumps, and purging with inert gas after. We will need to control the pumps and valves, as well as add sensors for feedback. Our design must include an intuitive user interface, and the control sequence must be customizable in case the process is changed. Most importantly, our hardware and software must be extremely robust, with redundancies wherever possible. Failure at any point of our design could cause Helicon's recipe to fail, with serious consequences. So why do this to ourselves?

What motivates us to take on this project will vary among the group members, but what we all share is the desire to become better engineers. For this reason, we are choosing a project similar to what real engineers in the industry face every day. These real-world projects carry more nuanced challenges then what would normally be faced in senior design. Not only do we face the technical and interpersonal challenges of this project, if we don't communicate effectively with Helicon's design team, we will be unable to integrate our system with theirs. If we cannot address the concerns of Helicon's leadership, our system will not be adopted. If our system is adopted, and fails, it will cause significant economic loss to the company. The challenges are daunting, but they are the reality of being an Engineer, and that is what motivates us.

2. Project Description

The following section gives a broad overview of the project. It outlines the motivation behind the project, objectives, requirements, and a market analysis.

2.1 Project Motivation and Challenges

An engineer that is not challenged by difficult projects, as a sailor that is not challenged by rough waters, is an engineer that does not grow. We have chosen a challenging project because we are motivated to become better engineers. This section will outline our motivations and challenges, and how sometimes they are one in the same.

Helicon Chemical Company is a start-up Research and Development firm focused on developing advanced composite materials using a fundamental chemical approach. Their current formulations put them in the Aerospace and Defense industry, with most of their future focus being in the Space industry. Over the past three years the company has experienced great success with its research efforts and their customer base is taking note. With such success brings challenges, however, and as a result Helicon is quickly outgrowing its current operations. With a backlog in their production schedule, the company is motivated to search for short and long term solutions to increase their throughput. This new pump system that our project is targeting is Helicon's answer to their short term production needs. The automation of this system is our group's attempt to help them bridge the gap towards more long term solutions.

The executive summary introduced the motivations for our senior design group. On top of the real world engineering experience to be gained, we are excited to be contributing to a start-up company doing cutting edge research in an exciting field, as well as working on a multidisciplinary project. These motivations come with unique challenges, which will also be explored in this section.

The nature of Helicon's industry and their position within that industry makes the opportunity to work with them an exciting one. The space industry is arguably one of the most inspiring fields, and is home to some of the greatest challenges, successes, and failures of our species' history. The engineers that pursue careers in space tend to have genuine passion and drive for it. The industry is also currently experiencing a game-changing revival with the explosion of the commercial market. On top of all of this, the most significant location in this industry is in our backyard! On top of the exciting field, Helicon's technology is cutting edge. Based on recent customer feedback, it can be said with cautious optimism that their work has real potential to solve some key problems in the accessibility of space. Once these problems, among many others, have been solved we may see humanity become a spacefaring civilization. While not all of our group members have our eyes on the space industry, we do see the future of it and are very motivated by the opportunity, indirect as it may be, to contribute to moving the industry forward.

This project is not only motivating on the macro level, due to the technology and the industry, it continues to be motivating on the micro level, due to its multidisciplinary nature. Not only will the skills required be technically diverse, but many will be non-technical. Helicon is a small company with less than ten employees. The nature of a company this size means that multidisciplinary skills are valued higher than extreme specialization. In dealing with such a company, our project will

require a similar philosophy. We will need to understand from the start that our electrical and computer skills are not enough to solve the challenges of this project, they're not even enough to solve just the technical ones. Understanding this, and developing our non-technical skills through this project, will make us more complete engineers.

The non-technical skills required for this project can be intimidating. The small nature of Helicon means that we will need to work with every level of the organization. This means understanding what is important to the members of each level, and adjusting our discussions and expectations accordingly. For example: a meeting with the CEO, Dr. Reid, will be more focused on the risks this project brings to his company, the rewards it offers, and the balance between them. Even though Dr. Reid is a Ph.D. Chemist, we will want to steer clear of specific technical discussions and questions. He will care less about *how* we implement our solution and more about *what* our solution will need to do. Meetings with him will need to be concise and relevant. A meeting with Ian, our technical point of contact (TPOC) on the project, will be more focused on the specifics of implementing our solution. He will be helping us integrate our two systems so technical questions are relevant and important to him. We will also need to work with the accountant to work out details on ordering parts and justifying expenses. Fortunately our team member Jason is the POC for this so it will be straightforward. One important consideration of Helicon's small size is the need to meet with the CEO, and ensure that his valuable time is well spent. Adjusting our approach to meetings based on who's at the table is important to maximize the value for both parties, and just one example of the soft skills that will be required for this project.

Another non-technical challenge we will face can also be contributed to Helicon's small size, as well as the nature of their work. Start-ups work with limited resources, and employees often have to take on challenges that no-one in the organization has solved before. Helicon is no exception, and this is compounded by the cutting edge nature of their research. This has a few implications for our project. First, the system we are working with is performing a simple chemical process that, with generic reagents, is fairly common. This system is new to the company and no-one in the organization has worked with it before. There will be no expert to solve our problems, and we will need to work with our TPOC to understand and fix any issues together. Second, Helicon has never partnered with a student team before. They are entering unfamiliar territory by sponsoring us. There are no guidelines in place for us to follow, and no previous mistakes to learn from.

Compared to the non-technical challenges, the technical challenges in our respective fields do not feel particularly daunting. To meet our primary goals we are working mostly with concepts and devices that feel familiar to us.

While Helicon's small size introduces various challenges, these same start-up characteristics make this project exciting and motivating. The lack of rules and guidelines means we have complete freedom in our ideas and our approach. The project is very open ended and our initial meetings with the Helicon team have confirmed this. This freedom can be intimidating, as the burden is on us to succeed, but if we do succeed we can be confident that it was deserved. This is important to address so we've made a concerted effort to outline what each party would consider as success, to be discussed in the following section.

Another source of excitement from Helicon's small size is the immediate and direct impact our work could have on the company's operations. In a large company, we could spend a year on a project just to provide the company with a footnote in their community engagement newsletter that is sent to the whole company and no-one reads. The only person in the company that cares being the project organizer who just wants a promotion. This certainly is a bit of an exaggeration, but not that far from the truth. With a small company, the story is much different. Helicon is designing this pump system as a critical solution to problems with the company's primary activities. If successful, this system sets in motion a series of major upgrades to Helicon's manufacturing process. Now, this system will work without our group's proposed automation upgrade, and our upgrade will not be adopted if Helicon is not convinced of its reliability and success. Despite this, it's clear that one of the presented scenarios is much more inspiring than the other. The possibility of our project being implemented in a critical system, combined with the cutting edge research this system will be enabling, and the industry the research is for, presents a perfect storm of motivation for success.

A third source of inspiration from Helicon's small size is the ability to work with all the management levels in an organization. This was presented as a challenge earlier, but it is also a source of motivation. Working with the CEO, the Project Engineer, the Lab Manager, and the Accountant allow us to experience multiple perspectives from small business professionals. These perspectives will give us valuable insight into the diverse skill sets required for the success of a real-world industry project.

If Helicon continues to grow, and if their pump system proves successful, the company will need to continue to innovate their manufacturing process. All it takes is one closed deal with a large customer, and a bench scale chemical process could need to be upgraded to a pilot plant. Such an upgrade would take an incredible amount of work, work very similar to what our group will be doing in this project. If Helicon decides to create an internal position to help with this process, members of our group will stand out from other applicants due to their intimate experience with the Helicon team. If this is something that excites us then the burden is on our group to make this intimate experience a positive one, and use this project to demonstrate that we are capable engineers. It takes some optimism to use this prospect as motivation, but it is one conceivable scenario that is worth exploring.

2.2 Project Objectives

After a thorough exploration of our motivations for this project and diverse challenges, along with a kickoff meeting with our points of contact in Helicon, our Outline for this project can be outlined. They will be different from each party's perspective, with some overlap.

From the perspective of Helicon, our project is a quality of life upgrade for a system that needs to work with or without our input. The most important considerations for them when working with us, in order of relative importance (most important to least) are robustness, reliability, company secrets, and compliance on DoD contracts. Their objective for our project is to provide this quality of life upgrade without compromising any of these considerations. If this objective is not met then Helicon will not implement our design into their system.

From the perspective of our group, our objectives can be expressed in tiers. Primary objectives are non-negotiable, if these are not met the project will be considered a failure. Secondary objectives are important to our core motivations for taking on this project specifically, if they are not met then we will pass, but be disappointed. Fringe objectives are fringe goals that are not critical to our real or perceived success, but would either marginally improve our project or are optimistic outcomes to our work.

Primary Objectives:

- Design and build a system that works
- Meet the requirements of EEL4914
- Maintain compliance with regulations that Helicon are obligated to
- Maintain secrecy of all Helicon's proprietary information

Secondary Objectives:

- Design and build a system that meets Helicon's objectives.
- Implement our system into Helicon's process.

Fringe Objectives:

- Develop a control loop to manage temperature
- Develop profile system such that Helicon can update our project with new profiles if the manufacturing process grows or changes.

2.3 Requirements Specifications

Listed below are the requirement specifications for our pump automation project. Included are both functional and nonfunctional requirement specifications as specified by us, Helicon Chemical Company, and related standards and constraints listed in section 4.

Requirement Specifications
The system shall contain two pumps, a control unit, and 4 valve control motors.
The system must be operable 90% of the year
The system shall have an emergency shutoff switch that safely powers the system off.
The system shall retain a complete record of the last chemical profile run.
The system shall provide the user with the ability to handle different chemical profiles.
The system shall utilize an LCD screen to display information to the user.
The system shall safely execute a chemical profile, or safely shut down.
The system shall be water resistant
The system shall evacuate lines with an inert gas before and after performing chemical manufacturing.
The system shall be able to handle 40mL to 2L batches of chemicals.
The microcontroller in the system must be able to control the 4 valve control motors.
The system shall be removable from Helicons components.
The system shall be able to be cleaned.
The system shall give an auditory warning in the case of a failure.
The system shall give a visual warning in the case of a non-critical malfunction.
The system shall be able to draw power from a standard US outlet.
The system shall be able to operate without user input after starting a chemical process.
The system must be able to safely shutdown in the event of a loss of power.

Table 1: Requirement specifications

2.4 House of Quality Analysis

Shown below in Figure 1 is a House of Quality Analysis for the automated chemical manufacture system.

House of Quality

Correlation matrix	
+ +	Strong positive
+	Positive
–	Negative
– –	Strong negative
	Not correlated

		Efficiency	*Output Power*	*Dimensions (main controller)*	Cost	Failure Rate	Implementation Time
		+	+	-	-	-	-
Cost	-	-	-	-		- -	- -
High Power	+	+ +	+	- -	- -	+	- -
Install Ease	+			+ +	- -	+ +	+ +
Accessibility	+			+ +	- -	+ +	- -
High Performance	+	+ +	+	- -	- -	+ +	- -
Targets for Engineering Requiremnents		>70%	<50W	12"x12"	<$1300	<0.1%	<One Week

Figure 1- House of Quality Analysis

3. Research Related to Project Definition

This section explores existing projects and products that could be drawn on for inspiration. It also covers existing technologies that could be used to facilitate parts of the project, as well as the strategic selection of parts and components. It also contains an in depth comparison of microcontrollers.

3.1 Existing Projects and Products

Although Helicon's chemical process as a whole is unique to the company, the section of their process that we are focused on is a very common process used throughout the industry. This section will explore how other plants and facilities automate their manufacturing processes so we may have a better idea of how these types of jobs are done in a professional setting. This section also serves as a market analysis and idea gathering expedition.

3.1.1 SCADA

A similar control system that is popular in the automation industry is the Supervisory Control and Data Acquisition system. The Scada system is a mix of other software and hardware systems which come together to offer data to be monitored and controlled on site or from a separate location. A Scada system is also capable of saving data for bookkeeping. Going back and checking on previous sensor, motor, pump or other component data is a key feature and allows plant owners and engineers to assess possible problem areas or failing devices, as well as overall productivity. Scada Utilizes human machine interfaces to allow easy control of valves, switches or other control points. It focuses on the power of microprocessors such as remote terminal units (RTU's) and/or PLC's to interact with each other and the various different control subsystems each microprocessor is tasked with controlling. The various intelligent subsystems communicate via lan and then transfer data to the main monitorial CPU through lan or internet. Modern SCADA systems have Structured Query Language (SQL) databases and web based applications that facilitate plant information and control access from anywhere so long as an internet connection is available. Below, in figure 2, is an example of a SCADA system consisting of several PLC's, one controlling a pump system, another a conveyor belt, and one more connecting to the SCADA software on the desktop.

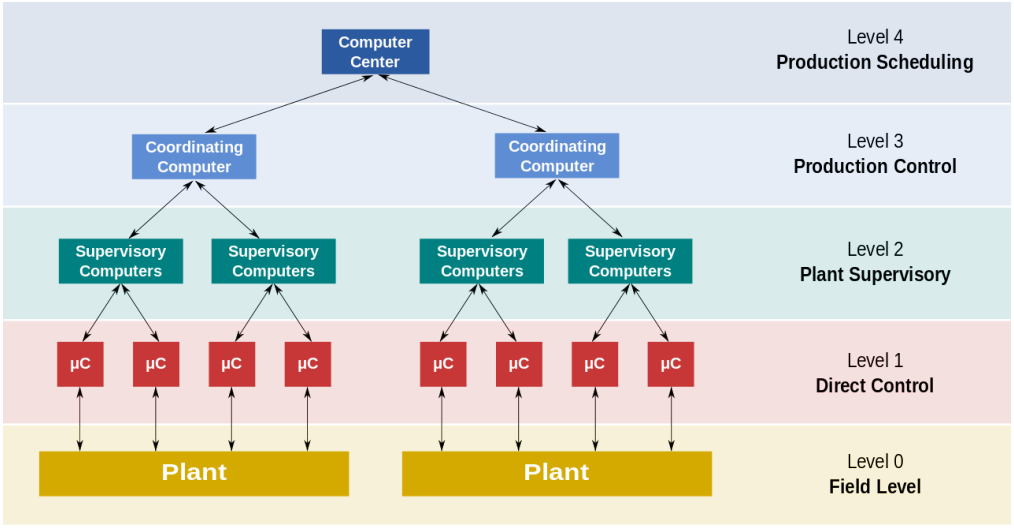

Figure 2- Scada System Diagram (Wikipedia)

SCADA is different from our project mostly in terms of scale. SCADA is for monitoring large plants where many different processes and data needs to be collected for efficiency or even legal reasons. For example a fully automated Ice plant using SCADA might need to control the ice making machines, the bagging process, and storage. Each of these would be a different subsystem with its own microprocessor. We are working on a much smaller scale, only controlling a few pumps, valves and sensors. This Essentially we will be programming and controlling what would normally be a subsystem for SCADA.

3.1.2 RTU

Remote terminal units are microprocessors in charge of interfacing with components in the physical world such as sensors and controllers. RTU's come with setup software for defining inputs and outputs as well as troubleshooting failings in installation or other possible issues with data transmission. An RTU has at least one complex circuit card which can have an array of different sections used to fulfill custom jobs as needed for the job. However, most consist of a CPU with an interfaceable communications option and cards capable of inputting as well as outputting digital and analog data. Communications are typically done through serial RS232 cables (also known as ethernet) and can support some standard protocols for communications such as Modbus, DNP3, IEC 61850 and many others. An RTU could also be for smaller processes such as alarms, PID controls, filters, and others.

An RTU can have a power supply with an AC to DC converter and a battery in case of power failure. Digital inputs come in either on or off and the RTU program is responsible for interpreting this for its application. Analog Inputs can come in the form of ampere ranges such as 0-1mA or in the form of Voltage ranges such as 0-10V. Digital outputs are simply on/off switches for powering devices that need to be controlled. Analog outputs are less common but theses can still be implemented for devices that can use analog inputs for control. An example of this is a separate RTU or a PLC that will use that analog input for its own purposes.

Our design should be able to function much like an RTU control system so using some of the common ones used in the industry to the capabilities of our final end product is a good way to measure its usefulness.

3.1.3 PLC

Depending on how important it is to the design our PCB has to be for grading purposes, we could elect to have it take on a peripheral role of data storage and instead have a Programmable Logic Controller (PLC) as our main control unit. A PLC is in essence a computer that will sit in our control box and be in charge of all the inputs and outputs of the process. PLC's are widely used in the industry for automation jobs as they are relatively simple to program and can do what would normally require a large amount of relay logic.

A PLC consists of two major parts, The CPU module and the input/ output module (I/O). These modules can control anything from 240VAC to 5VDC depending on necessity and can even include analog inputs and outputs. The conventional programming language used by PLC's is called ladder logic and it consists of mainly of inputs being labeled as normally closed or normally opened and once the opposite condition to these is met the line reaches an output. These inputs and outputs could also be internal lines but normally you'd want to instead use memory blocks for this. A PLC is normally programmed using a specialized software provided by the PLC manufacturer which can be used to watch the code work real time, find parts of code that are not being used, and overall provide a visually easier to debug process than other alternatives. Below, in figure 3, is an example of ladder logic programming, where -| |- (like input X2.2 Auto_PB) means normally open -|/|- (e.g. input X2.3, Manl_PB) means normally closed and the circle at each end is an output (M2.0, Auto_mode). These outputs can also be internal memory blocks, timers or other program blocks. The software also allows for other command lines to be visually next to each other for debugging ease.

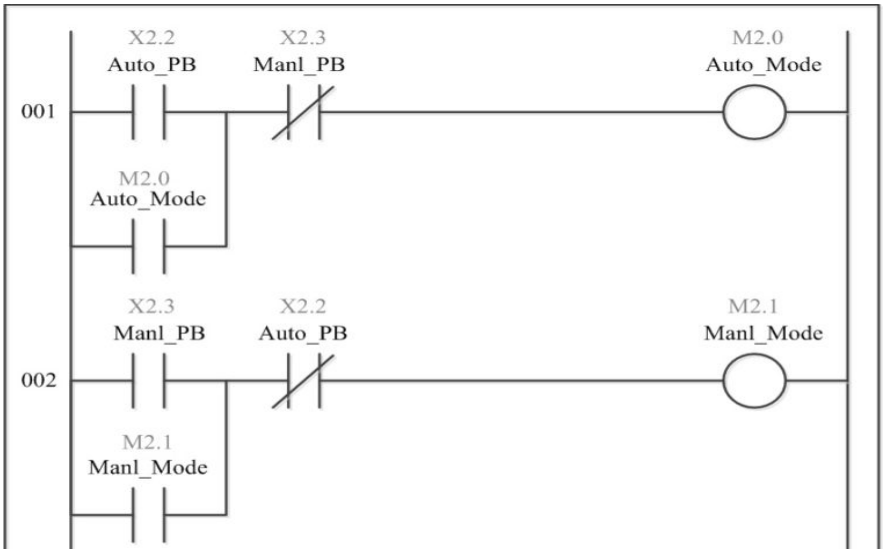

Figure 3-Example Ladder Code

A fringe goal is for Helicon to be able to upgrade the system by adding more components and processes and have a CCU that can compensate for such later on if they so need/want. The PLC has

plenty of inputs and outputs to begin with and adding more is usually as simple as buying an extra module from the manufacturer and attaching it physically and electronically with some basic instructions. The original code will likely not need to be changed much and only the new processes will need to be programmed. Another fringe design option was to be able to control the process from a completely different room. A PLC can easily achieve this by simply running a data cable from the CPU to the I/O module in the other room. Lastly it is able to connect to the internet using various protocols many of which most complex parts come pre-programmed allowing for ease of use not just by us but by anyone wishing to update the Helicon system later. In addition PLC's are usually made to be din rail attachable which will save us some time in the workshop.

Although a PLC is highly versatile it does have some flaws. For the minimum requirement we have set, which is for the system to work, it may seem like overkill, being able to do much more than is necessary and so making its significantly higher cost go to waste. Space could be a bit of a constraint as one option is to fit our control box onto the center of a cart with other devices on it. Although the PLC isn't exceptionally large of a device it, along with any extra modules, will take up more space than our PCB alone. Lastly the largest problem with the implementation of the PLC is our relative inexperience with these devices with only one member being familiar with how they work. Using the PLC under these circumstances could lead to issues programming the system to operate how we need it to. If nothing else we can use the options the PLC presents as something to compare our own CCU to keeping in mind cost and man-hours spent programming.

3.1.4 Automated Bartender

In essence, an automated bartender is trying to accomplish an end goal similar to our project, which is to safely mix substances to a set of specifications. Most of these projects include the use of at least one peristaltic pump, similar to ours, and the ability to choose between different chemical profiles, usually using an LCD screen. A classic example can be built using a raspberry pi as the 'brains' behind the operation and featuring a number of pumps to control the liquids being mixed.

Unlike with an automated bartender, where the main goal is the end product and it doesn't matter how fast or in what order things are mixed, the process really matters for our project. We must also equip our system with an emergency shutdown feature and logging capability as safety is a critical concern.

3.1.5 SmartCAFS Fire Suppression System

This is a system designed to mix compressed air, water, and a chemical foam to exact specifications for the purpose of fighting fires. Figure 4 illustrates the use of LCD screens to display sensor information and allow the user to change settings. This system differs in that it monitors engine RPM and is not temperature controlled. This system also does not need to be customized in the same way that our project does because even if the composition of the chemical foam changes, it will still need to be mixed with the same two things, air and water. That leaves the only customizable parts of the system the ratio of the elements and the rate that they are pumped out. Fire engines also do not use peristaltic pumps, but do have to take into account the corrosive effects of the chemical foam on pumps and electronics within the system, since they have sensors in direct contact with chemicals. This is something our project seeks to avoid.

Figure 4- SmartCAFS system by Hale Products

3.2 Relevant Technologies

The following section explores technologies that will be relevant to our project, and provides a background of available tools and technologies that may be utilized.

3.2.1 Process Control

The core technology for our project is process control theory. Formally outlined in 1922, process control is the practice of automating systems to achieve greater safety and consistency than could be done with human control alone. A key realization by early developers of this theory was the importance of stability, rather than general control. This stability could be maximized by separating error correction into three parts: error accumulation, instantaneous error, and rate of change of error. Each part is controlled by an integral, proportional, and derivative controller, respectively, together forming a PID controller.

The fundamental building block of any control system is a control loop, shown in Figure 5. The control loop uses a controller, a sensor, and a control element to manage a single process variable by reading its state and reacting accordingly. Figure 5 is an example of a control loop that we will need to implement in our project, in order to manage the flow of reagents in the pump system. In one iteration of the loop, a sensor (flow transmitter) reads the instantaneous flow rate and communicates this to the controller. The controller (flow controller) compares this value to the desired flow rate, and computes the instantaneous error, the accumulated error, and the rate of change of error. The controller then attempts to correct this error by producing an output signal

based on these error measurements and sending the signal to the control element (flow control valve). The control element adjusts the flow rate according to this signal and the process is repeated. In this example with an electrical system the process can repeat very quickly, and if the controller is designed properly, the system will reach steady state with zero error before there are any adverse effects.

Many of our control loops will be managing discrete control variables, i.e. setting the position of a valve to three different states. In these cases the controller design will be very straightforward. For the continuous variables, such as flow control and, as a fringe goal, temperature control, our application of control theory to create these loops will be one of our greatest technical challenges.

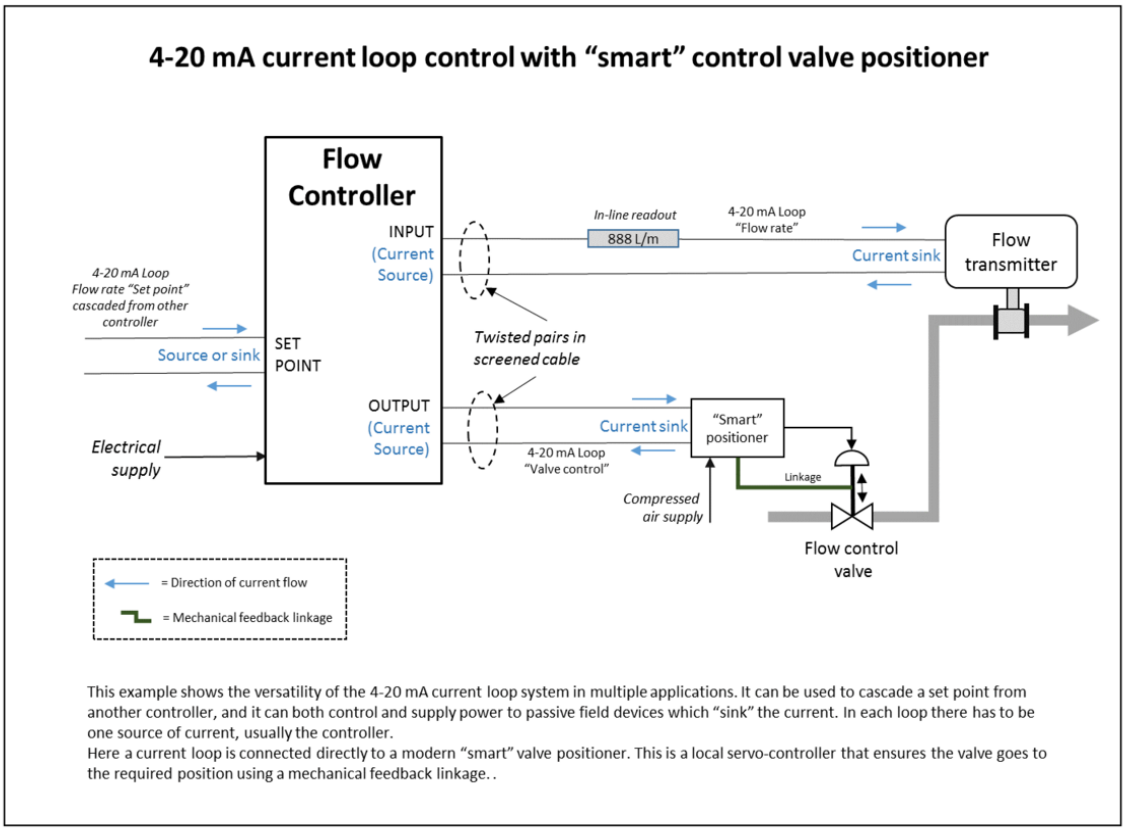

Figure 5 - Control Loop managing Flow Rate (Wikipedia)

3.2.2 Servo

Helicon's current design uses manual ball valves to control flow direction in the system. In order to interface these valves in a control loop, we have two options. We can replace the valve with one that is electrically compatible, such as a solenoid, or we can modify the mechanical valves with an electrically controlled motor. Initial discussions with our TPOC indicate the second option may be necessary. In which case our best option would be a servo.

A servo is a rotary motor that allows for precise control of position. The servos angular movement is broken into discrete steps, with each step corresponding to a signal level. A controller only needs to produce the corresponding control signal to command the servo to any specific position, constrained only by the step resolution. Built into the servo is an encoder to provide feedback to the controller in case of any deviation from the set point.

Possible obstacles to finding the right motor would be its strength, however for our needs a relatively small motor should be enough and if the first does not work then the ones needed for this project are fairly inexpensive allowing for trial and error to be a cheap decision making tool. These motors are commonly found requiring 12VDC to operate.

3.2.3 Peristaltic Pump

While we will need to interface with Helicon's system mechanically with servos, we will also need to interface electrically with their pumps. Two peristaltic pumps are used to control the flow of the reagents. A peristaltic pump has a rotor with rollers attached to the ends, shown in Figure 6, and tubing is threaded through a channel along most of the rotor's circumference. As the rotor rotates, the rollers pinch the tubing, driving material through in the direction of rotation. The material's flow rate is directly proportional to the angular velocity of the rotor. From the perspective of process control theory, the pumps in Helicon's system are the control elements in a control loop driving the flow rate variable of the pump system. We will need to add the controller and sensor elements to complete the control loop outlined in Figure 5 above.

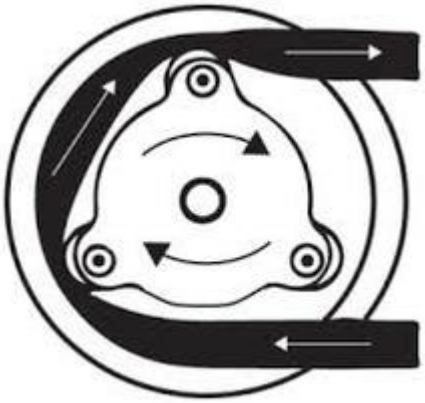

Figure 6- Peristaltic pump: basic schematic

3.2.4 Pulse Width Modulation

The peristaltic pumps currently implemented in Helicon's system have their own independent controllers. These controllers rely on human input however and must be modified or bypassed in order to control the pumps as part of our project. The pumps come equipped with DB25 female connectors with pins dedicated to third party control signals. These pins read an analog voltage from 0-20mV, or with an adjustment, 0-10V, and internal hardware translates that voltage to a flow rate. Our controller will be working with digital signals, so our system will need to implement digital to analog conversion. The most common D-A converters use pulse width modulation (PWM).

Every digital controller has a clock signal, which is a simple square wave at a constant frequency. Using a 5V clock signal as an example: The signal switches from ON (~5V) to OFF (~0V) in one cycle, with both states sharing half of the cycle period. In one cycle, voltage is set to 5V for 50% of the time, this is called a 50% duty cycle. If you were to read a perfect clock cycle with an analog voltmeter (with an analog low pass filter in between), you would see 2.5V, half of the cycle's peak voltage, as the peak voltage is only active half of the time. PWM is the practice of adjusting the duty cycle of a square wave to achieve an analog signal proportional to the duty cycle. The clock signal acts as a carrier wave, and the duty cycle is the voltage that is carried. Increasing or decreasing the duration of the ON state for each cycle will increase or decrease the voltage of the resulting signal. The output is still a high frequency digital signal, however, and much be passed through a low pass analog filter to remove the carrier wave. What's left is a relatively stable analog DC signal. It's clear to see that the maximum analog output for our 5V clock signal is 5V.

Looking again at the pump's analog inputs for controlling flow rate, we have to option of using 0-20mV or 0-10V. Unless we use a controller with 10V logic, we will need to amplify or attenuate our analog signal before it reaches the pump.

3.2.5 AC-DC Conversion

To avoid undue burden on the operators, it is important that our system operates on the standard 120V 60Hz electrical grid. Since are largely looking at a DC system design, we will need to implement AC-DC conversion to make this work. This conversion is done with a rectifier.

The standard rectifier for modern electronics is a switched mode power supply (SMPS). The functionality of a SMPS can be broken down into four stages: Input rectification, inversion, voltage regulation and output rectification, and regulation. In the input rectifying stage, the AC current is sent through a full-wave rectifying circuit (bridge rectifier), and then smoothed with a filter. In inversion, the now DC signal goes through a power oscillator which outputs a high frequency square wave. The voltage at this stage is still high (120V RMS ~ 170V), so a step-down transformer drops to the desired voltage with its winding ratio in the voltage regulation stage. Lastly, the output wave is rectified and smoothed to give the desired output DC voltage. The output voltage is compared to a reference voltage in a feedback circuit to regulate the output power.

One thing noted while studying the SMPS is that AC-DC conversion is achieved halfway through the process, so why not stop there? These devices, called linear regulators, exist and offer an advantage in design simplicity. These devices dissipate significant amounts of power through ohmic losses, however, and the SMPS was an attempt to minimize these losses.

An important consideration when designing our power supply is the voltage requirements of all the components in our system. With careful part selection, we can theoretically work with a single input voltage. More realistically, we may have parts that operate at different voltages. In this case we will need a multiple output SMPS.

Due to the complexity of adding this type of conversion to our PCB we decided instead to have seperate power supply convertors to power our system. We will have one for Motors and one for

our PCB as this is standard practice and they each require different voltages. By separating the task of power control off our PCB it will also be less cluttered in terms of modules and replacing it would be easier if an accident were to occur.

3.2.6 Piezoelectric flow sensor

Our first obstacle for sensors is measuring flow rate of the fluids. Piezoelectric flow meters or ultrasonic flow meters will be our best bet to measure the flow of reagents inside the tubes. These sensors convert flow rate into an electric signal using acoustic waves. These values can then be used by our CCU to carry out operations and make informed decisions. The ultrasonic waves will be in the MHz range and are cascaded into the liquid at an angle compared to the flow's direction. We have two different types of ultrasonic flow meters to consider, Doppler effect or propagation type.

Doppler effect flow meters come with the convenience of only having to install one piece of hardware which sends and receives waves. As its name suggests it uses the doppler effect as waves hit and rebound off of particles the frequency of the waves change which the flow meter then reads. The faster the fluid is flowing the bigger the difference between the emitted wave's frequency and rebound wave's frequency.

The Propagation type flow meter uses two different piezoelectric devices which send and receive signals from each other. Based on the difference of propagation times going with the direction of flow of the liquid and the those going against the flow we can calculate the flow rate. The advantage of this is that we can measure different liquids or gases without having to change how we calculate the flow rate, since with this method the material it is passing through does not alter our equations. Figure 7 is an example of two piezoelectrics in effect utilizing the propagation type of sensing.

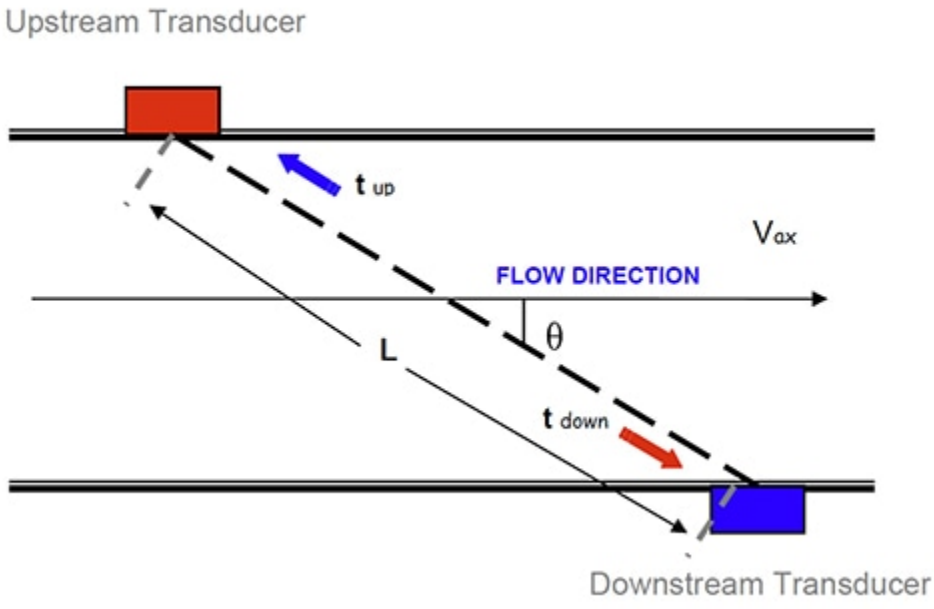

Figure 7-Doppler Effect Flow Meters

Another thing to keep in mind is the temperature the sensors can withstand as the liquids inside the tubing will be heated and we thusly need sensors that can operate under temperatures the tubing will be heated to. These sensors don't require much power only needing about 5.5Vpp to operate.

3.2.7 Pressure sensor

Although ideally we would want to interface with the Helicon system non-invasively, if we want to measure pressure we need to insert sensors into the points where we need to know specific pressure values. These sensors will be using the piezoelectric effect. As the amount of force the regent inside the tubes exerts on the wall(s) of the sensor the material the wala are comprised of create an electric charge which can then be turned into a signal, namely a 0-5VDC signal which we can then send to our CCU and have it make informed decisions based on the amount of pressure that has built up. Calibration of the minimum and maximum pressures is usually done on the sensor itself and will be determined on the job. Below is a diagram of how a conventional piezoelectric pressure sensor works. When a pressure is applied on the Diaphragm it creates a force on the crystal. This crystal has a unique chemical property and structure, such as quartz, which has it conduct a current (or voltage) through it which can then be measured and used as an analog input.

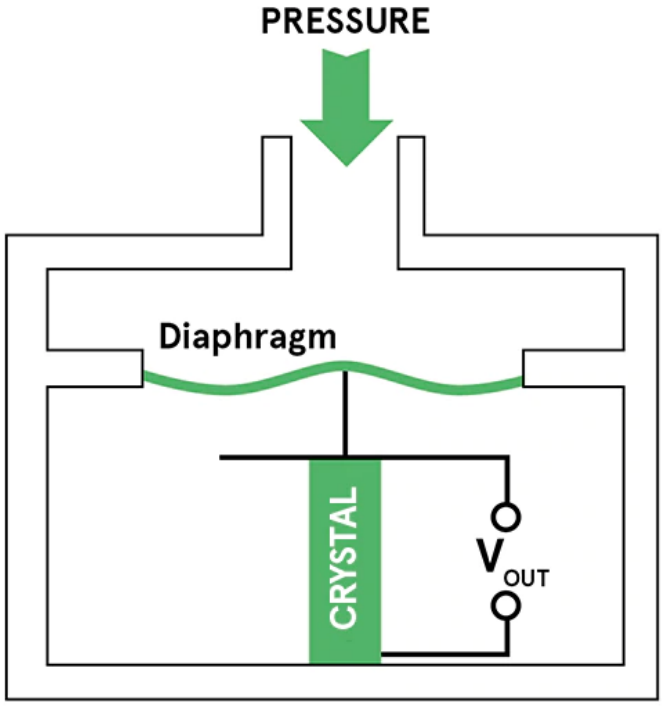

Figure 8- Piezoelectric Pressure Sensor

Currently in the industry there are three ways to measure pressure, in reference to atmospheric pressure, in reference to a vacuum or in reference to another sensor. These are commonly called Gauge pressure sensors, Absolute pressure sensors, and Differential Pressure sensors, respectively. A Gauge pressure sensor will show a positive reading when it is above atmospheric pressure and, if it can, it will show a negative reading if it is below it. The Absolute pressure sensor will always read a positive value and is best used when working with lower pressures that are typically below

atmospheric pressure. Differential pressure sensors are typically used when a certain difference in pressure between two different places in the industrial process is wanted or unwanted.

For this project it is important to make sure the tubes do not burst as they might contain flammable, corrosive, or some other type of dangerous material There could also be issues with the glass jars the regent is held in if the pump keeps trying to pull material that isn't there anymore creating a vacuum effect, which can shatter the glass and hurt workers nearby, or perhaps the pump itself could suffer from overexertion but that seems the least likely case. To this end it might be useful to have a differential pressure sensor at the two sides of the pump to make sure it is operating properly but this seems like a minor concern compared to the issues that could come from a tube bursting due to high pressure. Since we know an air vacuum will be in operation it will be best to use an Absolute pressure sensor or a Gauge pressure sensor which can read into the negatives. This is highly dependant on the chemical processes at work and whether precise pressure measurements are needed in order for the regents to react or be ready to react once they leave the system.

3.2.8 Breaker

A small one pole circuit breaker will be put into place before the power supply. This breaker will protect the other components from a current overload which could damage them. Breakers are usually better than fuses as they can be used multiple times, simply resetting them once they go off, instead of having to buy and replace fuses. A circuit breaker works by detecting large amounts of current running through them, usually by taking advantage of the heating or magnetic effects of currents. Typically these effects cause an internal switch to open which in turn triggers a mechanical response that uses springs or some other method of storing mechanical energy to pull the switch opening the circuit and thusly stopping all further electrical power from passing through and onto the rest of the system. A one pole breaker is all that is required as we will not be dealing with voltages above 120VAC. Breakers with two or 3 poles are reserved for 240VAC circuits with the 3 poles being for three separate lines and a neutral wire.

When a breaker trips the large load must be sustained by the breaker without overheating and also sustain whatever damage the electric arcs caused by a sudden opening of the system does to the breaker. A large current or voltage that trips the breaker will generate an electric arc. This arc is roughly proportional in length to the voltage and the heating is proportional to the surcharge in current. In order to mitigate this contacts inside the breaker are usually made of highly conductive materials such as copper or silver alloys which can help disperse the arc evenly across the material.

For our purposes we will likely be using a spring loaded lever circuit breaker which will trip if the current exceeds that which will damage our most sensitive component, most likely being our CCU circuit breakers typically come rated to protect a surcharge of 1,2,3 and then multiples of 5 Amps (10, 15,20 etc). As we will be working with relatively low power equipment which will plug into conventional wall outlets, which already should have protections of their own, this breaker will serve more of an extra layer of redundant protection or if for any reason the system is plugged into an incorrectly wired outlet.

3.2.9 Serial Communication

At some point in the project, we will need to be able to 'talk' to different peripherals attached to the microcontroller. For example, we are going to be logging runtime data to an SD card so that Helicon has easy access to it. Three ways to do this type of serial communication are UART, I2C, and SPI, all of which are supported by the microcontroller we have chosen.

UART is the slowest by far of the three and requires that both the sender and receiver of the signal have approximately the same baud rate. UART will likely be too slow for our data logging needs. I2C is faster and most Arduino compatible LCDs interface with I2C. Therefore, we will be using this for communication with the LCD.

The fastest form of communication our microcontroller provides is SPI. It is most reliable at short distances, making it perfect for the SD card module which will be mounted on the PCB. The downside to SPI is that it takes multiple wires, unlike I2C and UART which take two or three, but this won't be an issue for our application since it will only be interfacing with one device.

3.2.10 Liquid Crystal Display

Liquid Crystal Displays (LCD), are everywhere, from phones to televisions to computers. They are anywhere that requires a user to view information, which will be the case for our project. They work by utilizing liquid crystals, as the name suggests, polarizing films, and electrodes to create images.

LCD screens have relatively low power consumption as opposed to CRT monitors. They can be straightforward to work with given the right microcontroller, and a good quality LCD will last a long time. On the other hand, they can be relatively expensive for larger screens and there is a wide margin of quality between manufacturers. Over time LCDs can develop dead pixels, which are transistors that have stopped working and can no longer manipulate the crystals around them as they should. These dead pixels appear as black spots on the LCD screen.

For our project, there are a wide variety of LCDs to choose from, some of which are made specifically for hobbyists and have introductory tutorials. These different LCDs operate at different voltages and require different numbers of pins on the microcontroller to control them, the most common being controlled by 16 pins. All serious considerations in the choice of an LCD screen.

3.2.11 User Interface

The real challenge of the user interface will be having it seamlessly do the same operations as Helicon's current interface that controls the pumps while adding in new options like controlling the motors which move the valves. In addition, our UI should be operable by workers using gloves and be resistant to grime and chemicals. With these requirements in mind, one option is a keypad based human machine interface covered with a protective plastic to keep it from getting dirty.

One of the most critical aspects of our design will be the UI. Section 5.4 outlines our design considerations in detail, in this section we will explore what a UI is and the different types.

The main user interface is likely to be an LCD screen paired with tactile buttons for user input. This type of graphical user interface, or GUI, is a form that most users have experience interacting with,

unlike command line which appeals to a subset of users. This will increase the systems usability, and should also 'user proof' the system by cutting back the number of interfaces that the user is interacting with. By strictly controlling the amount of input a user can have on a system, it is possible to mitigate user error.

3.2.12 Relays

Although our CCU should be able to handle most operations a Relay would be used for, it is still important to research their uses incase their operation proves useful for logic done outside our processor. A relay is an electromagnetic switch that uses a small current to turn a larger power line on or off. This is especially useful in control systems since it grants the ability to use a small current supplied to the relays from a with a weaker, more sensitive electronic device to turn on much larger power intensive device like a large motor. A normally open relay is turned on when power flows through the coiled wire which creates an electromagnetic field which pulls a switch that closes the accompanying circuit. This accompanying circuit switch is held open by a spring or some other mechanical method. In the image below the current flowing from 1 is going into the coil which closes the switch in circuit 2.

Automation used to be comprised of boxes of "relay logic" where many relays were implemented such that when conditions were met an output signal would power a device. For example two relays would have to be in the on position, each one for a different reason, in order to turn on a pump. This was later replaced by ladder logic inside modern automation microprocessors.

3.2.13 Wires

Knowing which wires work best for us is also a vital concern. Cable power loss is calculated using P=I^2R. R is easily determined from the American Wire Gage (AWG for short) standards. However most cables tend to have minimal power losses when under 1,000 ft. A larger AWG correlates to smaller wire diameter, larger resistance, and a lower maximum current. The most common ones are from 10-16 gage for control systems. If we stick to those there shouldn't be too much issue with power loss or holding cables in contact points using screws.

An additional concern is the wiring going outside of our enclosure. Typically cables leaving the enclosure run through a metal tube running from the enclosure to either another enclosure or to whatever point of destination. In order to keep our design flexible however we can't rely on this as removing the whole setup if it has metal tubes running to each motor is impractical and slow. Instead we will use insulated sleeves typically found in homes and standardized by NFPA for the purposes of running our motors and pumps. This way unplugging and simply pulling the enclosure and its wiring out is a simple task only requiring minimal unfastening of organizational components. These sleeves are also standardized to reduce the risks of electric shock and the risk of a fire, both of which are important for our project.

3.2.14 HMI-Keypad with LCD

A human machine interface that already has an LCD attached to it might reduce the workload as well as allow for better communication between systems. To this end an HMI with pressable buttons or keys will be a good match. These HMI's at the very least have some up,down, left right, ESC, and Enter keys. Many can also be programmed in C with the program being made on a computer and then transferred over through either usb to RS232, ethernet or usb on more advanced models. These can even be numerous enough to allow several connections to a variety of other hardware. In addition these HMI's can have ports for audio, which we can use to sound off alarms or other auditory cues. Being able to enter values using a number pad would also make things a lot easier for the users to enter anything from temperature controls to flow rates desired. Below is a picture of a possible model to use complete with arrow keys and number pad from Kinco.

These in particular also come with SD card storage for data. The hurdle would be connecting this to our microcontroller as many come premade to connect either PC or PLC's. However there are others that are able to simply read inputs and write onto outputs without needing to be programmed to communicate with a different system. This would be highly beneficial as we can simply have our microcontroller send and receive analog inputs and outputs allowing for less room for error when programming our system.

These HMI's can also be in with waterproof and/or dust-proof plastic covers, something that will be very helpful for our system as it could facilitate workers to interface with the HMI with dirty gloves instead of having to take them off, increasing human to machine data transfer times. Alternatively to this we can have a rubber casing surrounding this HMI and any other buttons that will see constant use to protect them from wear and tear.

3.2.15 Unified Modeling Language

Unified Modeling Language, or UML, is used in the design of software. It was originally created to handle object oriented software design, but its use has grown to encompass software in general, and even beyond that to other systems such as manufacturing flow. Ultimately, UML seeks to standardize architecting and documenting systems.

UML makes it easier to plan and document code in one step. It is not a language per se, but a pictorial representation of the elements of a piece of software. It also encompasses how these elements interact with each other and the varying states that a system experiences throughout a process. Having a software design standard to adhere to will create code that is better documented and therefore more maintainable. UML can be used in our project for this purpose.

3.2.16 Printed Circuit Board

Printed circuit boards, PCB, are circuit boards designed with permanence in mind. They provide a sturdy, easy to manufacture and sustain, and easy to design layout. They consist of multiple layers (this can be changed to better accommodate whatever design the user is creating) of copper and non-copper layers, each layer being used for many different purposes. Typically the division between these layers can be simplified down to "Is the layer a Plane or not?".

In PCB design there are normal layers, used for everything that one would think they are used for: routing, soldering, keep-out (constraint layer), and mechanical layers. These layers, physically, are essentially a blank canvas to which the PCB manufacturer will add the copper routing traces and leads as defined by the designer's specification in the PCB file. The opposite is true for that of the Plane, as these are typically used when there are nodes that repeat multiple times throughout a design. For example, in every working circuit design there is a ground node that can typically be shared amongst any stage of the circuit. For PCB design, the ground can be set as a whole layer of the board so that whenever the designer needs to connect a component to ground, they can just create a via (hole in the PCB connecting layers) and set it to the ground layer from the layer that they are working in.

In terms of physical makeup, these layers are essentially one large sheet of copper, as these are typically set to a specific nodal value like a specific voltage or ground. The exceptions do exist for this as sometimes, due to design limitations like size, budget, and resources these planes can be sectioned off in partitions to accommodate multiple voltages/ grounds on the same layer.

The ultimate purpose of a PCB is to allow for a more durable end product that can be easily mass produced. Prototypes are usually built using a breadboard since components need to be easily swapped in and out during testing and development, but when the final design is decided upon, this ease of change could lead to parts falling off or wires pulled loose. Not good.

A PCB provides a solid base to mount components on as well as a way to electrically connect all of these components. Because a manufacturer is essentially laying down the solder lines, there is less chance for short circuits or incorrectly connected components. Components are then soldered onto the board to fasten them down and connect them to the circuit.

While it is possible to make your own, PCBs are now relatively cheap to have made, which allows for multiple boards to be ordered from a manufacturer for a fraction of the time and money it would take to make your own. This is great because there is always a chance that a board could have a defect or become damaged during development. Though a board might be cheap to make, depending on where it is being shipped from, it can be expensive to ship and take up to a month to arrive. Lead times on PCB orders are something to be considered when planning out the timing of the design and choosing a PCB vendor.

To create a PCB, the board must be designed first in software such as EAGLE. Eagle will then be able to generate a Gerber file which the manufacturer can use to produce the board.

3.2.17 External Memory

There is a possibility that the amount of log data that we want to retain for our project will exceed the amount of memory provided on most microcontrollers. Log data will only be comprised of characters, worth between 1 to 4 bytes each, but the log will be accumulating information from software prompts for the entire duration of a chemical process, essentially hours of information. This could very easily add up to many KB of memory. A rough estimate of how much 4KB of RAM could hold would give us anywhere between one to four thousand characters, which would

amount to roughly the size of this paragraph. Not enough space for a log to really be useful in the event of an emergency when Helicon wants to track down the problem.

While these are rough estimates, it gives the general idea that memory will be a factor in selecting a microcontroller, whether through on-chip memory, or implementing some kind of external memory. This external memory would be used for the log alone, and would not contain runtime variables which could cause run time errors to retrieve, or stack memory which is advised by most manufacturers not to be stored on external memory.

One way to implement external memory is through the use of external SRAM. The bus to the external memory will have to be configured similarly to other peripherals during the device initialization. If the microcontroller is already equipped to handle external memory, then the process could be as easy as knowing what address to point to when you'd like to write to external memory, and let UART handle things from there. The ATmega128 has some support for adding external memory, through the use of an XMEM pin, and this is the microcontroller the project is leaning towards. The way ATmega interfaces with the XMEM pins requires a latch in the middle that a user will have to supply to make external memory work. If another microcontroller is selected that has less hardware support, then things could become a lot trickier.

There are a few examples online of others setting up as much as 512KB of extra RAM to the ATmega128, which takes some creativity since the ATmega only has 64KB of addressable space. While the process can be a bit daunting to set up, it seems easy to test. Attempting to allocate memory in external memory that has not been configured correctly will cause a fatal fault.

Another possible choice of storage for external memory is the use of an SD card, as this would allow the card to be removed and then taken directly to a computer to be read. This is a more user-friendly approach and would not require much knowledge on how to access the right part of memory for data retrieval. It would also allow the user to swap out cards as they became full.

Implementing SD as a choice of external memory is much more straightforward for our choice of microcontroller. The MKR1000 board that we are considering using as a prototype has an SD module built in, and there are many readily available SD modules for purchase when we lay out our own PCB.

3.2.18 Stepper Motor

A stepper motor works much like a servo motor in that it has the ability to move discrete steps and accurately position itself. These steps are outlined on the product description in the form of degrees per step. However A stepper motor has a much larger amount of poles as a servo (50-100 versus 4-12). Each pole is a north or south magnetically generated pole and is what allows motors to spin. Stepper motors don't need encoders to determine their current position thanks to the large amount of poles they have. Stepper motors instead move 1 step at a time whenever they get pulse.

In exchange for this precision a stepper motor loses out on being able to achieve comparable torque at high speeds. This is because the stepper motor to move quickly has to go through many more current pulses through its windings in order to make 1 complete turn. It is unclear if speed of rotation

is a considerable design constraint as the valves these motors must control serve more like a three way option selector. We can also reduce the effects of this loss in torque at high speeds by supplying it with more power. On the other hand at lower speeds the stepper motor has much higher torque compared to servo motors. In addition when no movement is taking place the stepper motor has a large amount of holding torque. A stepper motor does not need an encoder and just needs a driver. If we run the stepper motor in open loop constant current mode we can achieve positioning results like that of a servo for a lower cost but it also means that both the motor and the driver will heat up quite a bit. This could be a potential hazard to workers currently working on the machine and if the motors are too close to the tubes containing regents it could possibly mess with the chemical reaction. A stepper motor can also be controlled in closed loop mode but an encoder would then be required to know of it's position. Below is an example of what a stepper motor system requires compared to a servo motor.

Stepper motor rotors also don't have winding slip rings, or a commutator, both devices needed in other motors to ensure that electrical power flows correctly through the motor. A stepper motor is generally less expensive and we can save the hassle of dealing with the encoder otherwise required by the servo motor counterpart. Possible high acceleration/torque requirements could make this motor unideal for our applications however.

3.3 Strategic Component and Part Selection

A Solenoid valve works by having current flow through a cylindrical coil of wire inducing a magnetic field which pulls up the piston allowing for the flow of fluids. Otherwise in the off-state the valve is shut closed mechanically by typically by springs or some other mechanical means and does not allow for any flow of fluids.

Solenoid valves have to be directly connected to the Helicon pump system and therefore have to come into contact with the various chemicals that flow through the system. If we were to employ these solenoids valves instead of the mechanical ones currently in place a means to control the flow of regents we would have to carefully choose valves that won't react with the chemicals in addition to having to remove the mechanical valves and install these new ones in their stead.
A solenoid valve also only has 2 settings, on and off. The valves currently in place have 3 settings, flow left, flow right, and no flow. Implementing solenoid valves would have us fundamentally change the Helicon design which is something we are trying to avoid in our non-invasive approach.

On the upside the valves being simple two state devices will give us an easier time programming. Servo motors will require servo drives which will have to be programmed and controlled through our CCU. To that effect the solenoid valves can easily be controlled through simple on/off outputs.

After much consideration we went against the idea of solenoid valves. Due to their on/off nature they would have been much more difficult to implement into our design that wants to control the valves in 3 states. It would have required much more physical alteration of the Helicon system which is not what we want to do.

3.3.1 Box Enclosure

The enclosure that will house all our delicate equipment has to be able to possibly withstand dust, lightly splashes of liquids, and possibly corrosive material. To this end we have the options of getting metallic or nonmetallic structures. These can be stainless steel, Polycarbonate, fiberglass and a few others. Of these, fiberglass enclosures and polycarbonate enclosures comply with NEMA standards for being resistant to corrosion or non-corrosive respectively. The cheaper option is the polycarbonate one and seems like the best choice for our project. It can even have a see through cover so technicians can better assess if something is going wrong internally.

With this in mind we will be using a 10x14x6 fiberglass enclosure box standardized with Nema 4x so it can resist splashes of water or other liquids and is also non corrosive in case of an accidental spill of hazardous material. This enclosure will have to be adjusted to allow us to mount all the different components placed upon it, such as the push buttons dials and molex connectors. The design is further explained in section 5. Originally we had planned to simply cut out a wooden box to use as our enclosure but apart from from violating some electrical standards it also would have made the design more difficult and would not have met the goals set by helicon to be splash proof and corrosive resistant.

3.3.3 DIN Rails

A Din rail is long metal (carbon steel) bar curved at its longer edges so as to make a place for electrical components such as breakers, relays and even industrial equipment such as PLC's to be mounted into an enclosure with ease. There are three widely used types of din rails. The hat type which is the most common for the previously mentioned electrical components. The C section type is an older model but still sees use in heavy power supplies or transformers since it's geometry provides great support. Lastly there is the G type or J type which is similar to the C type in that it offers a greater amount of support for larger components.

Our design will most heavily use hat type din rails as we will not need to work with large electrical components that would require the additional support which the other two types offer. In total we will need twenty seven inches worth of this type of DIN rail. Cut into three nine inch pieces for our top and bottom terminal block rows and one row to hold our circuit breaker and power supply.

3.3.4 DIN Mounted Terminal Blocks

These blocks are normally mounted above and or below enclosures and are a way to help organize cables coming into the enclosure and cables inside the box itself. They tend to be fairly standard with one socket on each side of it which is tightened by fastening a screw which clamps down on any wire inserted. The main thing to consider when choosing what Terminal blocks to buy and use is their color for color coding organizational purposes. For example we might want one green terminal block for ground, a set of four grey ones for our incoming flow sensor inputs, and a set of 2 blue ones for pressure sensors, etc. The colors we will use will be white for the live wire, black for neutral, and green for ground on the top terminal blocks with 3 grey ones as spares. On the bottom terminal we will try to match the colors shown in the molex connections in figure 35 our wiring harness diagram. So one red terminal block for +12VDC power, two black ones for servo motor as well as pump power grounds, one grey one for signal ground, four blue ones for our output lines and the rest are green which are for input lines. Having our colors match in as many places as

possible will make the building of the system as well as memorization of how it works considerably easier. There are also specialized bridges to connect several terminal blocks together in parallel but this won't be necessary for our design.

3.3.5 Push buttons, knobs and other inputs for HMI

We have a large array of possible HMI buttons to choose from if we decide to go about designing it ourselves. Just to list a few, we have push buttons that are a one time press, push buttons that are more of a toggle on and stays pressed until pressed again to release, switches which can satisfy the same conditions as the push buttons mentioned or can have more than two states (might be useful for valves as they are also 3 state), push button switches which are two in one and basically don't allow them both to be pressed at the same time (useful for on/off operations), and of course these can all come with some LED indicators to show which is currently active.

When we go about choosing which input mechanisms to apply to our HMI we need to primarily ask three things, what the function is they need to serve, whether the inputs available are rated to handle the power going through the cables, and lastly whether misusing the input can potentially dangerous consequences. For example a knob that controls the strength of the pump with no limits could see the pump over exert itself or have the pressure build in the varying weak points of the Helicon system and cause a rupture.

3.3.6 Alarms

It's imperative that our system not only recognizes a problem and reacts accordingly in an emergency but also notifies the workers and technicians that the system encountered an error. This can be done with the use of alarms and lights. Simple LED lights should be used for minor errors such as invalid value inputs or an inability to connect to the rest of the Helicon system. Sound based alarms should be reserved for possibly dangerous or equipment damaging emergencies such as a leak (a sudden unpredicted loss of pressure in the tubes). These sound alarms should be a simple control by our intelligent system to simply turn it on or off, meaning that the sound it would make should come preinstalled or alternatively we can code it in ourselves through a speaker.

The other type of alarm to consider is one that detects certain dangerous chemicals in the atmosphere if there is a gas leak which Helicon is working with. Integrating a device such as that into our system should be as simple as the others but such systems might be better left separate as an extra precaution.

3.3.7 Out of Enclosure interfacing

We want our enclosure to be easily separable from the cart if necessary so we need the enclosure to have non-permanent connections to the outside components, namely the pumps, motors, and power. For power an AC power entry module shown below in figure 11 was selected as the best choice since it comes standard and is easily replaceable if damaged. As for our pumps and motors we will have a molex connector with all the ports necessary for all of our wires to run in. molex connectors are also easy to instal and we have room for options if more connections are required.

3.3.8 LCD

We need an LCD to display menu options for the user. A 20x4 character display is a good balance of cost versus functionality. It gives us enough space to display all the text necessary for menu options and still leaves room to display user input options and button functionality. We have the option of forgoing button functionality and using a touchscreen LCD, but these tend to be much more expensive and will also pose a problem for users who will be wearing protective gloves.

Most LCDs made today include a Hitachi HD44780 LCD control chip, so this doesn't narrow down the search for an LCD much but it is worth noting that our choice of LCD must have this spec to make it significantly easier to program. The color of the LCD, weather white characters on a blue background or black character on a green background doesn't really matter for our purposes. Since both are equally readable, and if all other things are equal, the cost would be a deciding factor there.

The final choice of LCD is RioRand's 20x4 white character on blue background LCD. It's reasonably priced at about $8, works well with the Arduino LiquidCrystal library, and seems straightforward to power and control. The only drawback is that the quality of the LCD might not be great. There is a chance that it will not stand up to the repeated use it will get over the lifetime of the finished product. There was not much information provided by the manufacturer about expected lifetime of the LCD, nor was there much information provided by any other manufacturers about their LCDs within this same price point. During senior design 2, we will be consistently using the LCD during development, which will be a good test to see how durable it is. If it can't hold up, we will have to consider other options, like perhaps going with a more expensive, commercial-grade LCD.

3.3.9 Power Supplies

We had initially planned to have the power conversions be done by an extra module on the PCB however due to difficulty in the design as well as heating concerns we decided to instead have seperate power supplies take care of the issue. We chose two power supplies. We needed one to supply enough power to four servo motors each one requiring 12VDC anc 0.4 amps. We chose a power supply rated for 3.3A and 12 VDC which should be enough and have some left over. The other power supply just needs to cover our microcontroller which needs 5VDC, We chose a Power supply that is rated for 2.4A and 5VDC to cover that. Both of these are products from MiWi a well known company for electrical components. Possible issues that may arise from using these is the amount of space they take up compared to the module on the microcontroller. We have fourteen inches to work with lengthwise and there are other ways of saving space in our enclosure if we need to.

3.3.10 Backplane

Since we will be using a standardized enclosure we will also need a backplane to hold our Din rails and conversely hold all of our electrical components. The Backplane is put in place to as to not have to modify the back of the enclosure by fastening screws onto it and damaging or simply breaching it from the back. Most enclosures come with preset screw holes for back planes, ours included, and we will be using those to mount it. The backplane is a somewhat thin sheet of metal

and can be grounded and thusly used as an extension of ground. It also helps in dissipating heat. Our backplane will be slightly smaller than our enclosure, with it being 9x13. We could not find one of this size for a reasonable cost so we will attain a slightly larger one and machine it down to what we need it.

3.3.11 USB DC Module

We require a conversion from two wires with a 5 volt differential to a usb module in order to power our microcontroller. We want the PCB design to be as similar to typical current market PCB designs such as arduino in terms of ability to repurpose and ease of use. For these reasons we want to power it with what many market designs use, a USB port. To this end we will use a step up boost module which connects takes in 1-5VDC and converts it onto a USB port.

3.4 Possible Architecture and Related Diagrams

This section covers an in depth comparison of microcontrollers and an exploration of FPGA design. Both are considered as a possible means to control our project, and the pros and cons are weighed for both.

3.4.1 Microcontrollers

A microcontroller is at the heart of a control unit. It is comprised of some sort of processor, memory, and input/output devices. There are dozens of microcontrollers on the market to choose from, each with varying levels of capability. Choosing a microcontroller for prototyping that will be easy to transition from when the final PCB design is finished is imperative.

At this time, Helicon has expressed concerns about having the system connected to the internet, so IoT capabilities can be discarded from consideration in choosing a microcontroller. Otherwise, the selected microcontroller must transition well to a final PCB design, be able to support multiple I/O ports, and be compatible with an external source of memory, such as an SD card. The ideal microcontroller will also be well documented and have a robust development community to make troubleshooting easier.

3.4.1.1 ATmega series

The microcontroller inside the Arduino Uno is the ATmega328, which has an 8-bit, 32 register RISC processor along with 23 general purpose I/O ports, 2KB of SRAM, and programming via USART. The ATmega series also presents a relatively approachable way of working with LCD screens and configuring UI output.

Arduino boards present a simple, straight-forward solution for rapid prototyping a control unit to test valve and pump control based on sensor inputs. Arduino boards are relatively cheap, easy to program, and have many readily available tutorials. If this microcontroller is chosen for the final PCB design, it will make transitioning from an arduino prototype much easier since the microcontroller will be the same.

Unfortunately, the ATmega328 might not have enough I/O ports for our purposes, but the ATmega128 has double which makes it a good contender for choice of microcontroller.

3.4.1.2 ARM AMD series

The ARM AMD series of processors that are designed for low-power high efficiency projects such as ours. Some of the advantages they provide is they are very cheap compared to other processors. This paired with the fact that, compared to TI MSP430x series microcontrollers, they have less I/O pins allows us the flexibility of purchasing two ARM AMD series processors for around the price of one MSP430x series microcontrollers. The inclusion of two ARM AMD series processors allows us to create a master-slave relationship between the two processors, which provides ease of programming to one microcontroller while controlling two. This effectively negates the downside of not having enough pins as we can take the I/O pincounts of both microcontrollers and add them together.

The two ARM AMD series processors that we have taken into consideration are the ATSAMD21G18A-MU and the ATSAMD11D14. Our choice for these two processors stems from the fact that these two processors provide all the needed specifications such as adequate memory, I/O ports, frequency, power, communication type, and cost. In addition, these two processors are the same two processors that are used in the arduino MKR family boards, ZERO and motor carrier. In terms of function, these two boards provide exactly what we are looking to do in Helicon Chemical Company's existing pump system, which is to automate motors, log data, and control sensors. These two boards are made to work together and for this reason they will serve as a good starting point in terms of design and prototype testing. Also they are programmable using arduino's IDE and exist within the arduino development community which is filled with information, libraries, tutorials, and open-source code all found online for us to use or reference if/when we need it.

3.4.1.3 TI MSP430x Series

The MSP430x series features a wide range of memory configurations, I/O ports, and computing speeds. It is a very low power series of microcontrollers, which can be seen in its use of FRAM. Some microcontrollers in the series, such as the MSP430xG461x series, provide support for working with LCDs.

All of the group members are familiar with TI's MSP430x series through the use of the MSP430G2 LaunchPad in introductory course work that is part of the core curriculum. Many members have also taken Embedded Systems and so have experience with the MSP430FG4618 as well.

The downside to using a TI product is that there is less of an open source development community surrounding the technology. While TI has advertised product lines for beginners and students, such as the MSP430 LaunchPad, there is less support for doing more complicated things or working with less commonly used boards. While TI does have a dedicated support page and a small support forum, it might be more difficult to find answers to troubleshoot problems that are a step outside of the beginner zone.

3.4.1.4 TI C2000 Series

Besides the MSP430 series, TI offers the C2000 series of 32-bit, real time controllers that are geared towards efficient power systems. Most of the applications for this series seem to be vehicle related, as can be seen in the use of CAN for communication, but the series is something to consider given the large number of I/O ports it supports.

The downsides to this series, besides the lack of an online troubleshooting community, is that there does not seem to be any built in support for LCDs. While this series makes working with sensors easier than the MSP430, it would make programming the LCD significantly harder, especially with the lack of forum help. This series also comes with a larger price tag due to all the processing speed associated with a larger number of peripherals.

3.4.1.5 Silicon Laboratories

Silicon Labs uses ARM Cortex M processors in all of their 32-bit microcontrollers. Their microcontrollers are under consideration mostly for that reason as a few group members were hoping to get more experience working with this technology.

EFM32 Wonder Gecko is a very low power series of microcontrollers, and specializes in energy sensitive environments. This is not something under particular consideration for our design purposes, but this series does provide LCD support. Silicon Laboratories has some sampling for their microcontrollers so it could be possible to test one out during the prototyping without having to make a commitment and buy one. On the other hand, Silicon Laboratories suffers from the same problem as TI where there is not a robust community of forums for troubleshooting.

3.4.1.6 Comparison of Microcontrollers

The table below gives an overview of the different microcontrollers considered for quick reference to help make design decisions.

Comparison of Microcontrollers

	Memory	I/O Ports	Frequency	Communication Type	Cost	Power
ATmega328	2 KB SRAM	23		USART	~$7 (for 2)	
ATmega128	4 KB SRAM	64	16 MHz	USART	~$7 (for 2)	4.5-5.5 V
MSP430FG4618	8 KB RAM	80		USART, UART	~$6	1.8-3.6V
MSP430F67671	32 KB RAM	90		UART	~$6	1.8-3.6V
TMS320F28378S (C2000)	132 KB RAM	169	200 MHz	USB, CAN,	~$15	1.2-3.3V
EFM32WG990F256 (Wonder Gecko)	32 KB RAM	87	~48 Mhz	UART, USB	~$5	
ATSAMD21G18A-MU	32 KB SRAM	48	48 MHz	SPI, I2C, UART	$3	1.62-3.63V
ATSAMD11D14	4 KB SRAM	24	48 MHz	SPI, I2C, UART	$1.26	1.62-3.63V

Table 2- Comparison of Microcontrollers

3.4.2 FPGA

FPGAs are inexpensive integrated circuits, famous for being customizable after manufacture by the customer. They are an enticing option for this project because they can support many more I/O ports than a general microcontroller. One group member also has extensive experience working with them.

An FPGA can be programmed to run concurrent tasks making them faster at times than microcontrollers. FPGAs do tend to consume more power than microcontrollers, but power consumption will not be much of a design consideration for this project since the system will be plugged directly into a US standard outlet.

Using an FPGA would require it to be programmed using a hardware descriptive language such as VHDL or Verilog. All group members are familiar with verilog through previous coursework, but none have worked on anything this complex. This will add time to code development. Of all the options for a control device, an FPGA will be the most difficult from a programmatic standpoint. It also does not allow for an easy transition between a microcontroller prototype to a finished product that utilizes the FPGA.

3.5 Parts Selection Summary

The choice of microcontroller are the ATSAMD21G18A-MU and the ATSAMD11D14. Individually these two microcontroller do not provide enough resources to complete this project, but paired together in a master-slave format, as found in the two arduino boards MKR Zero and MKR motor carrier, these two microcontrollers are designed to handle exactly what our task entails. Not only is the combined cost of both of the microcontrollers less than its competitors, it provides more than enough memory, combined I/O ports, communication types, frequency, and all at a low input voltage. In addition, the ATSAMD21G18A-MU resides on the MKR Zero arduino board, in which it is already made compatible with a micro-SD card reader. This is an important compatibility issue that is already solved for us as we need a way to store small-medium amounts of log data in case of emergency situations that occur during usage of the pump system.

4. Related Standards and Realistic Design Constraints

The following section discusses in detail the related standards and realistic design constraints that apply to our project.

4.1 Related Standards

According to IEEE Standards Association, an organization within IEEE that develops global standards specifically tailored towards IEEE related industries, “Standards are published documents that establish specifications and procedures designed to maximize the reliability of the materials, products, methods, and/or services that people use every day.” There are many organizations that provide, accredit, and/or enforce standards like The American National Standards Institute (ANSI), which provides a search engine *NSSN: A National Resource for Global Standards* that helps users find national, foreign, regional, and international standards along with the related documentation.

Regarding our project specifically, the standards that we must conform to fall under both IEEE standards and OSHA standards. The IEEE standards that we must follow are relatively straightforward based on what materials and design we choose to use. In some cases, like the IEEE 1149.1 – JTAG standard, these will have to be included in our project anyway to ensure success. OSHA standards are made specifically with safety of workers in mind and seeing as we are expanding upon a system that could potentially be dangerous for workers, we must follow these standards very closely. OSHA standards, in general, fall under 4 main categories: Construction, Maritime, Agriculture, and General Industry. The OSHA standards that affect our project are related

to general industry and can be seen as general safety precautions that we must account for when working on a project of this caliber.

4.1.1 IEEE standards

Listed below are the IEEE standards that we need must comply with to ensure our project's success.

4.1.1.1 IEEE 829 - Software Test Documentation

As the name of this IEEE standard implies, this standard focuses on the meticulous documentation of any and all experimental activities regarding software testing. This standard specifies a set of documents (divided into 8-10 defined steps/stages) that each have their own set of documentation. In order for this standard to be used for any type of software or system testing environment, it specifies each document does not have to be produced and that there is no grading rubric regarding the information inside of the documents; the judgement of these falls upon the person or group following this standard. Listed below are the steps that we as a group will follow/document and our justification for its importance:

Master Test Plan (MTP): This stage is necessary as it will provide a generalized testing plan for our entire project and will have basic level documentation on all stages of testing, we pursue.

Level Test Plan and Design (LTPD): This is a combination of two levels documentation listed in IEEE 829, but we believe combining these levels will streamline our testing and overall documentation process. This level documents everything that is happening in our testing process. This includes, but is not limited to overall approach, all resources being used, components to be tested, and a general schedule of all the aforementioned tasks and when they are being done. In addition, this documentation will include a list of all the specific test cases along with their respective criterion, procedure, and ONLY our expected results (observed results will be in the Level Test Log documentation).

Level Test Log (LTL): This set of documentation is very straightforward, and will include most, if not all, of the data we gathered during testing. This documentation will also have a section specifying what problems we have in our data.

Master Test Report (MTP): This documentation serves as the conclusion of our set of documents and being so, summarizes all of the previous test documents and highlights all of the key information listed. The MTP should also serve as a 'resume' of sorts that we could show as a final document of our entire testing process.

4.1.1.2 IEEE 830 - Software Requirements Specifications

This standard describes, in detail, software that is in development/will be created in the future. This standard's general function and set of documents is essentially a user guide or agreement between the designer and user to ensure that the product is working as intended. Seeing as this project is not one that will be made available for all consumers, but instead for one specific consumer, this

standard is a key part of our project documentation. In order to comply with Helicon and ensure our project works as intended we must provide every important detail listed in this specification. Those details include, but are not limited to: purpose, general system overview, in depth features and functional requirements of all interfaces (User, hardware, software, and system), design constraints and reasoning, reliability, security, sustainability, cost, all hardware and software characteristics, etc. Seeing as this is a general IEEE documentation requirement, there is no specific grading rubric for how this must look or what information is needed; this is a user-designer agreement. In our specific case, our goals are to: fully describe the scope of our project, list constraints, provide accurate and precise software documentation for future modification if needed, list functional and nonfunctional requirements, provide excellent fault-avoidance shutdown procedure, and all other necessary information that we and Helicon believe to be needed in this documentation.

4.1.1.3 IEEE 802.10 - Standards for LAN/MAN security

This is a sub-standard in the *IEEE 802* family which outlines moving data in private and public networks. This standard is of utmost importance to follow as Helicon's proprietary information and all data that comes from this project are to remain confidential. Seeing as most of the data will be done through some specific hardware implementations, like UART or I2C, we should not really have to worry about data being intercepted from the pump system controller to computer receiving all of the data. We mostly will focus on making sure the computer receiving confidential data and information is secure through some means like using a WPA/WPA2/RSN protocol. Another way to make sure the data is secure in a private means would be to ensure that the computer transmitting and receiving the data from the pump system is not connected to the internet in any way.

4.1.1.4 IEEE 1149.1 – JTAG

The JTAG (Joint Test Action Group) is an IEEE standard for PCB testing. The JTAG is a physical component that is added to a PCB that is specific to an electronic component. The JTAG allows for users to debug, store firmware onto its respective component, and physical electrical testing. The debugging features of a JTAG are important for this project as they can be used to simulate certain I/O and their respective effects on our system. The storing firmware portion of the JTAG will be key for us as we plan to use a FPGA for our PCB and the JTAG will allow us to write to specific pins on the FPGA and allow for customizability. The physical electrical testing benefits of the JTAG allow us to test for faults, shorts, and any other physical routing issues that are occurring on our board. This is an obvious standard that we must follow as including a JTAG into one's system allows for many key bug fixes, testing, and design flaws that exist in one's final PCB. In addition, as per IEEE 1149.1 JTAGs can be daisy chained together (if system allows) for a master JTAG port on a PCB that will provide the benefits of JTAG for all components they specify. Figure 9 shows the state diagram and general flow of a JTAG implemented inside a controller.

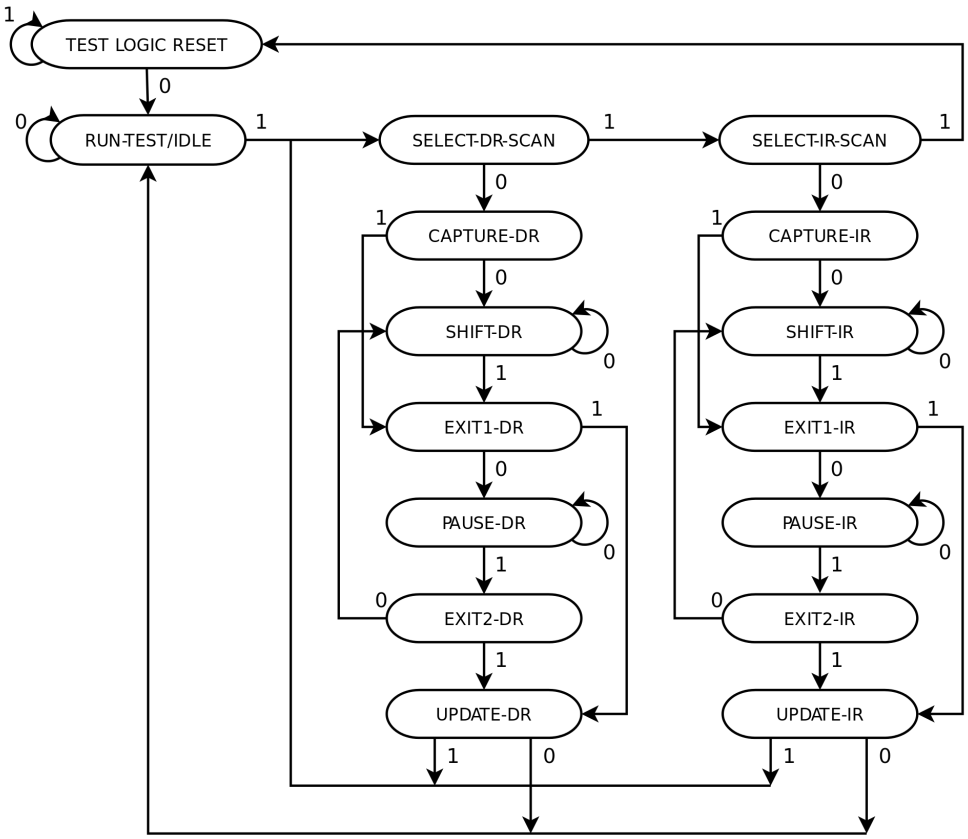

Figure 9- State Diagram of JTAG. Image under CC [1]

4.1.1.5 IEEE Power Switchgear, Circuits & Fuses Standards

This standard defines usage and constraints for switchgear, circuits, and fuses. This requirement is fairly straightforward as it relates to power conversion, voltage, and current standard, which all can be found in each respective electrical component's data sheet. It is important for us to document these and store the information and compile it into one document for easy access. This standard is not only important to us due to its emphasis on safety and organization, but also its information regarding various power conversions that may be present within our project. For example, we will need both an AC-DC power converter and a DC-DC power converter (it will be hard and generally unsafe to have all components on our PCB relying on an input voltage from one single power plane). The standard not only give general guidelines on the design of these power converters, but also identifies abnormal conditions to look out for during testing. With this information regarding design, test, and troubleshooting guidelines all stored for easy access and reference, analog and digital design and testing can be done with both ease and safety.

4.1.2 Design impact of IEEE standards

Listed below are the size of the impacts of the aforementioned IEEE standards and how they affect our design.

4.1.2.1 Design Impact of IEEE 829

The impact of IEEE 829 towards our project design is minimal. The IEEE 829 standard only specifies having thorough testing documentation and because of this, it has very minimal impact towards our actual design. The only possible impact that this could have to our design is in the hypothetical situation in which we have a working design in place and due to our thorough testing and documentation listed by this standard, we discover a fault in our design and from all the testing documentation we can go back into our design phase and correct the error(s). This standard serves as only a set of documentation guidelines towards our software, it does not present guidelines to actually create a working design.

4.1.2.2 Design Impact of IEEE 830

The impact of IEEE 830 towards our project is large. This standard describes, in detail, software that is in development/will be created in the future. This IEEE standard outlines our software components along with what components are and how each component works in conjunction with each other to produce the desired outcome, which is essential to our design. We must first complete and understand these software requirement specification documents in order to have a working design of the project. In addition, because the software is reliant on the hardware that we chose to use (microcontroller vs. FPGA), knowing what requirement specifications we need to meet from both our and Helicon's point of view is key to creating a working design.

4.1.2.3 Design Impact of IEEE 802.10

The impact of IEEE 802.10 is very large. This standard is not one that we have the luxury of ignoring about, as this standard is widespread and heavily used in the industry that Helicon is a part of. This IEEE standard outlines network protocol and security, and this has a major effect on our design as it affects how we decide to implement data transfer in our design. Initially we were planning on creating a design which would utilize a smartphone and give it the ability to remote into and give access to the pump system. This however has much to many security flaws and was scrapped from our design and set a stretch goal. In addition we were also planning on being able to access the data from one's smartphone, but this however could lead to another major breach in security as all data being transmitted and received from the pump system, to protect Helicon's privacy, is to be kept private. Lastly, the actual data transmission protocol also needs to be in question for our design, like whether or not we will use UART or I2C in conjunction with WPA, WPA2, RSN protocol, or whether or not we will even have a computer connected to the pump system connected to the internet in any way, shape, or form.

4.1.2.4 Design Impact of IEEE 1149.1

The impact of IEEE 1149.1 is very large. This IEEE standard outlines the uses for a design implementation of JTAG (Joint Test Action Group) in creating a working PCB. In most scenarios a working PCB of this caliber cannot even be created without having a implementing a working JTAG as the benefits of having one include creating complex and increasingly useful designs while having relatively easy troubleshooting. Seeing as the JTAG is a physical component that works in conjunction with a specific electronic component, such as an FPGA or DSP, they will impact the

design drastically as we will need to take them into account when creating the PCB and routing components. In addition to routing components, a common feature of the JTAG includes being able to be daisy chained along with other component specific JTAGs to create a 'master-JTAG' to troubleshoot, write-firmware, and test the physical routing to our board. This will be key to our design as almost all of our software and physical testing of our PCB will be done through communication with the JTAG.

4.1.2.5 Design Impact of IEEE Power Switchgear, Circuits & Fuses Standards

The impact of the IEEE Power Switchgear, Circuits, 7 Fuses Standards Collection is also a very large one as this IEEE standard outlines the proper electrical insulation needed for our project. According to Helicon, our pump controller must be properly enclosed and kept secure to avoid any complications with the actual pump, valves, or liquid/gas that is being transported in the pumps (the latter being of utmost importance). This affects our design heavily our entire PCB enclosure is kept in check by this. Our plan to stay in compliance with this standard includes creating a 100% electrical and thermal enclosed structure and have it near/but not a direct par to the pump system that is being enclosed by.

4.1.3 OSHA standards

As previously mentioned, OSHA standards are one that we must comply with in order to protect Helicon employees from any potential hazards that could occur from our design. OSHA standards are comprised of many general and industry specific (construction work, maritime operations, and general industry) standards, of which we are focusing on general industry as this applies the closest to the work that Helicon does. The OSHA standard that we must comply with is 1910 Subpart S - Electrical. Due to the fact that most of the information regarding the specific uses and day-to-day operation of the full device (controller, pump, data collection, chemicals, valves, and pipes) are proprietary to Helicon, we do not have to worry about specific OSHA related to toxic and hazardous substances. These standards include, but are not limited to:

1) 1910 Subpart D - Walking-Working Surfaces
2) 1910 Subpart E - Exit Routes and Emergency Planning
3) 1910 Subpart G - Occupational Health and Environmental Control
4) 1910 Subpart H - Hazardous Materials
5) 1910 Subpart I - Personal Protective Equipment
6) 1910 Subpart J - General Environmental Controls
7) 1910 Subpart L - Fire Protection
8) 1910 Subpart N - Materials Handling and Storage
9) 1910 Subpart Z - Toxic and Hazardous Substances

To reiterate, these standards are standards that we do not have to comply with as we are not employers and are not working with the chemicals, only the controller of the system that pumps and mixes the chemicals. Lastly, it is important to note that these standards are being met, just not by us, but rather Helicon Chemical Company themselves.

4.1.3.1 1910 Subpart S - Electrical

This subpart of the 1910 (General Industry) OSHA standards specifies electrical safety requirements that must be met in order to protect employees. This subpart of the 1910 standard contains many divisions such as safety-related work practices, safety-related maintenance requirements, safety requirements for special equipment, definitions and design safety standards for electrical systems. The division that applies to us for this subpart of the 1910 standard is the design safety standards for electrical systems. This subpart states “Design safety standards for electrical systems. These regulations are contained in 1910.302 through 1910.330. Sections 1910.302 through 1910.308 contain design safety standards for electric utilization systems. Included in this category are all electric equipment and installations used to provide electric power and light for employee workplaces. Sections 1910.309 through 1910.330 are reserved for possible future design safety standards for other electrical systems.”

4.1.4 Design impact of OSHA standards

This OSHA standard (1910 Subpart S) impacts our design heavily. As designers of the pump controller system, we must ensure the safety of whomever is using it. This includes but is not limited to making sure there are no electrical faults or shorts, no loose wires, proper cable management, electrically insulated enclosure of PCB, correct components (connectors, capacitors, resistors, etc.), and properly routed components on PCB.

4.1.5 NEMA standards

The National Electrical Manufacturere’s Association standards for electronic enclosures are meant to clarify concepts like waterproof, sealed, and dust free. These standards should be consulted for our enclosure. NEMA also rates motors these are for rating its frame size, efficiency, testing, and operations. For Stepper motors manufacturers came to an agreement to have NEMA rate them in terms of their size.

4.1.5 Design Impact of NEMA standards

The main impact of these standards is for us to decide whether we need our enclosure that houses delicate electronics to be resistant to things like hose directed water, or corrosion. The enclosure will be close to chemical processes which could result in a fire taken out by a sprinkler system or in a spill of a corrosive chemical onto the enclosure itself. If the likelihood of these situations is somewhat possible we should seriously consider using enclosures that are capable of withstanding such accidents so that helicon does not lose the control system as well. For our enclosure the following standards are relevant.

Table of NEMA Standards

NEMA Standard	Protection description
NEMA 1	For indoor use and provides minimal protection against solid objects falling into it or dirt falling into it. It also provides a protection against possibly hazardous components inside the enclosure itself.
NEMA 3	Can be for outdoor or indoor use and does everything NEMA 1 does but in addition provides protection against windblown dust, ingress of water from sleet, rain or snow. And won't be damaged by ice forming on the outside of the enclosure.
NEMA 3R	The same as NEMA 3 but does not protect against wind blown dust.
NEMA 4	Provides the same protections as NEMA 3 but adds protection against splashing of water and hose directed water.
NEMA 4X	Similar to NEMA 4 but adds protection against corrosion.
NEMA 12	Similar to NEMA 3 but specifically provides protection against fibers, lint and other small wind carried objects from entering the enclosure

Table 3- NEMA Enclosure Standards

As shown in Table 3, NEMA 1 is is great for a low budget design that does not require much worrying about liquids. NEMA 3 is not a much better upgrade for our purposes since our process will take indoors and in an environment where ice forming outside the enclosure simply will not happen. NEMA 4 is closer to our ideal since light splashing is something that Helicon was concerned about. NEMA 4X or better will be the requirement for our enclosure and Helicon was also concerned about corrosive chemicals splashing onto the enclosure. This type of standardized enclosure provides protection against everything we need. NEMA 12 may be relevant later on if Helicon decides to change locations to an area where windblown debris is of high concern. Enclosures can several ratings so taking on any extra ones can only be a positive so long as cost does not become an issue.

Table of NEMA Stepper Motor Standards

NEMA Standard	Length
NEMA 8	20mm
NEMA 11	28mm
NEMA 17	42mm
NEMA 24	60mm
NEMA 34	85mm

Table 4: NEMA Stepper Motor Standards

Table 4 shows several stepper motor sizes we considered. These standards are put in place in order to make replacing the motor easier since a NEMA 17 motor will fit into a mount another NEMA 17 motor was on. A smaller motor is normally not as powerful while a larger motor is typically heavier. There are exceptions to these rules but we chose to go with the middle ground of NEMA 17 as we have the space for it for our design and it should provide ample strength for our purposes.

4.1.6 UCF Health and Safety Standards

Helicon Chemical Company resides within a UCF affiliated business incubation space, therefore our design must also take into consideration UCF Health and Safety Standards. There are many standards within UCF's code of standards such as radiation, laser, workplace safety. Fortunately for us, the type of work that Helicon Chemical Company is involved in does not involve radiation and lasers, therefore these health and safety concerns can be ignored. In addition, due to the fact that we will not be operating the automated pump device that we have been tasked to create and the fact that we are not employed by Helicon Chemical Company, there is only so much we can do on our end to ensure that workplace safety is met. This involves specific development of our PCB enclosure as it houses all of the potentially dangerous electronics. The specifics of what we are going to do to meet those safety concerns is elaborated upon further in section 4.1.7 proceeding this section.

Unlike UCF workplace safety, UCF fire safety concerns must be acknowledged and taken into consideration when designing our final product. Despite the fact that the environment that our pump automation cart will reside in is already properly situated for any fire or workplace issue, we must also attempt to mitigate all possibility of fire hazards within our product. The specifics of UCF's fire safety policy involves many facets, most of which do not apply to Helicon Chemical Company or our device. These include fire safety regarding forest fire and other outdoor fire safety

precautions. In our case, the only two precautions to note are emergency protocol regarding fire safety (this is already in practice at Helicon Chemical Company in their emergency fire safety protocol) and NFPA accordance. The NFPA is the National Fire Protection Association and so long as our electronics are thoroughly tested and operate within their given operating conditions (found in electronic specific datasheets) then we will have been in accordance. To avoid any complications and potential hazards in design, we will be prototyping our design from the arduino MKR motor carrier and MKR zero boards which have been thoroughly tested and vetted by arduino, as they are consumer available and ready products. Just in case, we tested the products even more as a precaution. The testing of our electronics and more details can be found in section 7 project prototype testing plan.

4.1.7 Design Impact of UCF Health and Safety Standards

The main impacts of these standards are enclosure and testing related. In terms of enclosure, to avoid any electronic hazards from directly interfering with the chemicals that are involved in our system and the operator of the system, we must design an enclosure that houses all our open electronics. In addition, this design must be electrically and thermally stable so that in the event that our electronics were to overheat, spark, or break the enclosure will prevent any further damage to our system by blocking off direct contact to the chemicals and the user. The actual design prototype of our enclosure can be found in section 5 project hardware and software design details.

4.2 Realistic Design Constraints

The following sections describe the many different realistic constraints that we have to comply with in order for the outcome of our project to be a success. These constraints are built upon three key factors, the projects actual requirement specifications, engineering specifications, and Helicon's specific requirements. All of these different constraints work in conjunction to both make the project more challenging, but also help meet the exact design, operational, functionality specific goals that we and Helicon have set. These constraints lie in the many specifications we have set, but also in many realistic factors such as:

- Economic
- Time
- Environmental
- Social
- Political
- Ethical
- Health
- Safety
- Manufacturing
- Sustainability

4.2.1 Economic and Time Constraints

In terms of Economic constraints, this project's goal is to successfully design and build a working automated pump controller for Helicon's current pump system for around $800. Given that, currently, our project's budget is around $1200, this gives us room for potential problems that arise during our prototyping, testing, and building phases. When it comes down to price of parts, it really comes down to what parts Helicon is comfortable using, as this project is to be made customized for them. We could include customized components into the system like a customized flow sensor, pressure sensor, and temperature sensor to increase the controller's capabilities in their system, or we could choose to only include the sensors we need; The requirements of the project may change, not by a drastic amount, and we may have to include components that we did not initially account for. That is why we have a $400 buffer in our budget in case we ever needed to add additional components and requirements to our project. Another aspect of budgeting that we and Helicon must consider whether or not it would be more economically viable to just purchase sensors that are already calibrated, accurate, and precise or to have us individually build these sensors and hand calibrate them ourselves. Either one of these options are viable, but it comes down whether or not the parts we buy will be a right fit for Helicon's custom system or not. It is important to note that, we may be able to order custom parts and sensors from certain companies that allow it, but this should only remain as a forethought as doing this too much could eat into our budget and leave us with no money to spare in case something went wrong in senior design 2. A final thought to consider for economic constraints is that if we were to buy customized highly calibrated sensors that fit into our project, would they even be worth the cost? This decision is one the be made alongside Helicon in a discussion of their needs vs. wants. Helicon may not even need a flow sensor that measures flow rate and flow amount or a highly accurate and precise ultrasonic flowmeter. A normal flow meter that measures flow rate with a degree of accuracy of 0.1% might be good enough for them as they don't need, or it wouldn't be financially advisable to get a highly advanced industry topping device.

In terms of time constraints this project starts in first couple weeks of Senior Design 1 and ends in the final few weeks of Senior Design 2. The overall time frame of this project is around 8 months give or take a few weeks (September 2019 to April 2020). Starting from the beginning of September to the end of November, the goal is to have a base level design, finish all appropriate documentation (aside from documentation regarding testing, as this will be done in the future during Senior Design 2), and ordering parts. The month of December will serve a review month to make sure we have met all the requirements to jump straight into Senior Design 2 with no hassle. This month could also serve as a jumpstart into prototyping the project if we are ahead of schedule. The months of January to the end of April serve as the time where we finish prototyping, finalize a design, build our project, and implement it within Helicon's system. We are very fortunate that the team at Helicon who are serving guidance for this project are on a very similar schedule to us and are willing to help us at any stage of this project. A more in depth and elaborate timeline of this project with tasks in chronological order can be found in the Milestones section of this report.

4.2.2 Environmental, Health, and Safety constraints

As previously mentioned in the section 4.1.3 on OSHA standards, environmental, health, and safety constraints are very important to us in designing, testing, and building this project. In terms of

Environmental constraints, on our part, there are few to worry about as all the major concerns and constraints are to be taken care of by Helicon themselves. It is important to note that we as designers are building this system with guidance from a Helicon employee as a lot of the information regarding the uses, what chemicals are going to be in the system, etc. are proprietary Helicon information and because of this most of the severe Environmental, Health, and Safety constraints, according to Helicon. Some of these concerns that Helicon will account for are:

- Environmental and Health issues regarding Toxic and Hazardous Materials/ substances
- Protective equipment when using our device
- Fire protection when using our device
- A stable environment for this project in both off and on states
- Occupational Health and Environmental precautions
- A clean and safe workspace for this project to be in
- Exit routes and emergency planning regarding our project

Even though Helicon will cover these major environmental, safety and health concerns regarding our project, there are some very important constraints and requirements that Helicon requires from us in order to ensure that all environmental, health and safety concerns regarding the project are accounted for. Some of these constraints/requirements that we need in our project is a robust alarm system (along with all necessary sensors), emergency shutoff protocol, and an electrically and thermally sound product.

One of the major requirements that is needed due to the environmental, health, and safety constraints is a robust alarm system with all the proper sensors to notify the pump controller. This is a fairly common requirement in any device or system in which failure could be of paramount importance. In our meetings with Helicon, they have emphasized that an alarm system and sensor combination is not only important to have it is necessary to meet safety standards, as this project will deal with hazardous and toxic chemicals. In addition, Helicon has notified us that the key things that could go wrong in the system is flow problems (specifically not being able to properly control chemicals), temperature stability, and pressure stability. To properly account for all these systems, we need to create software that, when triggered by a specific sensor will trigger its respective alarm. For example, if there is a temperature fluctuation outside of our desired operation range, a flag would be raised in our code and a physical LED would flash on our controller to notify the operator of the system and would trigger an automatic shutoff protocol. To properly implement this, we need multiple sensors (flow, temperature, and pressure) that are enclosed in some protective PFE or PTFE plastics to have a layer of separation from the system. These sensors will be connected to our controller and will have a range of operation and anything not within that range will trigger the LEDS and the automatic shutoff system. More information regarding this system are in the Project Hardware and Software Design Details.

In conjunction with a robust alarm system, we need an automatic shutoff protocol for the system for emergency situations. This protocol will be made possible with the automation done to the valves through software-controlled stepper motors (specifically quadrature encoding; for more information regarding software-controlled stepper motors, see Project Hardware and software Design Details). To generalize the protocol, upon detection of a fault in the system (through the robust alarm system mentioned previously) the stepper motor will immediately shut off the pump

and close all the valves to avoid any movement or operation within the system. The air pump will be turned on automatically after this process is complete. Then the valves will open incrementally to allow the air pump to pump air through the pipes and valves to clean the system. The excess and unwanted chemicals in the system will be run off into a proper waste container to be disposed of. Upon prototyping this protocol with our software, we will make necessary changes where need be, but generally this type of emergency shutoff protocol is what we strive to make.

The last and most straightforward environmental, health, and safety concern we must account for is having our system electrically and thermally secure and isolated from the chemicals. The main part of our system that will house the many electronic components is the main pump controller. This will house the PCB, the heading connections, all other electrical components. The PCB will be electrically tested for faults and other possible wiring or routing issues through the JTAG (see 4.1.1.4 section on JTAG standard). In addition to this the physical enclosure housing the PCB and electrical components will be kept separate from the actual pump system on a nearby pushcart. This will ensure that the dangerous components are kept a reasonable distance away from the pump without causing a loss in efficiency. In addition, the enclosure of the electronics will be made out of durable and isolating plastic to avoid any electrical and thermal interference. Lastly the wires connecting our controller to the pump system will all be bundled together neatly to be manageable and as a general safety precaution.

4.2.3 Ethical, Social, and Political constraints

Some more constraints that we must take into consideration are those related to ethical and political issues. It can be argued that when designing a project of this caliber the constraints that affect the project the most are those related to monetary, health/safety, and ethical/political. In terms of ethical constraints, the ones that affect the project the most are related to Helicon Chemical Company. Due to the nature of the work and products that are associated with Helicon Chemical Company, some information regarding this cannot be shared in any shape or form to the public. In addition to this, because we are not employed by Helicon, but rather seen as student sponsorship it is within our ethical responsibility to keep all proprietary and sensitive information regarding Helicon Chemical Company and this pump automation project under strict secrecy. Although taking ethical responsibility for this information is generally seen as a constraint, as we cannot have outside forces help us with the specifics of this project, it can also be seen as a strength as the mutual trust between us and Helicon Chemical Company will ensure a strong working relationship. Lastly, in terms of ethical constraints, is to maintain a top level of work integrity that is of the same level as our academic integrity. It goes without saying but any level of plagiarism, copying, or stealing information, work, or ideas from outside sources is not only heavily unethical, but in most cases illegal. Due to the fact that this project is in association with Helicon Chemical Company any unethical behavior that we indulge in could affect them, which we cannot afford to happen.

In terms of political constraints, there are specific regulations that we must abide by due to the industry that Helicon Chemical Company is in. These specific political regulations that we must abide by are set to restrict and control technologies related to the defense industry. In general the brunt of these constraints and standards apply only to Helicon as they are the ones involved in the sales of these technologies, but nevertheless we must take these constraints into consideration as we have chosen to work with them for this automated chemical pump system. Of these constraints

the main ones that apply to us is distribution of information regarding their technologies, as we are technically in the realm of contract work. Helicon has done more than enough work on their end to ensure that only essential information regarding their pre-existing pump system is divulged to us. Information such as physical composition, pump, and valves have been divulged to us as these are safe to share. The specifics of the chemicals have not and do not need to be divulged to us as these can be seen as a violation of their non-disclosure policy.

These strict governmental controls set and enforced by the United States ensures that the safety of the United States and people, but there is still an international concern for safety regarding technologies that could be seen as harmful or dangerous. This is why the United States has a regulatory board, ITAR (International Traffic in Arms Regulations), that we must strictly abide by in order to not undergo any political or legal actions against us.

4.2.4 Manufacturing and Sustainability constraints

The last set of constraints that we must take into account are those related to the process of building our project during Senior Design 2 and making sure our project is sustainable in the long term for Helicon Chemical Company. One of the benefits that comes with designing this project specifically for Helicon Chemical Company is that we do not need to worry about large-scale production of this product. Typically there are many important manufacturing details to account for like, shipping, assembly line production, labor costs, etc., but we only have to abide by those constraints that apply to custom made products. Some of those manufacturing constraints that we must account for include fabrication, prototyping, calibration/testing, purchasing supplies, and assembly. These manufacturing steps are layed out with a rough scheduling estimate in our discussions on milestones (8.1). For this project specifically, fabrication, prototyping, purchasing supplies, and assembly will be done with the strict documentation and methodology as per the IEEE standards mentioned previously. In terms of calibration, this step of the manufacturing process is key for us as our project relies heavily on our sensors (temperature, flow, and pressure). Above every other component in this project, we must ensure that in the manufacturing process our sensors are calibrated to measure data accurately and precisely. To properly calibrate a sensor, we must use a master sensor that know is accurate (for flow sensing purposes, the industry standard is using high-quality and established sensors like the KROHNE sensor/meter combination in Figure X), then compare and program our sensors to match that of the master sensor.

When designing a project like this that is specific to a company of purpose, we must ensure that the project is sustainable. This means that our project needs to be built to last and with solutions to possible future issues in mind. One of these issues that could occur in the future are those related to cleaning of the valves and piping chambers. It is possible that the valves and piping in our system may be contaminated, clogged, or just in need of replacement in the future and because of this, we must design our sensor and stepper motor subsystem so that they can be removed and added back to the system easily. In addition to this we need to make sure all of our materials like piping and enclosure plastics are built to last and are not in need of constant replacement. To ensure that our materials are made with high-quality, durable, and resistant material, we will be using either PTFE or PFA plastics only for piping and enclosure as these are the industry standard and work excellently in projects similar to this.

5. Project Hardware and Software Design Details

The following section describes, in detail, our project's hardware and software design. In our design phase we created each of the following sections with full intention to not only function as freestanding modules, but also function integrating into the system as a whole. However, it's important for us to understand that our designs may need revision as we move forward with the implementation phase.

The figure on the following page shows our project design overlaid with the current Helicon system. As noted in the legend, Helicon's system is in blue, our system is in black. The main components/parts of our project are represented in 'block' form to show the flow of hardware and software feedback loops that our project will contain. The components represented in Black, showing our system, will be outlined individually in this section in the following pages. The blue section, showing Helicon's system, shows the following major components, their purpose, and the interaction of each component will have with our system:

Reagent Tanks - Stores the chemicals for the reaction. Our system as it's currently proposed will not interact directly with these components.

Argon Tank/Vacuum Pump - Helicon's reagents are air sensitive. As air will be introduced after each reaction during cleaning, the system will need to be purged with inert gas to remove reactive elements. The Argon tank and vacuum pump will perform this function. Our system as it's currently proposed will not interact directly with these components.

Peristaltic Pumps - Drives the reagents/inert gas through the system. Our system will interact with these pumps through a DB-25 interface to remotely control flow rate, as well as receive feedback signals generated by the pump hardware.

Manual Ball Valves - The valves allow for both the reaction and the purging to use the same pumps and tubing. They switch between the two lines and isolate them from each other. Our system will modify the manual valves to be automated and receive feedback from the added components.

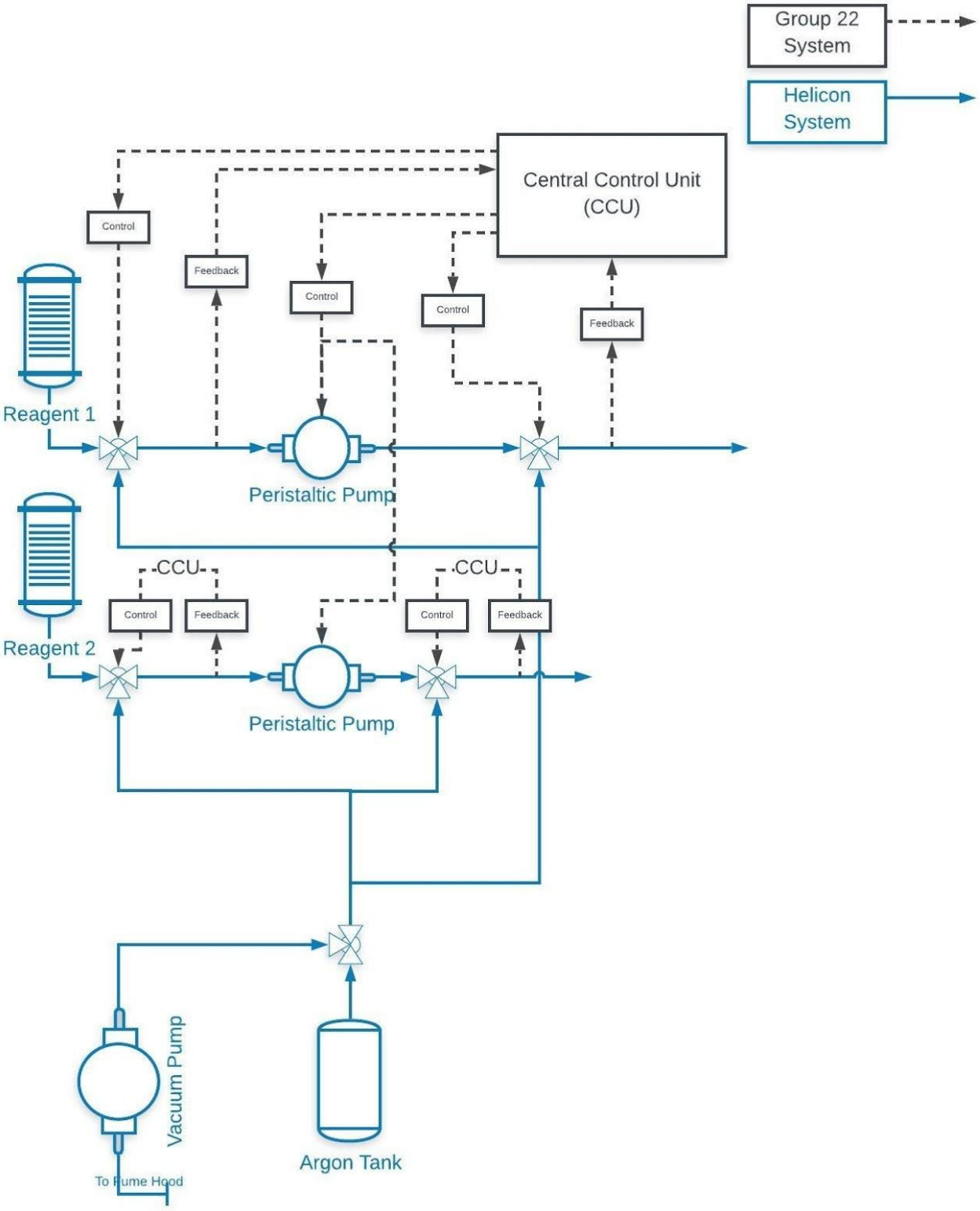

Figure 10 - Diagram of Helicon's Pump system (Blue), and our Automation System (Black)

5.1 Initial Design Architecture and Related Diagrams

Our system will be integrated into Helicon's existing pump system. Figure 10 above shows how the two systems and their integration. The two systems will interface physically in the case of all valve controllers and sensors, and electrically in the case of the pumps. Figure 11 below shows Helicon's pump system and the cart that will hold it along with our project. Important to note that this system is still a work in process for Helicon, and not all components are present at the time of this picture.

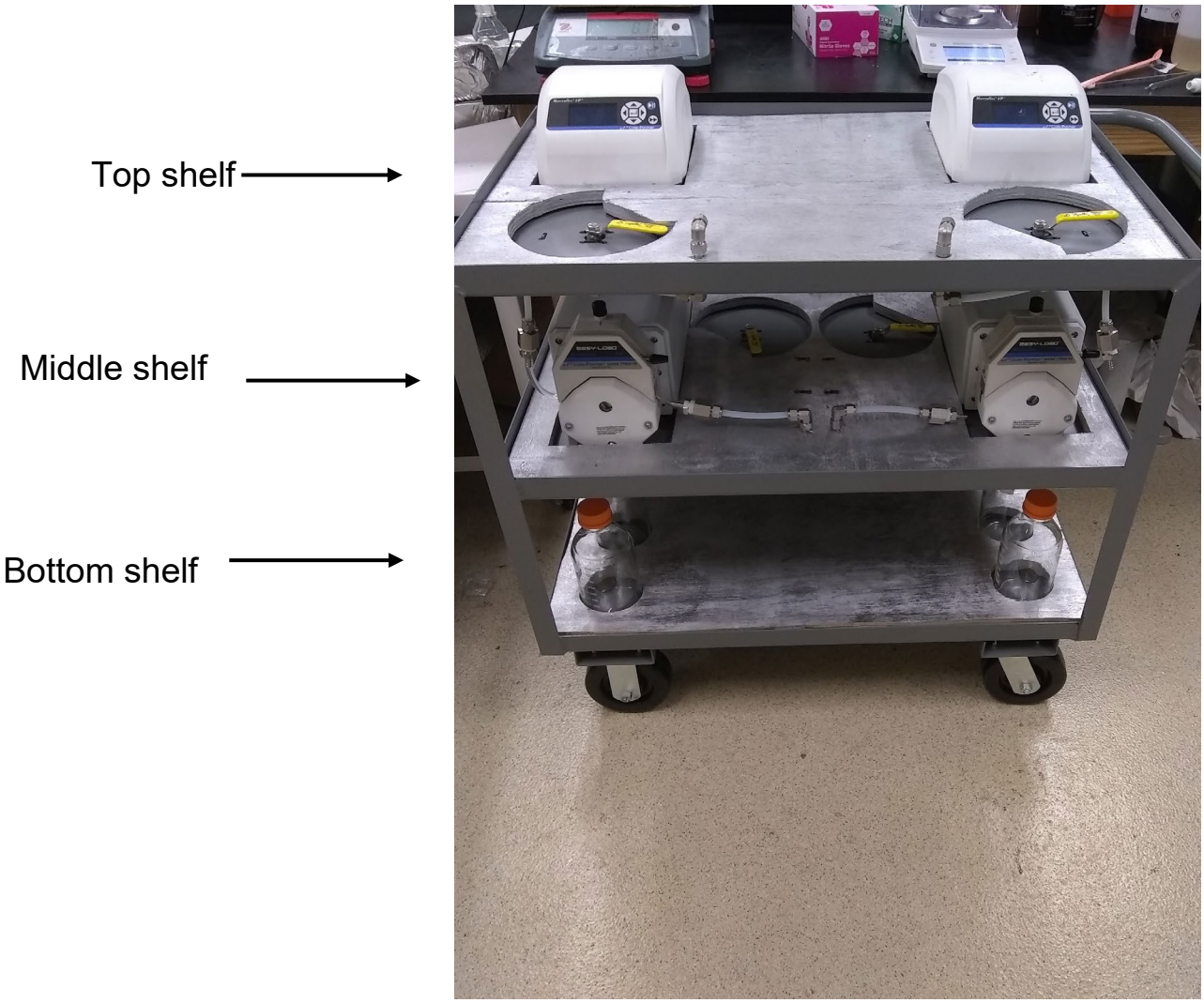

Figure 11 - Helicon Pump System: incomplete and before our system integration

The Top Shelf of the cart houses the two peristaltic pump controllers, two mechanical ball valves mounted underneath, accessible via through holes, and two through holes for tubing with elbow joints attached. These joints will eventually connect tubing leading to gas and vacuum lines for purging the system. The top shelf is where our control box will be housed, in between the two elbow joints. We have 11" of lateral space to work with, up to 24" of width, and up to a foot of height.

The only hard constraint being 11” of lateral space. Figure 12 below gives a better visual representation of the space available to us.

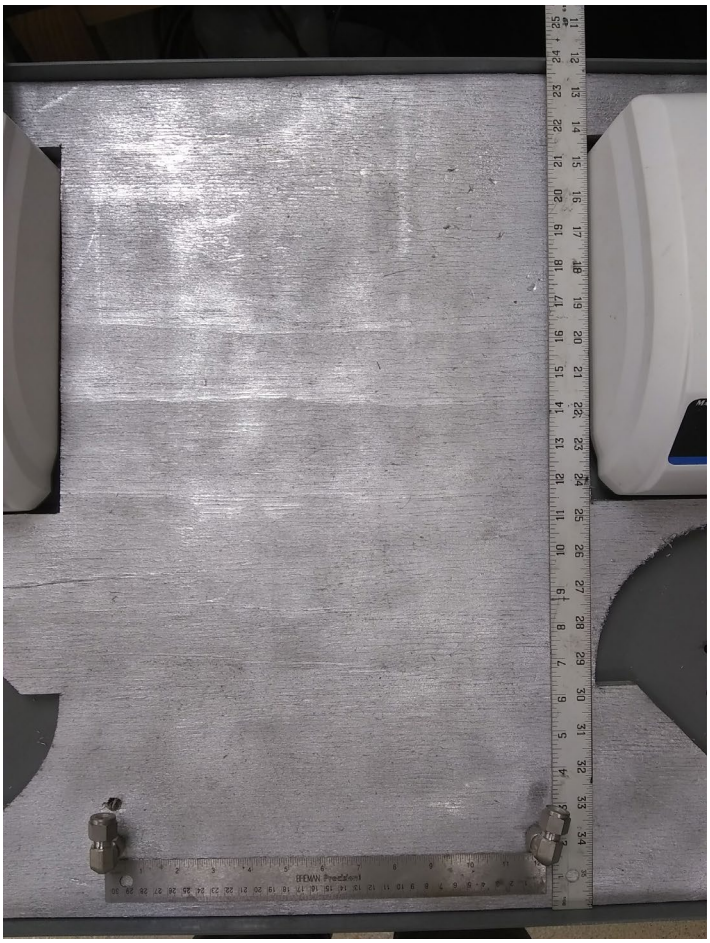

Figure 12 - Available space for our CCU housing

It’s important to note that the lateral space is straddled by two hose fittings that will have tubing connected when the system is operational. Also while we have roughly 24’’ of width, we are only planning to use between ½ to ⅔ of that space in case it’s needed in the future. On the top shelf we will also mount two motors over the ball valves for automatic control. This process is outlined in more detail in section 5.3.2.2.

The Middle Shelf of the cart is where the two peristaltic pumps are located. Along with the pumps are two ball valves, and vias for tubing. Our current design will interface with the middle shelf by mounting the other pair of motors over these ball valves. A via will be drilled through the top shelf to pass wires from the control box to these motors. The location of ball valves on the middle shelf introduces a notable design constraint. While it should be plenty of room, we only have about 11 inches of vertical clearance. See figure 13 below.

Figure 13 - Vertical clearance for middle shelf ball valves

Current designs for our motor mounts, shown in section 5.3.2.2, are within this height constraint. It is yet to be determined if this design is sufficient however. Shown also in the figure above is one of the two Masterflex peristaltic pumps we will be working with. Interfacing with these pumps is outlined extensively in sections 5.3.2.1 and 5.5.3.3.

The Bottom Shelf holds six media bottles for reagent access at the beginning of the process, and reagent storage after the process. Our current design does not call for any interaction with the bottom shelf.

The only components of Helicon's system not located on the cart are gas cylinders and a vacuum pump. The gas is for purging the lines of air before the system is run, and the vacuum pump is for pulling the gas through the lines. Any air in the system could react with the reagents and cause inconsistencies in Helicon's process.

The initial design for our system calls for a six control loops all connected to a central control unit (CCU). Four control loops will manage flow direction, and two will manage flow rate. For all six

control loops, the controller will be a subsection of the central control unit. In the four flow direction loops, the control element will be either a solenoid valve or a mechanical ball valve modified with a servo motor. The feedback mechanism will be a flow sensor or a servo position sensor (built into the servo). For the two flow rate control loops the control element is the peristaltic pumps, and the feedback mechanism will be a flow sensor.

On top of the CCU and control loops will be a human interface, which includes visual interface, controls, and an alert system. While there will be overlap in the functionality of each, the CCU, control loops, and user interface will each be considered a distinct subsystem for the purposes of this section.

5.1.1 Physical Component Diagrams

For the Physical Component Diagrams section we will focus on the physical component mounting and enclosure design. The diagrams are not necessarily to scale and instead serve as a guide to how our end product should look. With that said, they have still been designed with consideration to the dimensions outlined in the previous section, and will fit within these constraints.

The physical enclosure housing the CCU and most other components will be the “face” of our project, so unlike most other design considerations, we will approach this design a bit more artistically. Although it’s also important to note that as the Helicon team interacts with the physical enclosure, the artistic appeal will diminish as quickly as the Florida winter.

Considering the height of the cart and the average height of the technicians at Helicon. It is best if we integrate our user interface into the top section of the physical enclosure. This is the most ergonomic of our options. A diagram of the top view of the enclosure is shown below in figure 14.

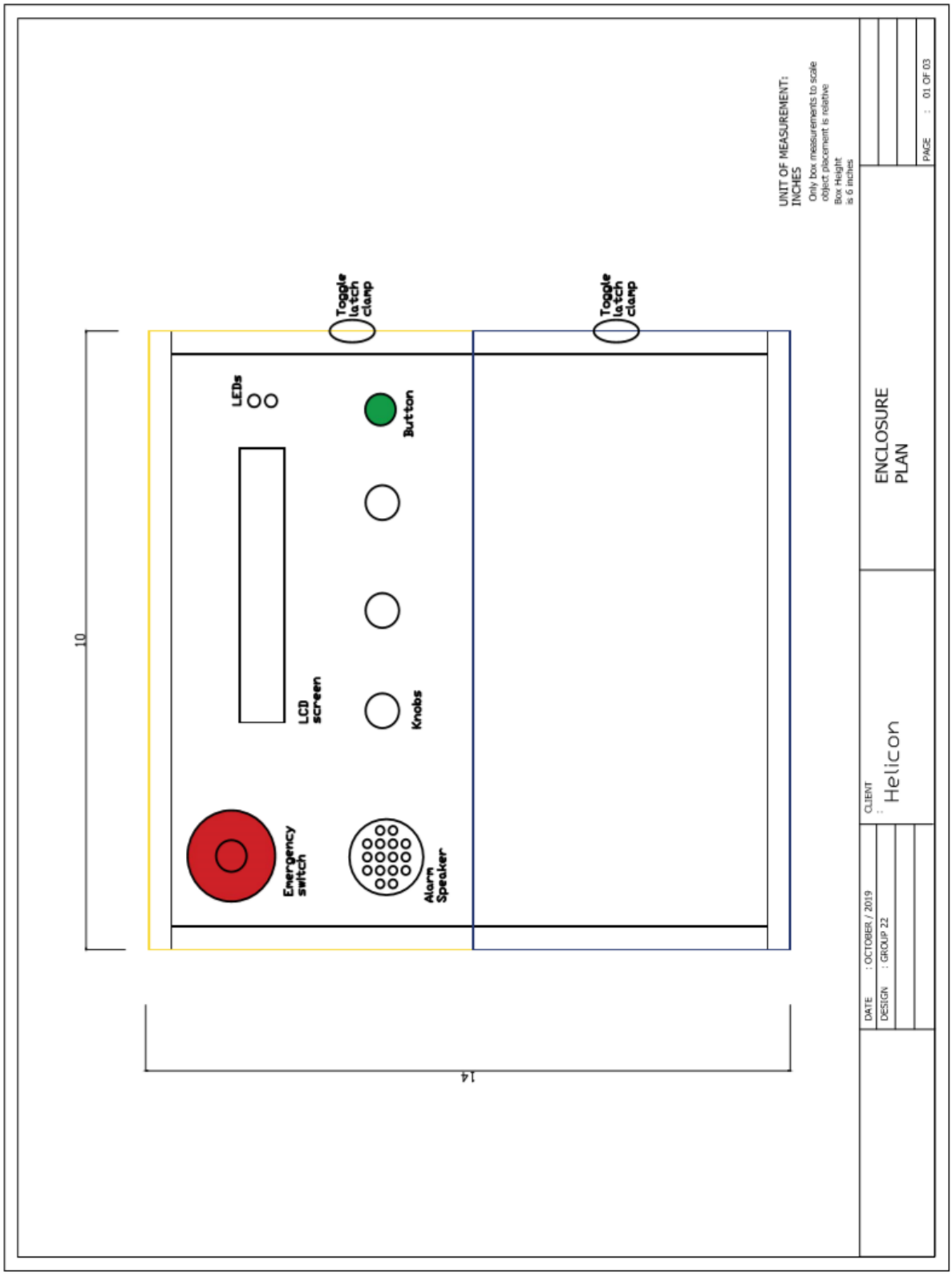

Figure 14 - Top View of Enclosure

In Figure 14 above we have a top view of the Enclosure and its component layout. The width cannot exceed 10 inches so it may fit snugly in the cart space available to us. We have room to work with as far as height and depth are concerned but 14 inches in height and 6 inches in depth should be more than enough. We originally were going to make a wooden enclosure and have our enclosure door split into 2 separate doors as shown by the colors yellow and blue above. Each door will be openable separately and will each close using a toggle latch clamp. This is to allow easy access to the PCB but more importantly its SD card without having to see the circuitry going on in the top half of the enclosure. However This was before learning of the many standards placed by the NEC. Because of this we decided to simply purchase an enclosure that complied with the Nema standards that Helicon asked for. This did not impact the design of the enclosure very much as the measurements still fit other than the fact that there will only be one door now instead of two.

On said top half we will have 3 knobs, a push button, and an LCD screen for our User interface. Two LED's, one red to signify a problem has occurred and one green to signify everything is working as intended, will be to the right of our LCD screen.An alarm speaker will work in tandem with the red LED to alert anyone within earshot that the system has encountered a problem and requires a technician's attention to solve. Lastly we have an Emergency switch to manually shut off power in case of an emergency.

The door will have some minimalist wiring canals running along the side and between the components. We still want easy access to the PCB so making sure that upon opening the enclosure there isn't a tangled mess of wires is an important aspect.

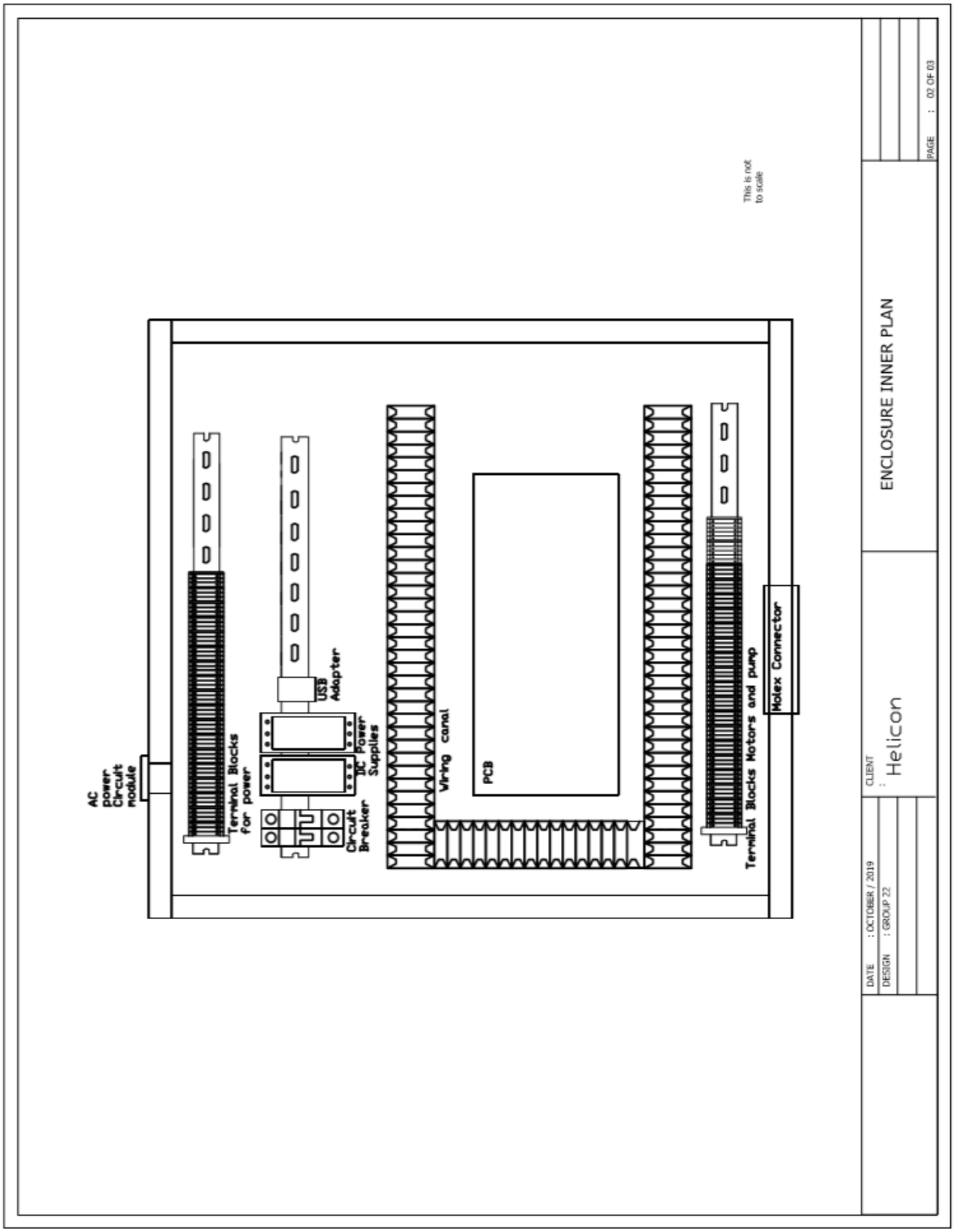

Figure 15 - Inside of Enclosure

Figure 15 above shows a rough positioning of the various physical components in our enclosure. Three din rails will be installed on the back-plane of our enclosure. Two of these will be for terminal block lines, one located at the top and another at the bottom. The amount of terminal blocks on the diagram above is for possible future upgrades. We will only need 6 on the top for organizing power. One for the live line one for neutral and one for ground. These will all be connected to the corresponding pins on the AC power circuit module. The other three will be there for temporary use or for testing or any other unforeseeable occurrence. If required we can save space by omitting that din rail entirely and instead connecting the AC power module directly to our circuit breaker or having the terminal blocks on the same dail rin row as the circuit breaker. The bottom side will require significantly more terminal blocks. A molex connector will bring in sixteen connections from pumps and motors. Each one will be routed to terminal blocks to keep our wiring organized. There might also be an unforeseen necessity for extra blocks down here so four more will be added.

The last din rail will be to hold any component which is not our PCB or that is not mounted on the door. We will have a circuit breaker and two AC to DC power supplies. One power supply will supply 12V and 3.3Amps of power for the motors. The other will Supply 5V and 2 Amps of power for our PCB.
Surrounding our PCB on three sides will be a line of wiring canals to give the design an organized and professional look.

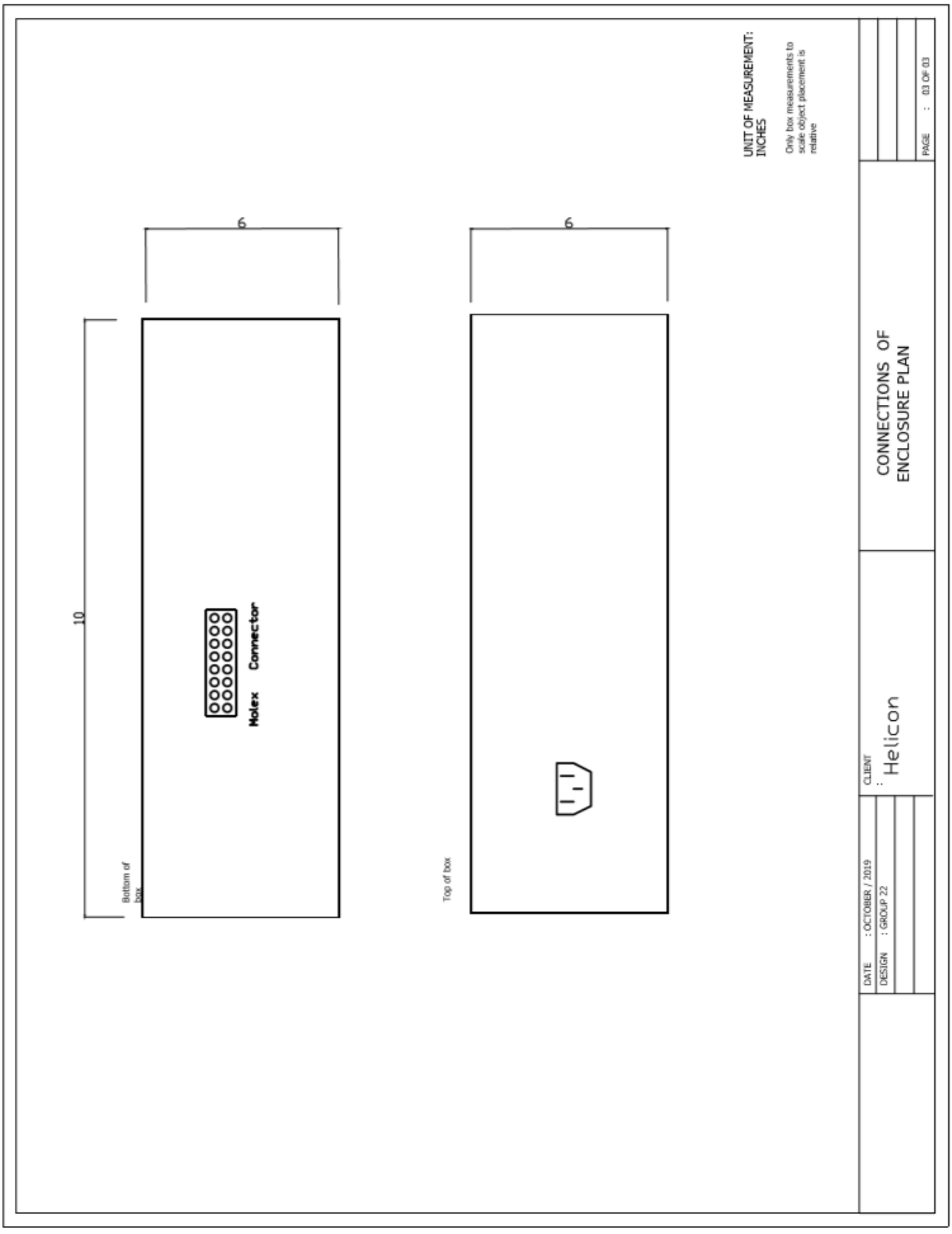

Figure 16 - Bottom and Top sides of enclosure

Figure 16 above simply shows the dimensions of the top and bottom sides of the enclosure as well as the relative placement of the sixteen pin molex connector and the AC power circuit module.

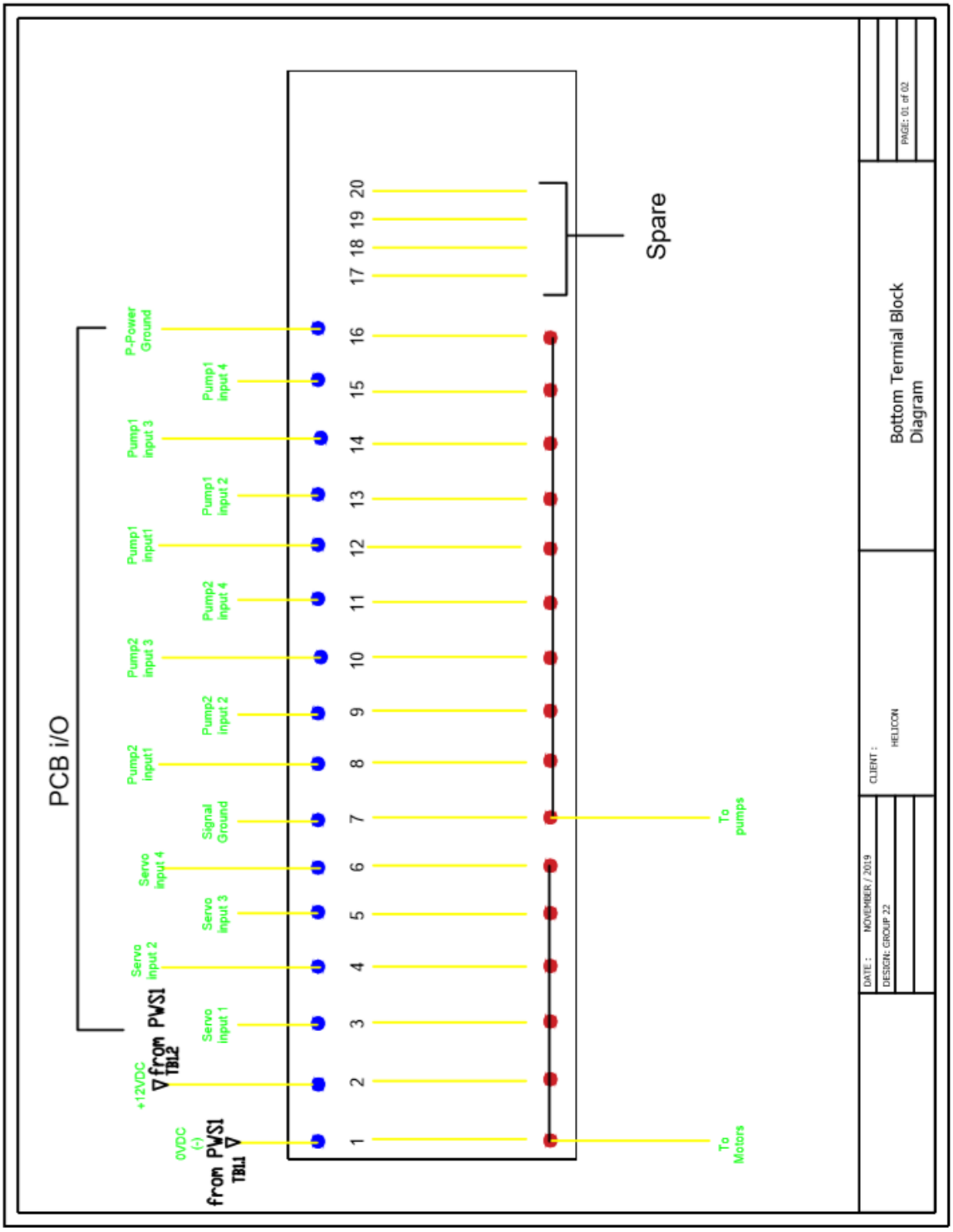

Figure 17 - Bottom Terminal Block Diagrams

From figure 17 we can see how the bottom Terminal blocks will be organized. All the connections coming into our enclosure will be coming in through our Molex connector. We will be using TB as the naming convention for the terminal blocks with TB1.1 being the first on the bottom row and TB2.1 being the first in the top row of terminal blocks.

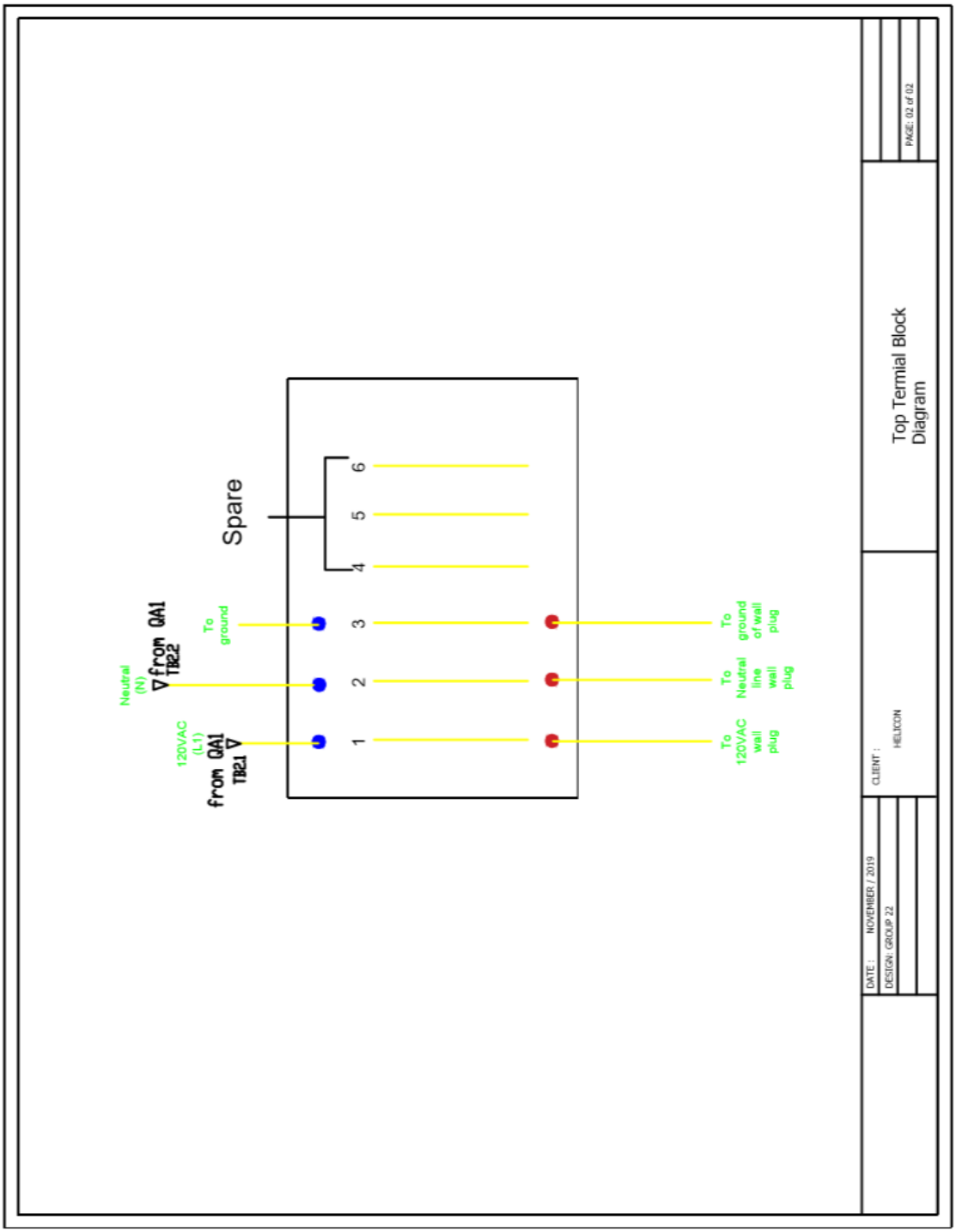

Figure 18 - Top Terminal Block Diagrams

In Figure 18 the top terminal block wiring diagram shows how we plan to organize the power coming into our enclosure from our AC power supply module. The three spare Blocks can be dropped for space if necessary.

5.2 First Subsystem: Central Control Unit

At the center of our design is the central control unit, or CCU. The CCU is the brain of our project and the most important subsystem, and all functions will be embedded on a single PCB. It will be the source of most of the design, prototyping, and implementation challenges we encounter during this project, and thus must be our primary focus. Physically the CCU will be contained in a box along with the Human-Machine interface, referred to as the control box. This box will either be fitted on the top shelf of Helicon's pump system cart, or a stand will be purchased to allow the control box to stand alone. If fitted on the pump system cart, the control box will need to be removable, to protect the sensitive electronics during cleaning.

The CCU can be broken further into subsystems depending on function. These subsystems, or modules, will be power, controller, and data logging.

5.2.1 CCU: Power Module

For the power module on our CCU, we will have component specific power regulations. For example, anything that is a non-core unit (this includes anything that is not our SAMD21 Cortex-M0+ 32bit low power ARM MCU and our ATSAMD11 MCU) will have as set of capacitors and voltage regulators that is shared among them to accomodate for the various voltages. For our CCU the main voltage outputs that our microcontrollers use are 3V, 3.3V, and 5V. Shown below in Figure 19 and Figure 20 are the power modules we will be using.

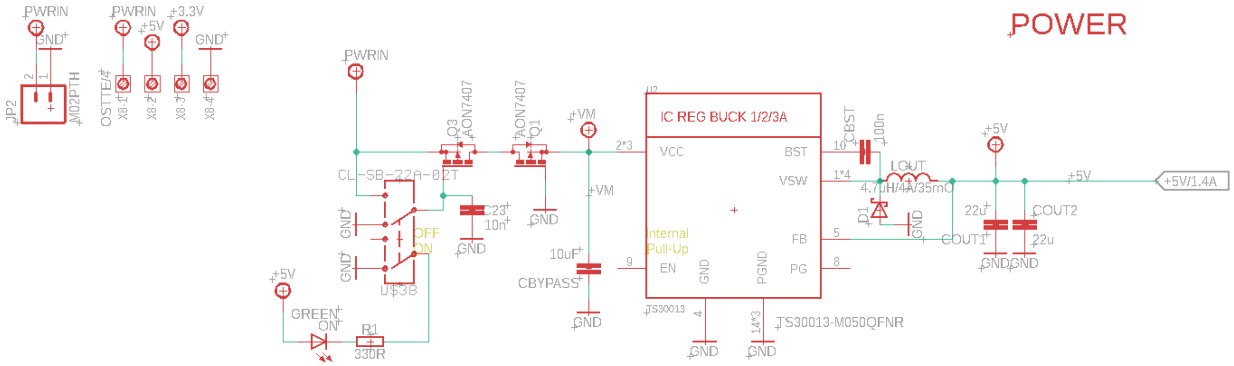

Figure 19: Power Module mkr motor carrier

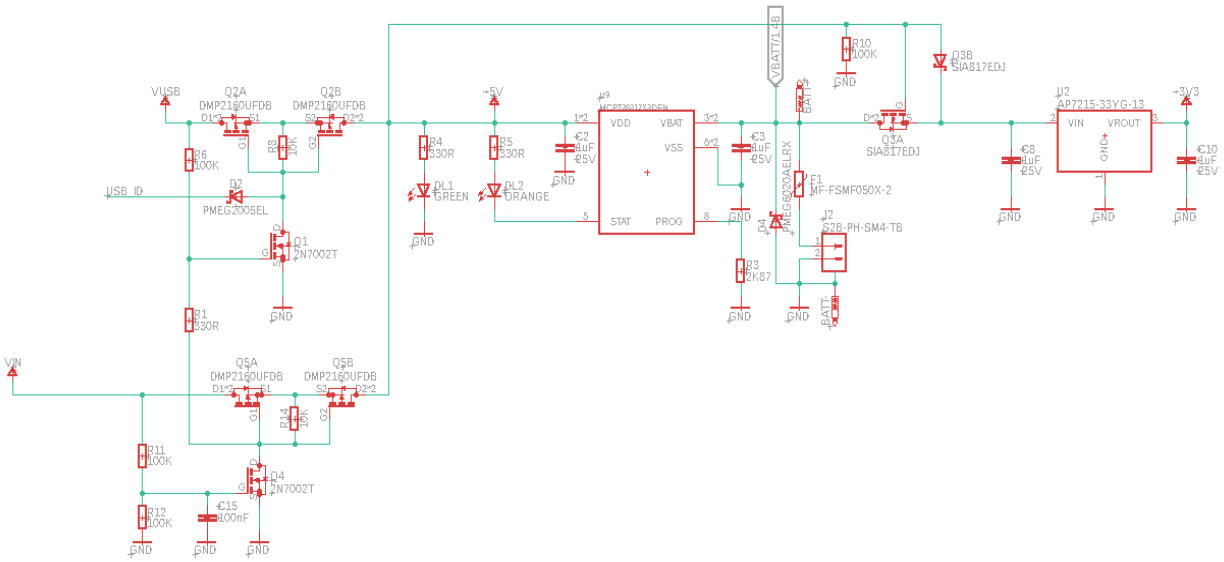

Figure 20: Power Module mkr zero

Shown in figure 20, the power module of the MKR motor carrier, the voltage regulator specifically is a DC-DC switching regulator that has a fixed 5V input and output but outputs the three different currents, 1, 2, and 3 Amps. These large currents must be properly managed by this module as these currents will be used to power the large drivers for the stepper motors and servos used in our design. These drivers and their corresponding maximum currents are tabularized in the following section 5.2.2. In addition the voltage must be properly translated from 3.3V to 5V for the communication between the ATSAMD11 microcontroller and the servo connectors. This is done using the LSF0108 bidirectional voltage level translator as a buffer between the two voltages. Figure 21 shows the the the LSF0108 as it is wired and labeled in our schematic.

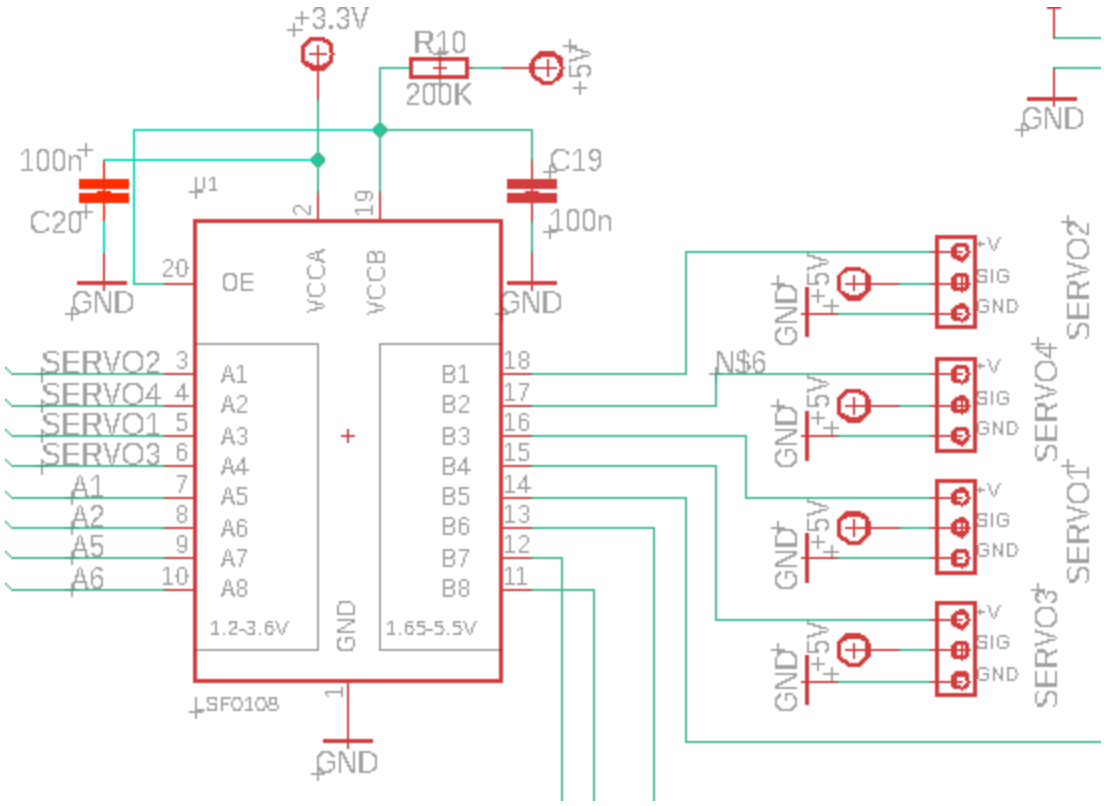

Figure 21 - voltage translator schematic

The final aspect to the power module is how we will be receiving power from our 5V micro-USB input. The way that this input is designed in our arduino prototype boards is with connection to a computer in mind (with limited use case disconnected, as a battery socket is present). This conversion with require changing from 120 volts at 60 hertz to 5VDC. While this is entirely doable on our end, the general and most basic way to accomplish this is through the use of a transformer. This is one aspect of power circuitry that we cannot afford to have mistakes with, and given the fact that none of us are experts in this field, we have chosen to simply purchase a reliable 120 volt, 60 Hz to 5VDC adaptor. Purchasing this adaptor not only decreases the power module's probability of failure, but also takes some design stress off of us which allows focusing on a more manageable part of the power module which is our DC-DC converters.

5.2.2 CCU: Processor Module

For the processor module, current design contains two microcontrollers, the SAMD21 Cortex-M0+ 32 low power ARM MCU and the ATSAMD11 (ARM Cortex-M0+ processor). The SAMD21 ARM processor will be the microcontroller that is actually being programed to, as this controller has access to both the onboard data collection/logging module and the motor control libraries used by the ATS ARM microcontroller. The ATS ARM microcontroller + processor must still be included in our design as a middle-man between the SAMD21 MCU and the stepper motors, servos, and sensors used in our design as they can possibly require a large current input. The ATSAMD11 will fix this problem for us as the drivers compatible for this MCU (the MC33926 and DRV8871)

have a max current of 5 Amps peak (heat sink dependent) and 3 Amps peak (current sense resistor dependent) as shown in table 4 below.

Table 4 - Basic on-board driver specifications

Driver	**Max Current**	**Additional details**
MC33926	5 Amps Peak	Value may be RMS current depending on heat sink
DRV8871	3 Amps Peak	Limited by current sense resistor

In addition, our decision of using these chips over other chips that work in a similar fashion comes from the fact that both of these chips are used in Arduino boards (the MKR Motor Carrier and the MKR ZERO), which are heavily used by professionals and enthusiasts, meaning that there are many resources online regarding usage, tips, and troubleshooting for them. The figure 22 and 23 below is the schematic of both the SAMD21 Cortex-M0+ 32 low power ARM MCU and the ATSAMD11 chips. It is to be noted that these schematic footprints do not show all pins in use as most of these pins are wired either to ground or to their respective signal planes. The routed connections shown below are the non-signal plane connections that have specified logic attributed to them. Another aspect to note is that in figure 23, the thick blue wire connection is the bus connecting the main pins from the ATSAMD11 to the voltage translator shown in section 5.21.

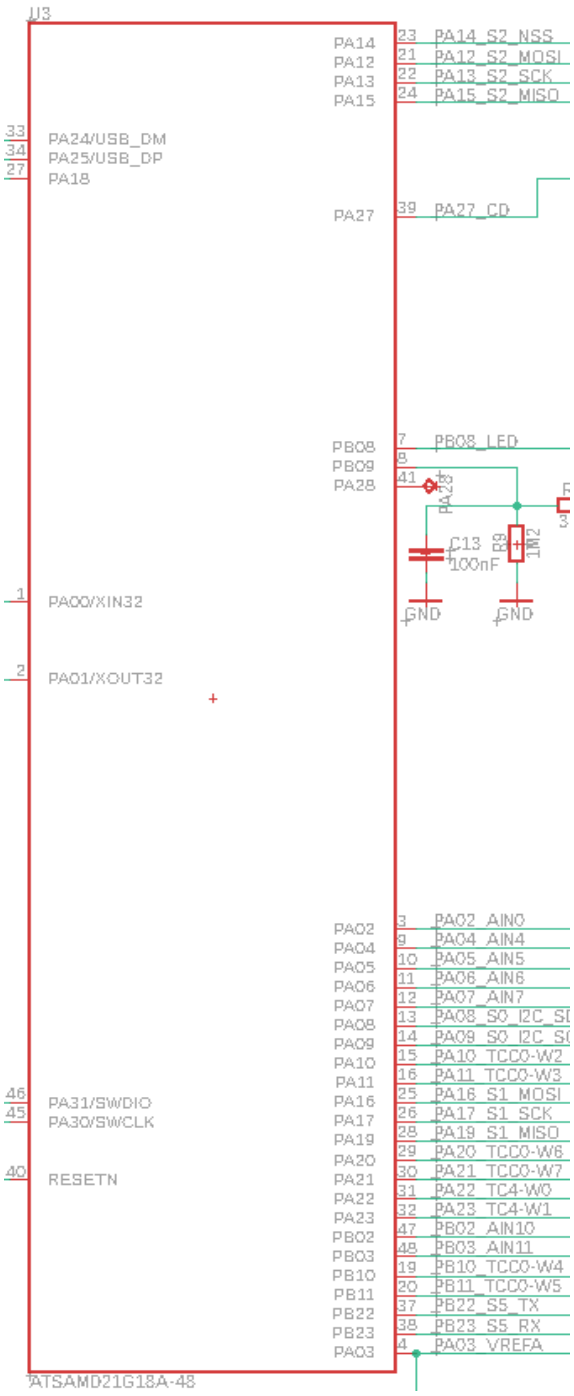

Figure 22 - SAMD21 Chip Schematic

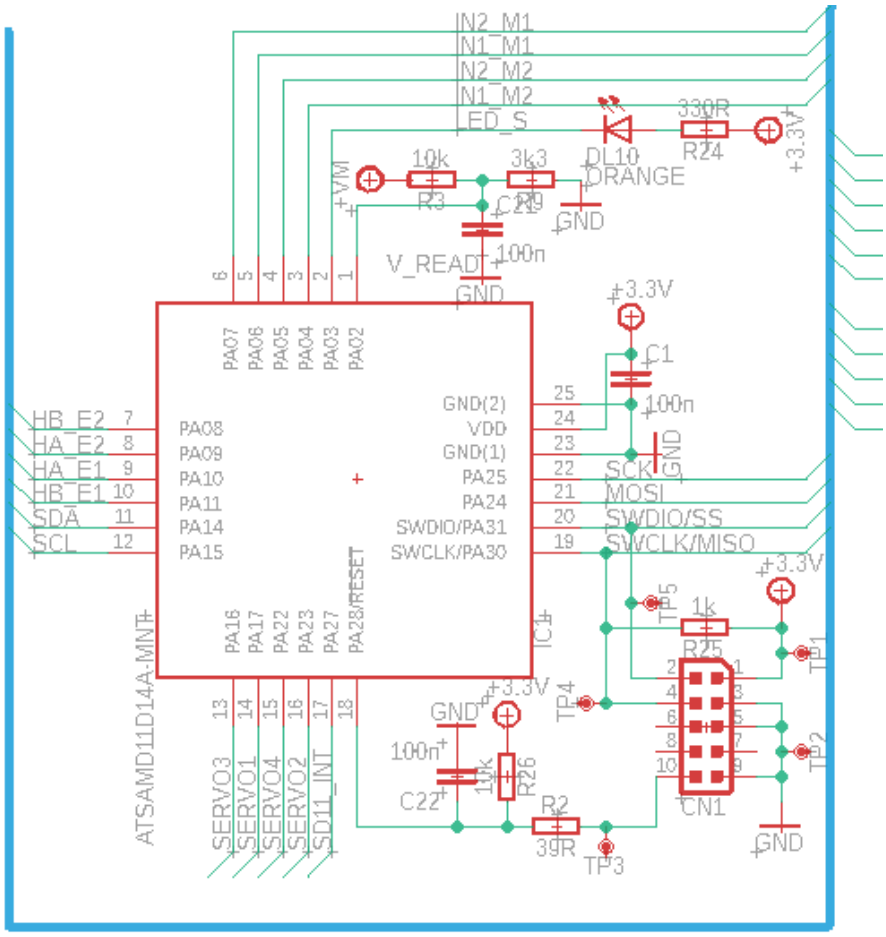

Figure 23 - ATSAMD11 Chip Schematic

As mentioned earlier we are implementing a dual microcontroller system. An extremely important factor to consider is how these two microcontrollers are going to communicate with each other. This will be done in a manner where the SAMD21 processor will be the master and the ATSAMD11 will be the slave processor. The reasoning for the choice of master and slave microcontroller is based upon the specifications of each chip, and seeing as the SAMD21 chip has larger program memory size (256 KB vs 16KB) and SRAM (32KB vs 4KB) it will be the master controller while the ATSAMD11 will be the slave microcontroller. There are many ways that this communication between master and slave can be done such as I2C, SPI, UART, and USB, but in this case we will be using an atypical form of communication, parallel port communication. Parallel port communication entails sending multiple bits of data at once using multiple data lines. In our case there are 28 data lines, most of them being basic data lines such as voltage values, reset, RX, TX, MOSI (Master Out Slave In), MISO (Master In Slave Out), etc. The following figures 24 and 25 represent the 28 pin parallel port connection of each microcontroller (SAMD21 and ATSAMD11 respectively).

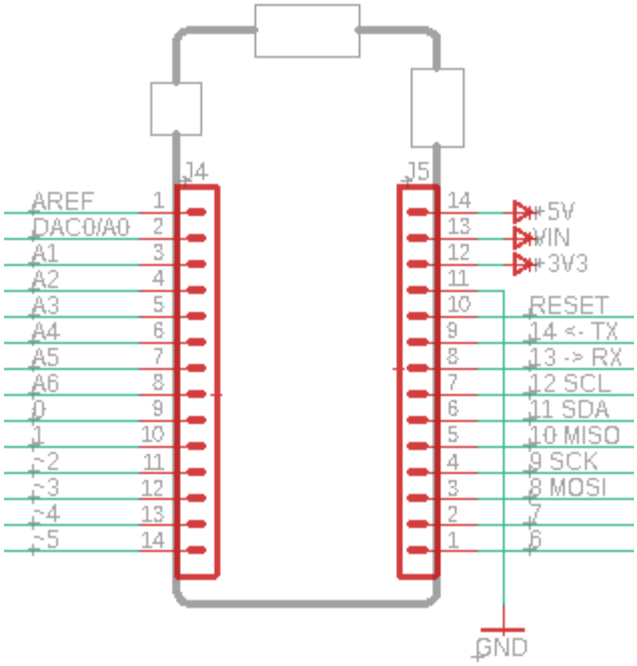

Figure 24- 28 pin port for SAMD21 (Master)

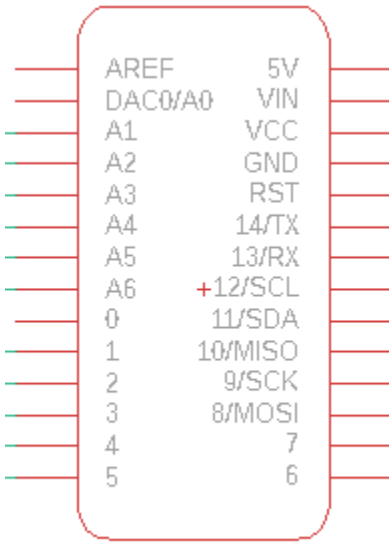

Figure 25 - 28 pin port for ATSAMD11 (Slave)

These two 28 pin connectors can be left unconnected in the PCB and can be physically connected through the pin headings, or can be wired together through the routing of the PCB. For our purpose we will be skipping over the option of physically connecting pin headers and will be wiring the two ports together in the schematic to avoid usage of an extra connector and/or multiple PCBs for each of our microcontrollers.

5.2.3 CCU: Data Logging Module

During the runtime of the chemical profile, the system will be making periodic appendages to a data log file. It will be adding all sensor data, as well as time stamps for valve opening and closing. Depending on the size of the file, this log will either be stored in the internal memory of the microcontroller itself or external memory accessed by the microcontroller. The user will need to be able to access this information, and while the traditional way to do transfers of data between systems would be through USB, our selected system will use a micro SD card reader connected to our SAMD21 (master) microcontroller. Figure 26 represents the schematic wiring diagram of our micro SD card reader/connector with appropriate net labels (PA netlabels represents pins used on the SAMD21 controller).

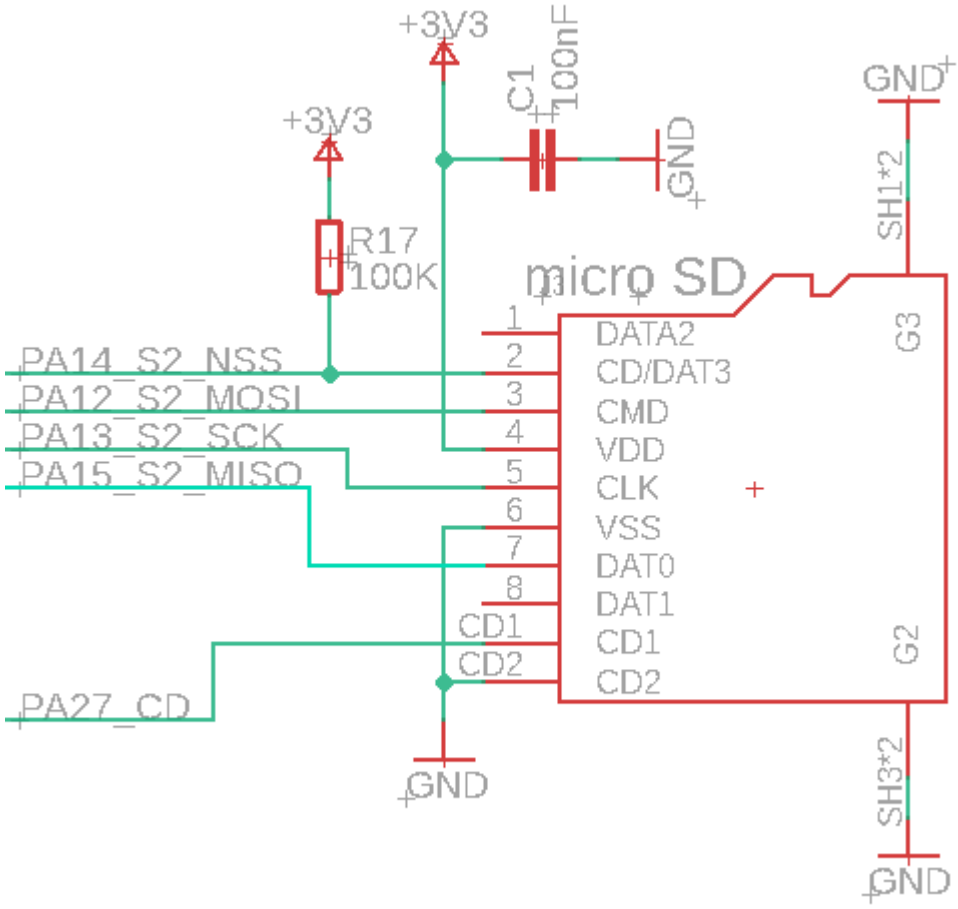

Figure 26 - Schematic for micro SD card reader/connector

5.3 Second Subsystem: Control Loops

The core mechanical function of our design are the control loops. Outlined in section 3.2.1, a control loop manages a process variable by reading the state of the variable, comparing to a set point, and making corrections to the variable as needed. Our design calls for two control loops to manage flow rate, one for each peristaltic pump, and four control loops to manage valve position, which will govern flow direction.

5.3.1 Controller

As discussed in section 3.2.1, the controller is the brains in the control loop. The controller needs to calculate the instantaneous error, the accumulated error, and the rate of change of error at any given moment. With this information, the controller will decide a corrective action to take and generate a signal to communicate that action to the control element. For the control loop subsystem, the controller module will overlap with the CCU subsystem and it's software and hardware will be part of the processor module.

If we are able to integrate temperature control, a fringe goal for this project, PID control will be necessary and the corresponding circuitry will be added to our PCB. As is stands currently, our control loops will not need that level of sophistication and the controller will exist entirely as software on the processor chip.

5.3.2 Control Elements

For the two control loops governing flow rate, the control elements are the two peristaltic pumps. The pumps are set up to be remote controlled and can be interfaced with via a DB-25 connection. Through these DB-25 connections we will interface with the pumps in the following ways:

5.3.2.1 Pump DB-25 Inputs/Outputs

Inputs used: Remote Start/Stop (Digital), 0-10V Speed Control Input (Analog)
Inputs unused: Remote CW/CCW, Remote Prime, Aux in

The peristaltic pump inputs work with current sinking outputs, through NPN transistors with open collectors, or with contract closures to earth ground. To start and run the pumps, a continuous low signal is sent to the Remote Start/Stop input. The pump flow rate is controlled via analog input signals. There are two options: 4-20 mA analog signal, with 4 mA being Stop and 20 mA Full Speed, or a 0-10 V analog signal, with 0 V being Stop and 10 V Full Speed. Both options offer 10 bit resolution. For this project we will use the voltage inputs and outputs.

Although our current design does not plan to use all inputs, it's important to understand them in case they are required in the future. For example, if Helicon determines their system requires the pumps to run in both directions, this can be done with the Remote CW/CCW (Clockwise/Counter Clockwise) input. This input can be pulled to active low to run the pumps counterclockwise. The pump will slow to a controlled stop before changing direction. Secondly, Helicon has talked about needing to prime their system by running each pump individually until the two reagents are right

on the edge of the mixer. This can be done with the Remote Prime input by sending a continuous active low signal.

Outputs used: 0-10V Speed Feedback Output (Analog), General Alarm Output (Digital)
Outputs unused: COM (Motor Running), Tach Output, Local Remote Indicator

The peristaltic pump uses built in feedback mechanisms to measure the flow rate and converts this to an analog output signal. Similar to the input signal, the pump outputs both a current and a voltage in the same ranges with 10 bit resolution. For the purposes of this project this satisfies the feedback section of the flow rate control loop. Along with the speed feedback, we will use the general alarm output signal, which will communicate when the internal pump circuit detects any errors. This way our system will be in tune with the pumps and our controller can respond to any alarms by notifying the Helicon team via our alert system, or by initiation emergency shutdown.

It's important to note that the Alarm Output uses an open NPN collector. This output gives a "low impedance" state at earth ground and is essentially floating when in "high impedance" state. What this means is that extra care must be taken in wiring this output to avoid damage to external equipment. It is recommended that we use a current limiting resistor to avoid current surges at the low impedance state.

As with the inputs, our current design does not plan to use all the pump outputs. Still, it's important to understand them in case they are required in the future. The COM output shows whether or not the motor is running, which can add redundancy to our feedback system. The Tachometer output shows the RPM of the pumps, which Helicon could find useful for data collection. The Local Remote Indicator output gives a visual indication that the pumps are being configured/operated remotely. This could potentially be useful in the future if Helicon implements a procedure where the pumps are controlled both manually and remotely.

With the inputs and outputs outlined, it's also important to note that the MKRZero uses 5V logic, which will be incompatible with the 0-10V Pump motor speed input/output signals without modifications. Since both sides use 10bit conversion, the resulting circuitry is simplified. When the speed control signal runs from MKRZero ------> Pump, the 0-5V signal will need to be amplified by two to access the full range of speeds offered by the pump. The amplifier has been designed and is shown below in figure 27.

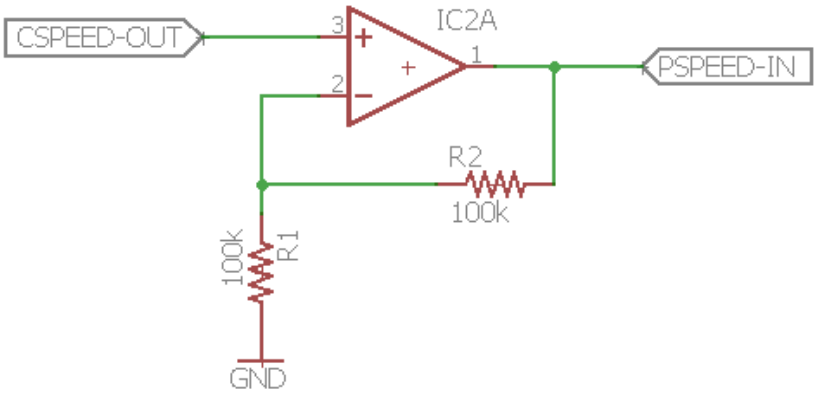

Figure 27 - Non-Inverting Amplifier between controller and pump speed input signal

The non-inverting op amp in the above figure operates on the following formula:

$$\frac{PSPEED - IN}{CSPEED - OUT} = 1 + \frac{R2}{R1}$$

So in order to amplify the controller output signal by two, R1 and R2 must be the same. Also note in this amplification step, we only care about voltage amplification, so R1 and R2 can be high such that the system draws low power.
When the feedback signal comes back from pump ------> MKZero, the 10V max will be too high for the MK's ADCs. This signal will have to be brought back down in a low power voltage divider. The circuit has been designed and is shown below in figure 28.

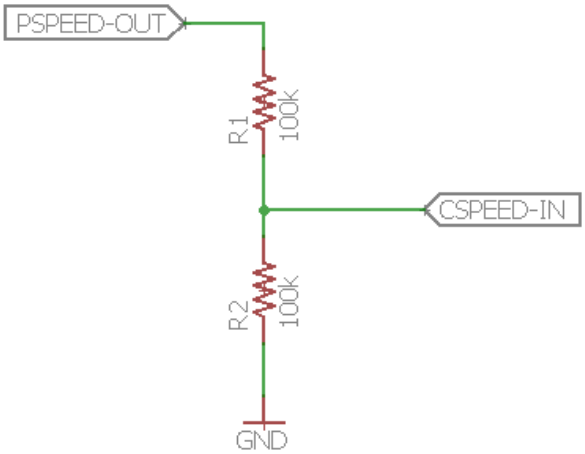

Figure 28 - Simple resistor divider between the Pump Speed Output and Controller Speed Input Signals

In the above figure, running the pump speed output signal through two resistors of equal value will split the voltage in half for the controller speed input signal.

5.3.2.2 Ball Valve Automated Control

For the four loops governing flow direction. The control elements are manually controlled ball valves that switch flow direction between two inputs to a single output. The ball valve will be modified so that a servo can be attached.

The motor will need to be high torque, and will need to work every time. We will need to purchase or machine an adapter to couple the servo and the rectangular knob of the ball valve. Also the servo must be securely mounted over the valve to ensure efficient and consistent force transfer. This mounting solution will need to be semi-permanent, and the servo must be removable. If our system fails at any time, Helicon must be able to convert their process back to manual operation.

To this end we have selected a NEMA 17 stepper motor with a 27:1 planetary gearbox allowing for up to 300 Newton Meter torque. Without the gearbox we would only have about 22Ncm torque which is much lower than what is required for this design. This stepper motor has a shaft diameter of six millimeters while the valve (motor interface) is 10mm so the coupling would have to account for this difference. In addition, in order to comply with our non-invasive approach we will need a coupling that is relatively easy to detach .

Figure 29 - 6mm to 10mm motor coupling

Figure 29 above shows what we will use. The motor shaft and valve will be secured in place through pressure by tightening the prisoner screws on the coupling. There are two concerns with this design idea. The least troubling is whether or not we can apply enough pressure with these to make sure the two pieces it adjoins do not slip. If this is an issue we will have to look to soldering or some other solution. The other concern is whether too much pressure is applied to the valve side and ruins the threading which helicon currently uses. A solution for this would be to customize the coupling to have two flat pieces inside the coupling to clamp on to the two flat sides of the valves (shown in figure 29 further down). Although soldering comes to mind first, industrial strength glue could work just as well if not better. These ideas will be further elaborated on in the testing section of this report if we do find these problems to be prevalent in our design.

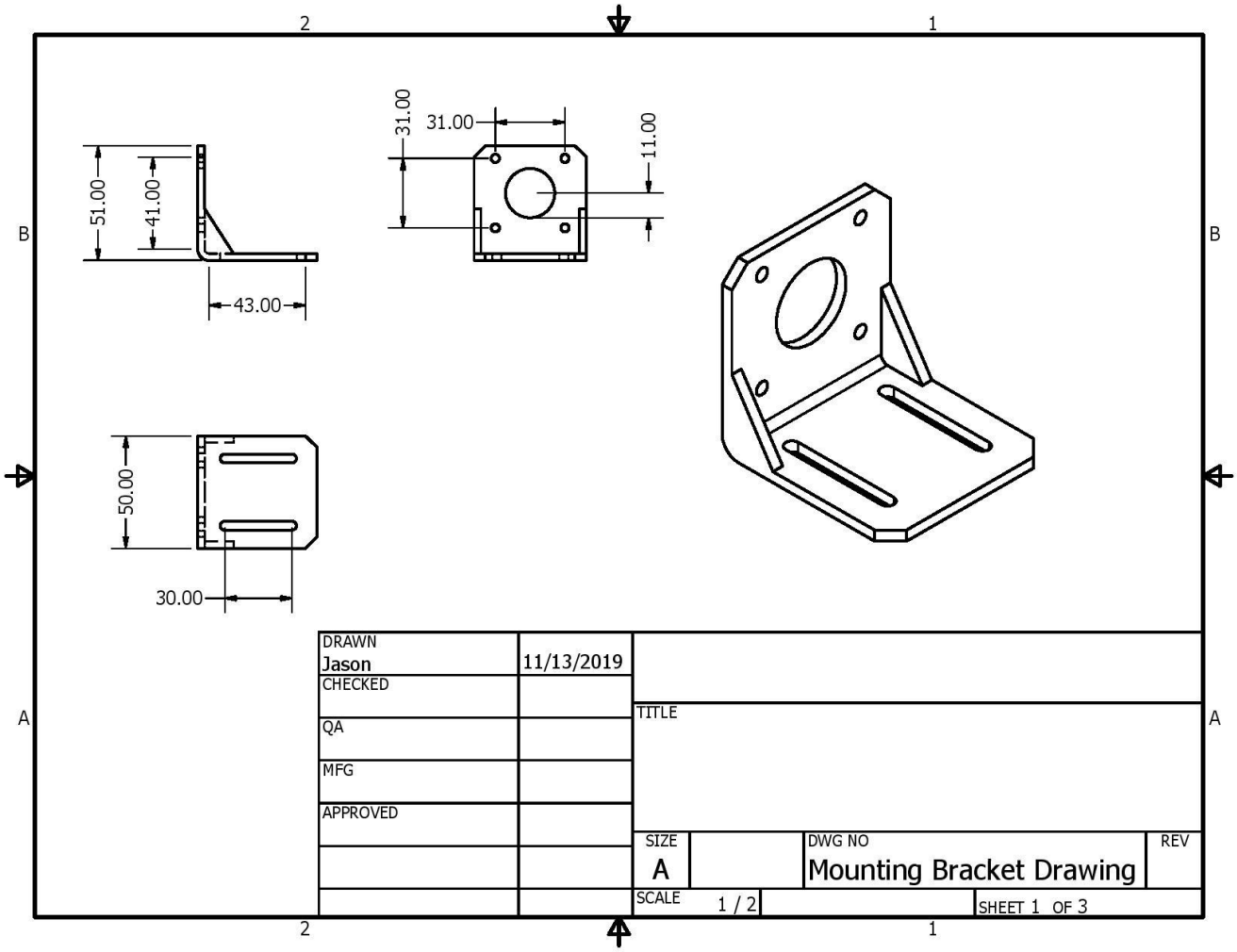

Figure 30 - Servo mounting bracket to be purchased

For mounting the motor, there are standard mounting brackets that can be purchased. In our case for our NEMA 17 stepper motor a corresponding bracket has been found and is included with our parts list. The motor bracket alone is not a sufficient solution, however, we will also need to design a custom bracket to mount the motor and motor bracket to the cart. Figure 30 above is a CAD drawing of the motor bracket. Although the bracket will be purchased, it is necessary to model it so that a proper cart bracket can be designed.

The cart bracket, connecting the motor bracket to the cart, needs to be sturdy enough to allow efficient force transfer from the motor to the ball valve, as well as semi-permanent so that the Helicon team can convert the ball valve back to manual operation if our design fails. Figure 31 on the next page shows the bracket design considerations.
Helicon's system has the ball valves mounted on the underside of a steel shelf, with the operation mechanism accessible through a through hole. Figure below shows the valve with handle for manual operation (left), and (right) with the handle removed and a small nub that our motor will interface with.

Figure 31 - Ball valve showing manual operation (left) and motor interface (right)

Due to the configuration shown above, the cart mounting bracket will need to be slightly offset from the valve's axis of rotation. The bracket can be bolted to the cart after drilling holes in the base layer to hold everything in place. With these considerations the bracket was designed in CAD software and shown in figure 32 below.

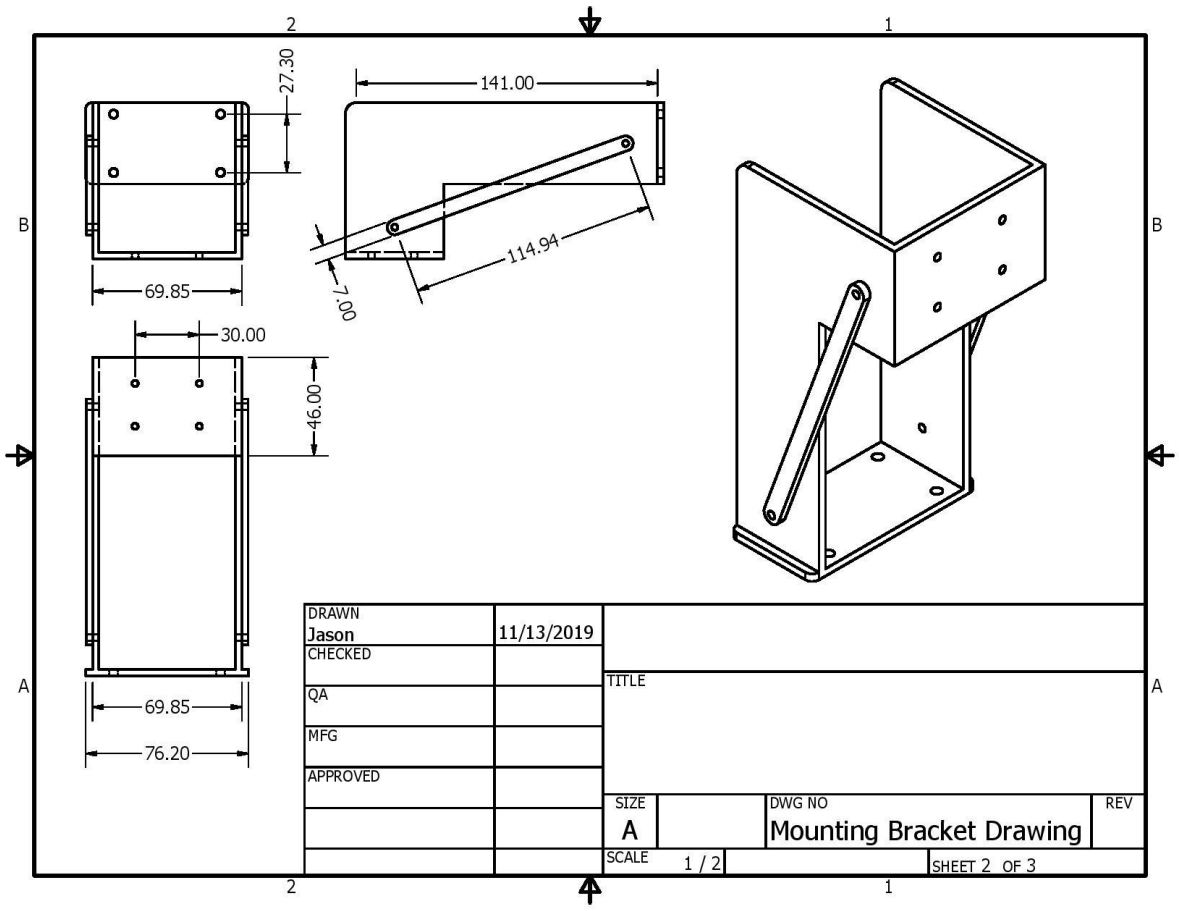

Figure 32 - CAD model of servo mount to be machined

At the moment of design, the motor was not finalized and thus it was still unsure of the final motor dimensions. So this design will be modified to accommodate these dimensions. The design can be 3D printed, or recreated with standard components. In either case we will need four of them, one for each motor.

The final assembly of the bracket mounting the four motors on the cart above the ball valves is shown on the next page in figure 33.

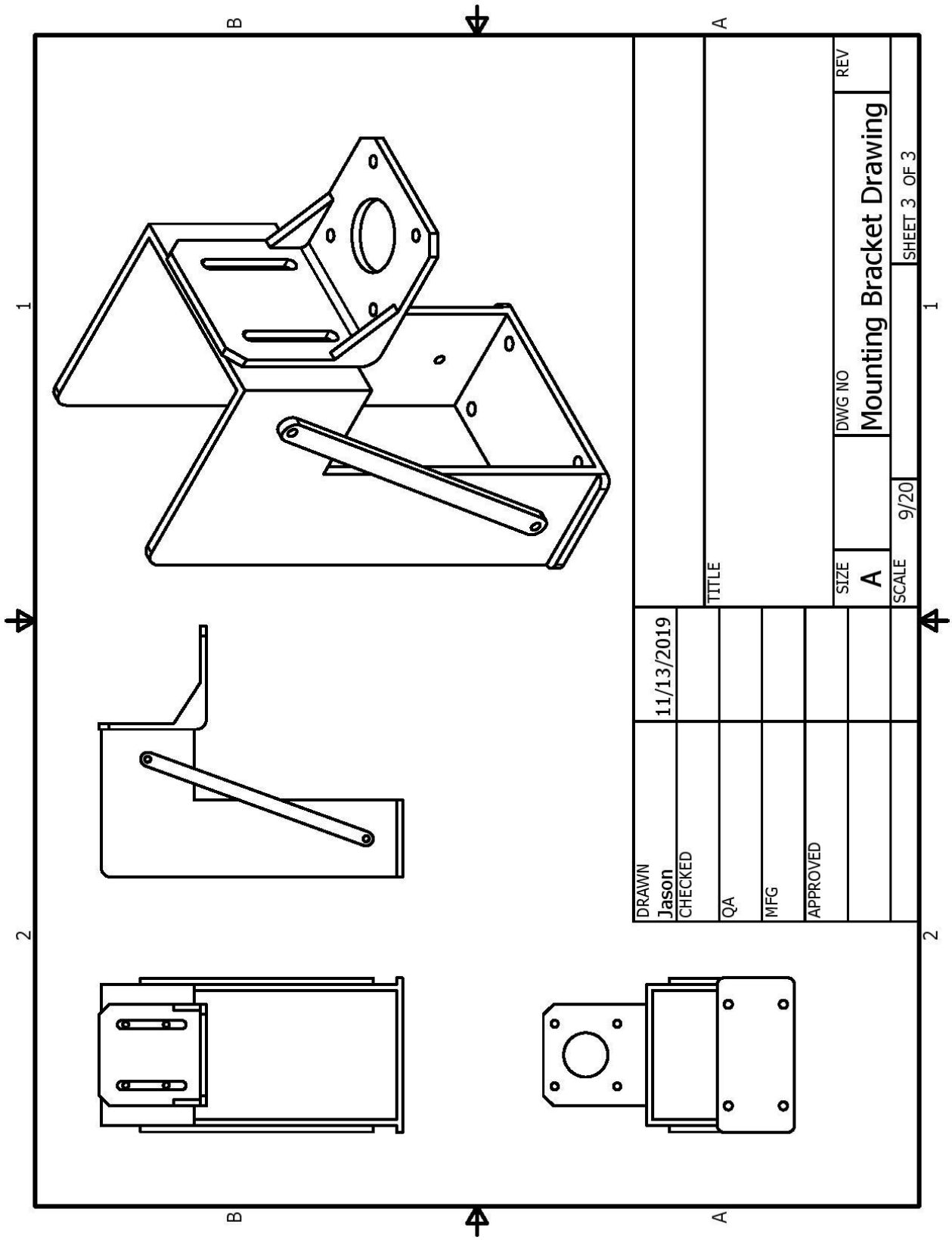

Figure 33 - Complete assembly of motor mounting bracket

5.3.3 Sensors

One serious design consideration for this project will come with the feedback mechanisms of the control loops, the sensors. Our sensors are the only part of the project that, depending on the type chosen, will come in direct contact with Helicon's reagents. This is significant for several reasons. First, installation of these sensors will require modification of Helicon's system, either by splicing chemical tubing or placing them immediately after the valves. This process will have to be done cautiously to avoid creating weaknesses in the plumbing. Second, it must be ensured that the sensors are chemically compatible with the reagents they will be in contact with. If any reaction occurs between them, the sensor can be damaged and, much more likely, Helicon's sensitive chemical process will be disrupted.

For these reasons we are currently looking at piezoelectric flow meters, explored in section 3.2.6. While parts of the sensor will still come into contact with the reagents, and thus must be chemically compatible, the sensitive elements of the sensor will be isolated. However, despite being contactless the sensor will still interact with the reagents via ultrasonic waves, and whether this will have adverse effects on the chemical process is yet to be confirmed by our TPOC.

Feedback is a critical mechanism for the control process, and will be important to us in order to verify commands from the control box have been executed successfully. However the nature of feedback involves interaction with the system. Helicon's novel process is very sensitive, and if it cannot be determined with certainty that a sensor's interaction with the chemical process will leave it unaffected, Helicon may withhold implementation of our project. For this reason we must also consider a design that does not include feedback.

Although the peristaltic pumps have a flow rate accuracy that is reliable enough to eliminate the sensor portion of their respective control loops, it is still good practice to include sensors to verify their actions and provide the feedback necessary to close our loops. However, if we run into the problem described above, the sensor can be omitted and the pumps will be controlled using an open-loop process. Similarly with the four valve control loops, each servo will have a built-in position feedback mechanism that can be used in place of sensor feedback. In this case our controller will only be able to confirm the position of the servo, which will be sufficient in 99.9% of cases. If for any reason, however, the valve opens and no material passes through when it should, our system would not catch this failure.

5.4 Third Subsystem: User Interface

The Human-Machine interface will be the "face" of our project. It will handle the interaction between our system and the Helicon technicians. For this reason, in addition to meeting technical expectations, this subsystem will need to meet a certain artistic standard. A system that is otherwise perfect will be unusable if the UI is not crisp and simple, yet also comprehensive.

The user interface can be further divided into distinct modules: display, user inputs, and alerts.

5.4.1 Display

An LCD screen will serve as the visual means of communication between the user and the hardware. It will need to inform the user of current processes underway, display pertinent sensor data and any less critical warnings, and also scroll through various chemical profiles that the user can select from. As such, the screen needs to be large enough to display all the necessary text, though it does not have to display everything at the same time. In fact, it would be a better idea to keep the display as simple and uncluttered as possible to make the system more user-friendly. The display will work in conjunction with a number of push buttons to allow the user to make choices which are then translated to the hardware.

The menu system will need to start at a place that gives the user the ability to start the current chemical profile, a quick start approach that will save time if the user is not changing between products after every batch. If the user does not wish to start the currently selected chemical profile, they must be able to choose to view other chemical profiles. From this menu phase, the user should be able to scroll forward through chemical profiles, and either make a selection, or hit cancel to be taken back to the main start screen. While a chemical process is being run, the display will show an estimated time to completion. This screen will be overwritten if there is a warning to display and the warning will not disappear until a user has acknowledged it via a push button. Warning that are displayed are not for issues that could be critical concerns, but for issues that can be addressed after the current process has finished. Figure 34 illustrates how the menu could flow and all the actual types of display images that will be required. There are no options to stop a process while it is running at this time because there is a dedicated emergency shutdown system in place that would do much the same thing.

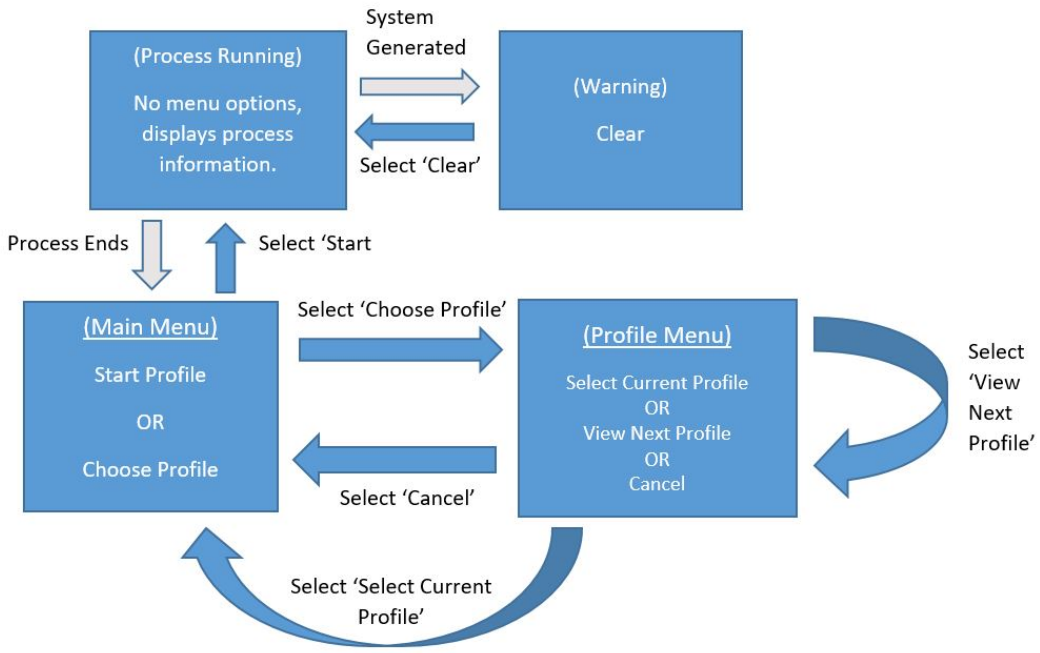

Figure 34- Decision Tree for the Display

5.4.2 User Inputs

Helicon technicians will need a straightforward and standard method of transmitting data from their brains to the brains of our system. Our approach to this is extremely important, because data transfer from human brains to computer brains is the biggest bottleneck of human processing power. A human processor (brain) can think of a question in microseconds, a computer processor (google) can answer the question in nanoseconds, but it takes several seconds to transmit that information, with your thumbs on your phone keyboard. It might as well be years from the perspective of the processors. To summarize, even the best brain-computer interfaces are horrible, and if we aren't careful with ours then working with our system will be very frustrating.

To add to our design considerations with this module, Helicon technicians work with their hands in a messy environment. It's inevitable that anything they touch consistently will get very dirty. For this reason we plan to use selection knobs, buttons, keyboard/mouse, or a combination of all in order to manage the user inputs. Depending on which methods we use, the software will need to follow the input methods. The simpler our mechanical input module is, the more sophisticated the UI software will need to be in order to cover all potential user inputs.

The first thing to add is an emergency off button likely in the form of a normally closed Red Mushroom emergency stop. These type of push buttons once pushed open the circuit to prevent electricity to continue running into the circuit. They remain pushed until twisted which releases them returning to the "on" setting. We can either have this emergency stop interact directly with the pump system, serve as an input for our CCU or both. If we have them both we can have the dangerous elements turn off and while retaining the current process data in the CCU so resetting the system has more options. Having the CCU recognize the emergency switch will also allow it to serve as another safety precaution, if for some reason the emergency switch is bypassed then we can also have our CCU turn off the rest of the system.

5.4.3 Alerts

With every engineering project, it is advised to hope for the best, but prepare for the worst. There are a multitude of scenarios where things can go wrong, with varying degrees of consequences. While we will attempt to automate our systems' response to problems, there is no substitute for human intervention. A critical module of the user interface will be a warning system to alert nearby technicians of any problems. Since our goal is to free the attention of Helicon's technicians, this warning system will likely be audio based if immediate attention is necessary. For less concerning issues we can add an indicator panel to quickly communicate error status, or display messages on the LCD screen.

5.4.4 Power Diagram

Our system will be fed 120VAC from a standard wall plug. First thing this should pass through is a circuit breaker for safety reasons. This circuit breaker is a Schneider iC65N C2A (shown below in figure 25 as QA1) which is rated for 120VAC and 10 Amps. From that circuit breaker, we connect to two power supplies to convert our AC voltage to DC. We will be powering four motors each requiring 0.4 Amps to power. So we need one conversion to 12VDC and at least 1.6 amps. We have chosen a power supply with 12V and 3.3 amps from MiWi which should be more than enough to power our motors (shown below in figure 25 as PWS1). We also need to power our PBC which requires 1.4 Amps and 5VDC so we chose a power supply that 2.4 A (also from MiWi and shown

below in figure 25 as PWS2). Not shown below is a conversion from simple wires coming from PWS2 to a USB port which our PBC will connect to for power.

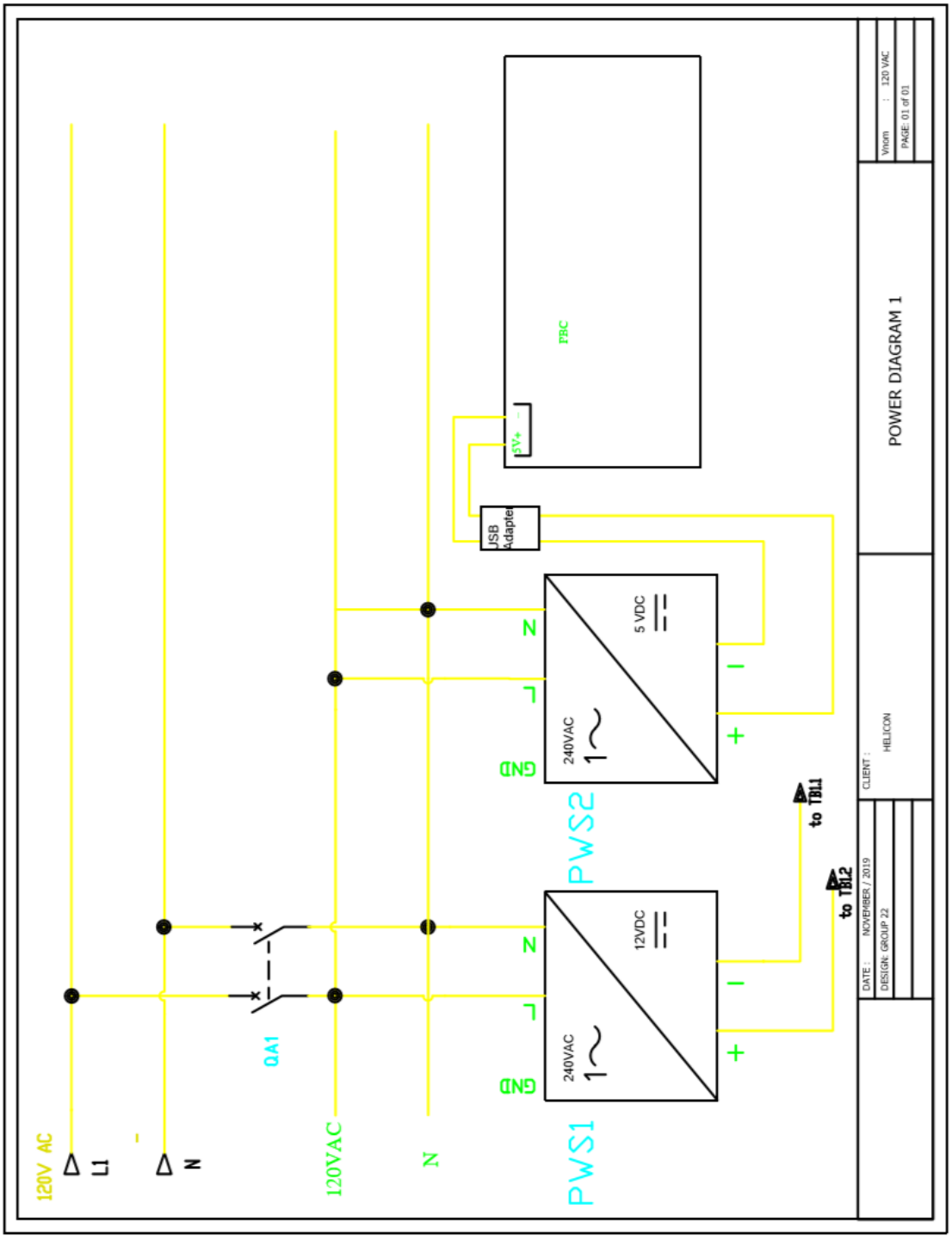

Figure 35 - Power Wiring Diagram

For our power supplies, we will use the naming convention PWS. As shown in Figure 35 above we have two power supplies. The first PWS1 providing 12VDC which will go to our bottom

terminal blocks (TB1.1 and TB1.2 for +12VDC and (-) VDC respectively) and from there connect to our molex connector and on to our servo motors. The second is PWS2 which will supply power to our PCB but this power must first pass through a USB adapter module so as to not change the capabilities of the PCB design itself. L1 is our live line providing the power this will be coming in from TB2.1 to our Circuit breaker QA1. N is our neutral line coming in from TB2.2 and also connects to our Circuit breaker. Ground wiring is not shown as it tends to add clutter. The Two power supplies will be connected in parallel to each other and in series to the circuit breaker so that the circuit breaker can protect them both.

5.5 Software

This section covers the design of the software, which will drive our project. It takes into account software considerations that have an impact on the design, the overall software design and program flow, and specific control of the peripheral components, including an in depth look at controlling each of the peripheral components.

5.5.1 Software Considerations

The language, IDE, application, and special features are all important design considerations for the software since they all have a hand in shaping the final code. Keeping these items in mind during design will create a product that more closely matches Helicon's requirements.

5.5.1.1 Arduino IDE and Language

One of the benefits to our choice of microcontroller is that we can utilize the Arduino IDE. This makes it easy to program the chip via USB since the IDE has a built in upload button. Of course, the IDE also provides a compiler for the code.

The Arduino IDE uses its own Arduino language which is based on C/C++ with more object oriented class structures. This makes it easy for our team to use since we are all familiar with C, but the object oriented nature will present a small learning curve to group members who haven't interacted with it before, mainly when they go to use the ready made classes from the libraries. The arduino language is composed of three main components, which are functions, variables, and structure. The structure component is also both familiar and not familiar to our team, as it contains the familiar logic statements like 'if' but also two required functions to make a 'sketch' work. These two functions are loop(), which takes the place of a main, and setup(). Loop() is repeatedly run without any necessary code on the programmers part. This is where the bulk of our code will go. We will put all of our startup code that will be run once on boot, such as pin initialization, within the other required function setup().

Another added benefit of using this language is that there are already many libraries that come included with the IDE as well as community contributed libraries available in public repositories like github. This will make it very easy to interact with the peripheral systems we are using, such as the SD card and LCD screen, which will save time during development.

5.5.1.2 UI Design

A small sized LCD, size 20 characters by 4 lines, will be used. This is a fair balance between cost and function and should serve our needs. The UI is primarily used to set two values by working in conjunction with two potentiometers and two push buttons. The code will need to be able to coordinate knob changes with value changes on screen for the user. Buttons will be used to select choices or cancel out of options. Buttons will have to be debounced or the LCD might unintentionally fly through values because the code read multiple button presses instead of one.

Placement of text will be limited by the 20 character dimension of the LCD. Due to this, shorter menu phrasing for options will be used, and in some cases the text will wrap to a second line. This still leaves two lines to allow for the ability to display the numerical menu options as well as indicate what the buttons represent for that menu page. Since the display is only utilizing two buttons, which will serve different purposes throughout the menu system, they will not be labeled physically. Instead we will rely on the LCD to indicate their purpose. As such, they will need to be situated close physically to the LCD. The buttons will be situated along the long side of the LCD ideally as this will maximize the amount of screen space for text versus placing the buttons on the short side. It will also serve the aesthetic value of providing a visually balanced user interface, as seen in Figure 36 below. Note that the two buttons in the figure, represented by black circles, are aligned under text that explains what actions the buttons will initiate.

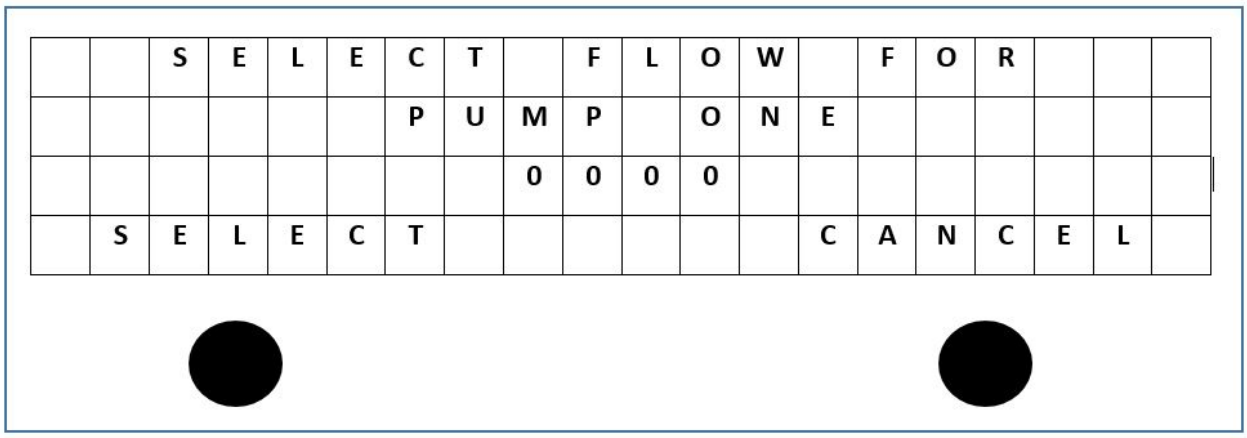

Figure 36 - Diagram of LCD and button placement

To create a more user friendly interface, we will use two potentiometers. These are familiar to anyone who has used an oscilloscope as they usually implement two knobs to fine tune input or manage the movement of cursors. The potentiometers in this case will make it easier for the user to set values because one potentiometer will be used for large increments, say 25mL, while the other will be for more finely tuned increments, say 1ml at a time. This allows the user to quickly make their selection instead of scrolling through every number individually from 1ml-4000ml, which is the range of volume that Helicon is working with.

5.5.1.3 Memory and Runtime Constraints

Working on the embedded level with limited memory and processing speed presents a unique challenge that encourages efficient code design. Saving function results for reuse, cutting out unnecessary loops, and reordering function calls in some cases can all speed up a process.

While the MKR zero has a program memory of 256KB, this might be an issue with the large number of libraries that we are using to control our peripherals. Program memory can be conserved by not including libraries that are unused, avoiding memory hungry actions, like list concatenation, and using local variables where possible that can then be discarded after use.

Another way to save memory is the treatment of string data. Arduino has two ways to handle strings. One is the same way that C does, with a char array, and the other way is to use a string class. While the string class provides many handy ways to do string manipulation, it also takes up more memory. So unless the string is going to be heavily manipulated, like when acquiring the date and time to create a file name which must be cut down to 8 characters, the string should be a char array to save program memory.

5.5.1.4 Emergency Shutdown

This is a safety critical function and will be partially implemented by using an interrupt. It will initiate a shutdown procedure that will first turn off the pumps. Then it will set valves to a neutral state where no reagent or inert gas can flow. This protects components and prevents reagents from being wasted. The emergency function will then update the log with the date and time that shutdown took place, as well as how far through the process the system was when it shut down. An alarm will also be issued using a red LED and speaker.

There are two ways to initiate the emergency shutdown. The first is through user input in the form of a large red button on the user interface. This button is connected to a pin that is monitoring for external interrupts. When this pin is triggered, it will read as high and the hardware will interpret this and signal to the software to interrupt the current process and go to a seperate emergency function. This emergency function is classified as an interrupt service routine (ISR) and cannot take any inputs or return any outputs. For that reason, in this emergency function, a global boolean flag will be set to indicate the emergency function was tripped. After that, the main process will resume. From there, this flag must be checked in the main process in order to call the actual emergency shutdown function that will turn off the pumps and set the valves, etc.

The other way the emergency shutdown is initiated is through an error in the return signal from the pump. The pump is checked periodically for functionality by the software, once every cycle through the main runtime function. If the signal returns not as expected, within a margin of error, then the software will set the global boolean flag to be checked by the software at strategic points. It is possible to update the log with which of these two events caused the shutdown by using two seperate global boolean variables. The only drawback is that the software would then have to check two flags every time it was evaluating whether or not an emergency shutdown flag had been set.

Since one of the ways to trigger the emergency function does not rely on a change in a digital pin, but instead on some logic on the software's part, it would be a good idea to go with a global boolean variable to indicate that an emergency shutdown needs to take place. This is because there would not be a way to attach an external interrupt to a pump error emergency shutdown without a pin to attach the external interrupt to. Intead, the flag will be set when the software checks the output signal with the returned feedback signal from the pump. This global variable flag will need to be checked at strategic points in the code. Since the emergency shutdown triggered by the pump will not be routed through the emergency shutdown interrupt function the same way a user initiated

emergency shutdown would be, it would make more sense to have the shutdown procedure be done in a different function that is called after periodically checking the flag. That reserves the emergency interrupt function for just setting the flag for an external interrupt before returning to the process.

Ideally, the emergency shutdown procedure would take place as soon as the emergency shutdown was triggered, but having to check a flag before initializing this procedure all but guarantees some kind of delay. This delay can be mitigated by checking the flag often, but we run into a few problems. In particular, our code will utilize a delay during the chemical process when the lines are being cleared by pumping through inert gas. During this delay, the flag cannot be checked. Another reason that using a flag is less than ideal is that it adds a lot of extra instructions to the program with repeated calls to check the flag. The more responsive we want our emergency shutdown to be, the more calls we will have to add. If we check multiple flags, then we add still more instructions to our program.

One way to handle checking the flag during long delays is to break the delay up. Timing would be important, but we would have control over how often the flag would be checked using smaller delays, anywhere from every few milliseconds to every few seconds. These delays might then have to be nested in while loops, that were formed something like ‘while(flag) do delay, check pump’ and immediately after the loop, check the flag again in an if statement to then divert the program to the emergency function if necessary.

It is worth noting that during the menu function, the pump should not be able to trigger an emergency function as it will not be getting any flow rate commands to check the feedback against. At that point, the pump should not even be started. It is still possible that a user could hit the red emergency button on the user interface, and the flag would be set since the system employs external interrupts to set the flag for that situation. But there is little direct purpose in initiating the emergency function from the menu function because at that time the valves should still be in a neutral state and the pump should be stopped. Therefore, there would be no immediate need for the user to stop the system while in the menu function, since the emergency function would return the system to a state that the system is already in.

The biggest issue of using a flag to indicate that an emergency shutdown needs to take place is deciding where the flag needs to be checked. It must be checked often, but not too often. It cannot be checked during delays, and if a flag is checked in a function, we need to decide how that function will enable us to handle the emergency shutdown.

The flag will not be checked in the menu function because as stated earlier, it would not provide the user much immediate benefit. The flag should be checked after returning from the menu function though because the next function call will be sending commands to the pumps which we don’t want to do at all after an emergency shutdown flag has been set. The flag should be checked periodically during the process function by using smaller delays, and that function should return something to indicate that the flag had been tripped and the current iteration of loop() should be brought to the end of the function, using a goto call, where the process can wait on some kind of user acknowledgement.

5.5.1.5 Protected Values

There are a few values in the code that should not be changed after they have been configured by our team. These are in relation to the valves on the system, each having a so called ‘dead zone’ that they should not be set to or it will open the lines between reagent and the vacuum pump. This would cause the corrosive reagents to enter the pump and thereby destroy an expensive piece of machinery. To protect against accidental tampering when doing code maintenance in the future, these values for the valves will be set in a header file and imported into the main program the same way a library is. The header file will include a warning against changing the values contained within. In order to import it into the project software, it must be placed in the same file as the other included libraries so that it may be selected from the drop down menu in the Arduino IDE, which allows the linker to find it.

5.5.2 Program Flow

The program must initially enter a setup function upon booting. This function sets the initial state of the pins, initializes class and variables, and does any calibrations for peripherals. The pump will need to be calibrated during this step according to the manual.

Near the top of the program we will start by defining anything that has to be hardcoded, such as pin numbers. It will include libraries for the LCD, SD, and PWM components. It will also define global variables at the top which eliminates the problem of ‘magic numbers’. This makes the code more easily maintainable.

After these elements, the main program begins. First, the user has to be able to set four values using the LCD in conjunction with two potentiometers and a push button. The user has to do this twice, once for each pump. The code needs to coordinate user actions with buttons and potentiometers with the LCD display values to provide a seamless user interaction. After the user makes one selection for the flow value, the code loops through the second value the user can select, volume. Once these two values are selected for each pump, a run loop can be called that actually oversees the chemical process. The figure below is the outline for the menu system, displaying the flow between LCD displays as well as where the software logic must be able to backtrack.

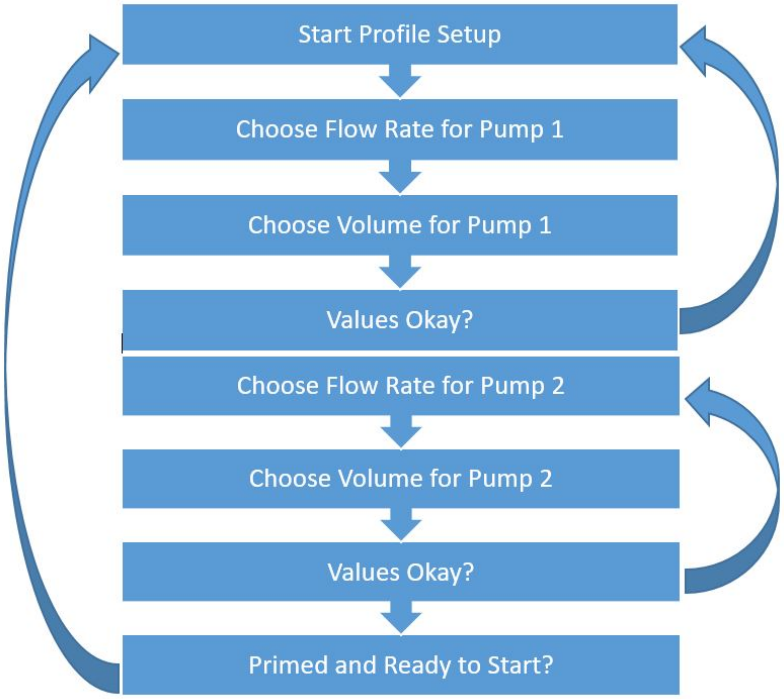

Figure 37 -Menu Flow

Once in the run function, pumps and motors need to be signaled by the microcontroller to start a purge of the lines with an inert gas. After a timed period caused by a delay in the code, the motors will be signaled to switch position to allow the reagents to flow. From there, the user needs to be able to prime the lines. Manual control will be handed over to the user by ceasing commands to the pumps. The user will then need to hit the push button on the user interface to signal when they are done priming. The LCD will display a 'Prime then Push button to Start' to indicate that the system is waiting for user input.

Commands will then be sent to the two pumps and this will start the process of mixing the reagents. This will happen for the calculated runtime based on flow and volume inputs by the user. The system will calculate runtime at the start of this loop by taking the volume, pump speed, and cross section area of the tubing. The runtimes will be measured against the real time clock using an RTC library, starting after the command is sent to the pumps and valves are opened. These values for the flow rate, volume, and start time will also be used to update the log so that Helicon can get a real time record of events in the case of an emergency shutdown.

When the profile is done running, an alarm will sound but the green LED will continue to stay on, indicating that the process has finished. This will differ from the emergency shutdown in that the red LED will not be lit, and another alarm noise is used that is different from the tone of the emergency system. The system will display a process complete message on the LCD that the user must acknowledge using the push button before another profile may be started. The log will be updated with the time the process was completed.

At any time a user may push the big red emergency button on the user interface. This will trigger an interrupt which will then signals pumps to stop and valves to close to a neutral position. It will also update the log with the message that an emergency shutdown occurred, what time it occurred, as well as how far along the process the system is calculated based off current runtime divided by total runtime. The interrupt will also turn on a red LED by setting that pin to high and signal the stereo to play a tone. The LCD will display a warning that needs to be cleared by pushing the push button on the user interface before another process can be started.

Represented below in Figure 38 is an overview of the process flow. It does not take into account the emergency systems or logging, but it does show the amount of user input that will be required for the system.

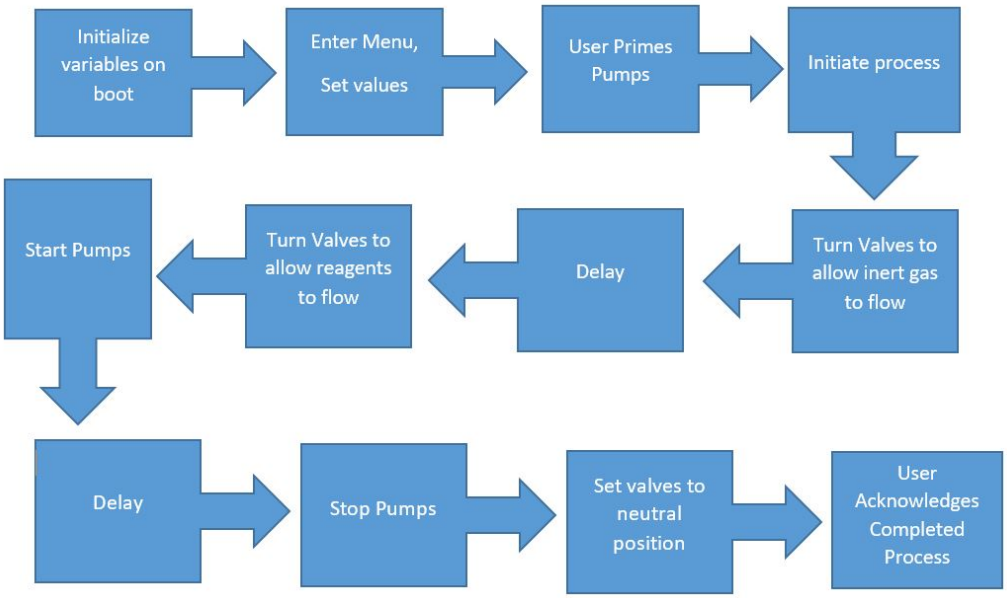

Figure 38 - Overview of Program Flow

5.5.3 Control of Peripherals

This section explores how the peripheral components will be controlled by the software. There are quite a few components, many needing to be initialized and handled in specific ways. Designing a system that can coordinate these peripherals starts with a good understanding of how to control them and the best way to do so.

5.5.3.1 LCD

To make an LCD Arduino compatible, it has to utilize a Hitachi HD44780 LCD controller chip so that we can use a specific library to interface with the LCD. Most alphanumeric LCDs use this controller chip, which is fortunate. There is the option to configure the LCD in 4-bit or 8-bit mode, and we are using 4-bit. This saves us 4 pins without a noticeable drop in quality from the LCD.

To control the LCD, we are using the LiquidCrystal library, which provides functions for turning the display on/off as well as printing to the display. The library has to be initialized with the pins it

uses in the setup() function, which is where we can designate by the number of pins which mode we are using.

At the start of the program, in the setup function, we have to define the size of the LCD in a begin() function as well as clear the LCD of any leftover displayed characters using a clear function. From there, the software is clear to display a startup message before transitioning into the values selection menu. The cursor functions in the LiquidCrystal library will be especially handy in configuring the UI so that it looks generally uncluttered and easy to read by keep values away from the edges of the LCD where they might be harder to read.

The code will strategically update the LCD display with user input to create a synchronous effect. To do this, the menu process will be in its own loop within the main function. The only way to break out of the loop will be to get to the 'okay' at the end of the menu system and have the user push the button that corresponds to 'okay'. The software will use digitalRead() on that pin attached to the button every loop to check if it has been pressed, which will set a flag to true if it has. This flag is being checked by the loop every iteration, something like a 'while(flag)', which is evaluated before any of the body of the while loop is run. Each portion of the menu, meaning each menu 'page', will have an equivalent loop within the code.

The values for volume and flow rate will be pre-populated for the user in the menu to the lowest value. This will be done by having the value actually be a variable within the software, initialized to the lowest value a user can select (1ml at this time) outside of the menu loop. This prevents a null error and even if the user mistakenly hits okay right away, they will have the opportunity to reset this value in the next menu screen.

Since the only thing the user is changing is a value for these menu pages, it is easy to update software variables behind the scenes according to the user input and then have the alphanumeric LCD display this. For this part of the code, we will use the setCursor() function to navigate to the position we are updating on the LCD, and then using the write() function to overwrite the value that was previously there. A bit of trial and error will be in order to see if filler characters will need to be used to overwrite values between user choices. For instance, if the user goes from 100ml down to 10ml, if the LCD doesn't overwrite that last 0 left over from the 100ml, then the LCD will still be displaying 100ml to the user, even if behind the scenes the software is storing the correct 10 ml variable. Filler characters would be a better solution than simply re-writing the whole LCD screen as it saves instructions in the code and increases the speed the LCD can be updated.

5.5.3.2 Pumps

The pump wiring interface was discussed earlier in section 5.3.2.1, all inputs and outputs mentioned in that section will need to be be read/generated by the software in the CCU. In this section we will outline this process for each signal. The I/O signals are reproduced below:

Inputs used: Remote Start/Stop (Digital), 0-10V Speed Control Input (Analog)
Inputs unused: Remote CW/CCW, Remote Prime, Aux in

Outputs used: 0-10V Speed Feedback Output (Analog), General Alarm Output (Digital)
Outputs unused: COM (Motor Running), Tach Output, Local Remote Indicator

Remote start/stop (Digital): Throughout the process both pumps will need to activate to drive material through the system. This will be done via pins as noted in table 6 from section 5.6 below. The program will generate an active low signal to these pin by pulling them to ground with a digitalwrite() command. The active low triggers the pump start, and the pumps will run until the pins are switched back high with a second digitalwrite() command.

Pump Flow Rate (0-10V Speed Control Input, Analog): The desired flow rate will be input from the user via the UI and stored in a CCU register. This will be done by reading the voltages across the two potentiometers that were set by the user. One potentiometer will be coarse, increasing "flow rate" by 10mL increments, the other fine, using 1mL increments. Both will be connected to the analog inputs and their values read with an analogRead() command. From here, the signal from the 10mL pot will be multiplied by 10 and added to the signal.

Working with these analog input signals will require some considerations. First, the input resolution on the pump circuitry is 10 bits. The default output for the MKRzero DAC's is 8 bit resolution. This is undesirable because it means our automatic system will not have access to as many speed inputs as a Helicon technician would have manually. Thankfully the DAC pins on our chip can be configured to higher resolutions with an analogWriteResolution() command. As long as the resolutions are the same then the process described in the paragraph above should suffice. Second, the maximum analog voltage is higher than the logic voltage of the MKRZero. We are somewhat lucky that the 10V max input is simply double the 5V logic level, but it is still less than ideal. So far our only option seems to be to send the pump input (controller output) analog signal through a simple 2x amplifier circuit. Originally it was thought the resolution would suffer but after further consideration it seems as long as both the MKRZero and Pump signal converters have 10bit resolution we should be good.

0-10V Speed Feedback Output (Analog): The Masterflex pumps have built-in feedback to determine their flow rate. The embedded controller will generate a 0-10V analog voltage signal that must be read by our software. Like the input resolution, the output signal resolution is 10 bits which means 1024 different values between 0-10, giving a step size of 10/1024 = 9.7mV.

The output from the pump will run through the voltage divider outlined in section 5.3.2.1 to step it down to the MKZero logic levels. The controller can then compare this value to the set point to verify the pump has taken the desired action. If the pump feedback differs from the set point by more than a few steps, a signal will be sent to the error indicators to notify the technicians. If the error is large enough, it may be necessary to initiate the emergency shutdown procedure.

General Alarm Output (Digital): The Masterflex pumps have their own alarm systems that differ from the alarms we will be implementing in our project. The alarm scenarios are outlined in the pump manual, which has not been included for copyright reasons. If the pumps encounter any internal issues that would display an alarm, the General Alarm Output will be pulled low. This action will need to trigger an interrupt in our program so that our alarm system can also be triggered and the technician can be notified. Since the specific scenario that triggered the pump alarm will not be communicated, it is necessary to halt all operation if a pump alarm is detected and wait for human input.

5.5.3.2.1 Unused Inputs and Outputs

As discussed in other sections, the Masterflex pumps have many more remote inputs and outputs that may be useful, but will not be used in the scope of this project. It's important for us to include loose wires in our DB-25 connector that access these for easy access in the future if Helicon wants to upgrade the system. In this section we will briefly outline the software considerations for these inputs and outputs in case it becomes necessary to include them.

Remote CW/CCW Input (Digital): The remote CW/CCW (Clockwise/Counterclockwise) reverses the direction of the pumps. In the default state this input is high impedance and the motor runs clockwise, to switch to counterclockwise the input is pulled to active low. This can be done with a digitalWrite() command. If Helicon eventually wants to implement this feature, this command will have to be written in to our main() function at the desired location.

Remote Prime Input (Digital): As mentioned previously, Helicon needs to prime their system in order to run properly. Helicon's system is a prototype and does not have the hardware capability to configure this step remotely. Therefore the current design calls for this step to be performed before our system is engaged. In the future, if this step can be automated, the prime input can be configured with a continuous active low signal via a digitalWrite() command. The program will pull this pin low for the set time, or until a future sensor detects the system has been primed.

COM Motor Running Output (Digital): The pumps are also capable of internally determining whether or not the motor is running. For our project we are using the flow rate voltage output as our feedback mechanism. However, if this proves to be insufficient, the COM output can also be used.

Tachometer Output (Analog): Along with a motor running output, the pumps can also output the motor revolutions per minute. In the future this could give us a redundant measure on the flow rate if we make the intermediate calculations. The manual specifies that the Tachometer output is an open collector, 1.0A @ 28V DC 100 to 6500 Hz, 50% duty cycle. The frequency output is simply divided by ten to give the motor RPM. See section 5.3.2.1 for specific considerations regarding open collector outputs. The Tachometer output can be read with an analogRead() command as long as the resolution has been set to ten bits.

Local Remote Indicator Output (Digital): The last unused output is the local remote indicator. The peristaltic pumps have a visual indicator when they are being controlled remotely via the DB-25 connection. The built in pump indicators will be displayed on the same level as our control box (see the white boxes on either end of the Top Shelf in figure 11 from section 5.1), so this output is unnecessary in our current designs. If needed, however, it can be read with a digitalRead() command. If the digital read returns HIGH (high impedance), then the pumps are in remote operation and this can be translated into our own indicator.

5.5.3.3 Motors

We are using the MKR motor carrier to prototype our project, which allows us to use the arduino library specific to this board, the MKRMotorCarrier library. It includes functions for setting the angle of servos as well as the duty cycle for motors. The software will have hard coded values for

the valve positions since these are something that cannot change without serious design changes. Each motor will be given a different value that signifies whether the valve will be turned to allow reagent to flow, inert gas to flow, or nothing to flow. These values will be saved in a separate header file with sufficient warnings to deter accidental manipulation.

The use of PWM signals to the motors controlling the valves will be done during strategic times within the program. The valves, which are three way valves, will be signaled to turn to a position that lets inert gas clear the lines first. After a set period of time, the valves will then be signaled to turn to allow reagents to flow. During an emergency shutdown or after a process has been completed, the valves will be signaled to turn to a neutral position that does not allow any material to pass through them.

As noted in section 7.2.5, motor control will be a very sensitive endeavor. These three way valves can never be in a position such that all three lines are exposed to each other. The majority of the solution for this problem will come from the testing before the program is run in real time. However, it is important to keep this "dead-zone" in mind when programming motor control.

5.5.3.4 SD

The SD card will contain the log for the profiles being run. To communicate with the SD card, we are using the SPI protocol, and to write to the SD module, we are utilizing the SD library. To select the SD card we would like to write to, use SD.begin(), without filling in any pins for the function call, and the default will be the SS line of the SPI1 that the built in SD module is connected to. This works well for prototyping, but might need some tweaking when we lay out our PCB board with different pin numbers. In that case, we would fill in the pin number that corresponds to the 'slave select' on our board, and we would have to set the pinMode of that pin to OUTPUT in another command.

The SD card must be properly formatted first so that this SD library can be used, making sure the file system is set to FAT16 using a PC. Naming conventions for files must follow the 8.3 format, where the name can be at most 8 characters and the file extension can be no more than three characters. This will pose an interesting challenge for our application where we would like our file names to be the date and time that they are created.

The layout of our file system is important and must be well defined so that users can locate the log files they want. It is also important in order to stop the program from prematurely overwriting files. A new file will be created for every run of a profile, created at the time the user selects okay to the settings they have just chosen using the UI. The software will generate a new file in the current directory by using the open() command on a file name, which creates a file if it does not already exist. In this case, it protects from errors because if the file does exist, it simply opens it. Either way, there is a file for the software to write data to instead of faulting at the earliest write() function.

All created files will be placed in the root directory. They will be named according to the time and date they were created. This will be done using the RTC library to get the time and date, manipulating this string to cut it down to 8 characters, which is the maximum that we can name a file, and then using this name to open() the file. Naming the files with the date/time they are created

will prevent an incorrect file from being appended to because it provides the file with a unique name. If these file names were simply incremented variables instead, then every time the software was started up, the SD card, with older existing files that have these same incremented values as names, would just append to those files. This is not what we want, we need files that accurately represent data from each run through a profile individually.

Figure 39 shows an example of the naming convention in action, with the first two digits representing the month, the next two representing the day, the next two representing the hour in military time, and the final two digits representing the minute. The file is placed in the working directory, represented by the '/' at the top of the figure.

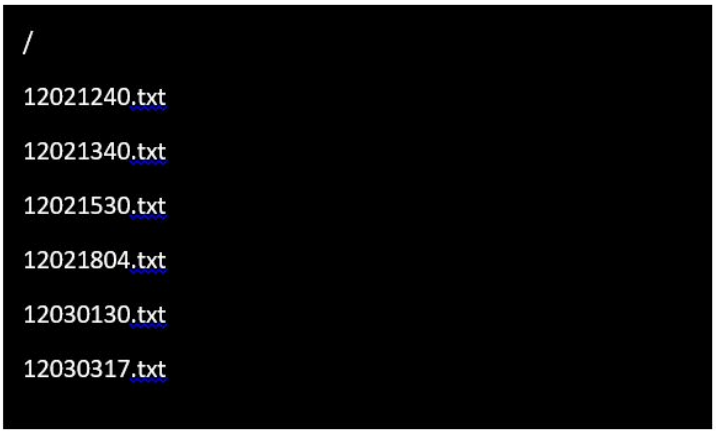

Figure 39 - File Naming and Directory Example

Files are closed by the software after updating them in order to save progress. Opening and closing files constantly slows down write speed to the SD card. This might become a cause of concern later in the project if we decide to implement sensors and sensor data needs to be periodically added to the log. Write speed can then be increased by doing strategic flush calls every 50 cycles or so instead of every time something is appended to a file. Waiting for this number of cycles is fast enough to mitigate the likelihood of losing data due to a sudden loss of power, but still exponentially speed up the writes.

There will be no software support for reading the SD cards in our project. The user must remove the card and read the content on a computer. The SD card has been configured in FAT16 so it will be recognizable and readable to a PC. This allows the user the opportunity to save logs and back them up on an external server for record keeping as the SD card itself will have limited space. There is no easy way to test when the SD card becomes full and to signal this to the user. The user will have to periodically remove the card and save data if they wish to avoid losing data as the card will rewrite older entries when it becomes full. During testing we will see how many average runs the SD card can store and provide that in documents going forwards.

5.5.3.5 LEDs and Speaker

LEDs and alarm sounds played from a speaker will be used to signal to the user if the profile is complete or an emergency shutdown has taken place. For the LEDs, the software strategically

triggers a high output to the red or green LED pin, both digital pins, to turn it on, depending on the situation.

The speaker needs to be able to play two tunes to signal if it is an emergency or just the profile finishing. These tunes will be coded into the software, not stored as mp3 files on the SD card. This prevents them from being accidentally deleted which would cause faults at runtime when the program tried to access files it couldn't find. It also doesn't take away from space on the SD card that could be used to store more log data. The downside is that it will take up more program memory instead. The speaker is controlled by an analog voltage, and there are many libraries available for coding a specific voltage to a musical note. One way to create a tune will be to set the analog pins to certain voltages for a set period of time. Arduino has a built in tune() function that could be used as it generates a square wave using a 50% duty cycle on a pin and can also be specified to run for a period of time. Since the melody will be kept simple, being only one note at a time for the same length of time, it should be straightforward to set a delay between notes so that they can each be distinguished from each other. The melody selected will be something generally thought pleasant to signal the completed profile, and mimic an emergency tone to signal the emergency shutdown. Ambient noise in the lab environment will need to be assessed to see if an amplifier will be required or if the speaker alone will be sufficient.

The emergency tune will be played from a separate interrupt function, not at the end of the process loop like the 'process complete' tune. This encourages the use of global values for the hardcoded musical notes, or the use of a library. Since memory is already a thing under consideration, it would be better for us to hard code ourselves only the notes we will need to make our two chosen melodies.

5.5.3.6 Buttons and Potentiometers

Stepped potentiometers will be used as they will make it easier to code up incremental changes to variables in the software. They are also familiar to the user and are intuitive to use. Two potentiometers will be used in conjunction to allow the user to fine tune their selection of flow and volume. One will be used for larger granularity, and the other is used for more fine changes, similar to the way function generators set values.

There will be two push buttons for the user to interact with. Both are necessary even though the software will take over giving it a different purpose depending on what the LCD is displaying because the user will need to be able to select 'okay' or 'cancel' in order to successfully navigate the menu. It will be used to select a specified flow or volume values, start a profile, and clear a warning alarm. The code will read high when the button is pressed, and translate that into a flag that will signal an exit from a loop in all cases. The flags will correspond to specific LCD displays.

The emergency button in particular will have an interrupt attached to it so that when the pin goes from low to high, or on the rising edge, the main process will switch control to an interrupt service routine. Having the interrupt trigger only on the rising edge instead of when the pin is high will keep the program from incorrectly triggering multiple interrupts if the button is held down.

The code will utilize the analogRead() functions for the potentiometers. While not an issue, it is good to keep in mind that the max speed of this function is about 100 milliseconds, or you could do 10,000 reads a second. The code will assign an initial value to the variable flow or volume,

depending which is being set at the time, by reading the voltage of the potentiometer and translating that using a multiplier based on the range, currently 1ml to 4000ml, to get a value in that range. The code will balance the two potentiometers by having one always add its value to the other. For example, if we make one potentiometer increment by 10 and the other by 1, and both potentiometers are in their lowest position, then we will be at 0. Turn the 10 increment knob and get 10 plus 0 to give 10. Turn the 1 increment knob and get 10 plus 1 to get 11. The software in the background is just constantly calling analogRead() on the two potentiometers, translating the voltages into values that match the scale for each potentiometer, and then adding these two values together for display by the UI.

5.5.4 Version Control and Debugging

GitHub, the online code repository that utilizes git, will be used to maintain version control of the software. While there is only one prototype board at the moment, having the code hosted independently of any one group member's machine allows multiple people to work on and view up to date code. It also prevents the worst from happening if the only laptop that contains all the code decides to die unexpectedly by providing a backup stored safely away on a server.

Arduino doesn't have an onboard debugger, nor does the Arduino IDE come with any built in debugging tools. This could prove challenging and add to development time as it will be hard to 'see' where the code is going wrong.

The Arduino IDE does provide a serial monitor that can be used to communicate with the microcontroller while it is running via a USB. This could be used to send in specific values for variables in order to test the functionality of certain parts of the code. It can also be used to help 'see' what the software is setting variable to in certain parts of the code. While this method of debugging will create some code to clean up after debugging is done, it could be a powerful tool.

Another good starting point for debugging will be utilizing an oscilloscope to see what, if any, signals are being sent from the microcontroller to the other peripherals it is supposed to control. The LCD will also be employed to test certain points in the code by outputting to the screen values for variables during runtime.

Something to keep in mind is that our program might be using goto commands to enable us to break out of deeply nested loops in the event of an emergency shutdown. Goto commands are generally frowned on because they make program flow more undefined which in turn makes debugging very difficult.

5.6 Subsystem Interface

Our subsystems will communicate through wires. It is important to take this design consideration seriously, as improper connections will cause communication failure and thus failure of the system as a whole. Our design calls for a wiring harness to keep the wires organized and clean. The wiring harness will be separated into two sections: section A will route wires to all the elements physically located on Helicon's cart, the servos and peristaltic pumps. Section B will route wires from section A to the corresponding pins on our PCB. The harness is divided into two sections due to the need to converge and diverge at a single point in between the control box and the control loops. This is

because if the control box is located on top of the cart, it will need to be removed during cleaning. If the control box is placed on a separate stand, a single disconnect is still very useful for ergonomic considerations. A multi-pin connector will be used at this point to create a semi-permanent junction that can be unplugged when necessary.

In order to ensure proper routing in setup, as well as for reference in the case of repairs, a wiring harness diagram has been created. The diagram maps all physical wires that run through the single connector. See Figure 40 below.

Along with a wiring harness diagram, a color code table has also been generated to aid in setup and maintenance. See Table 6 below. Note that for the DB-25 connectors several unused wires have been added in case they are needed in the future.

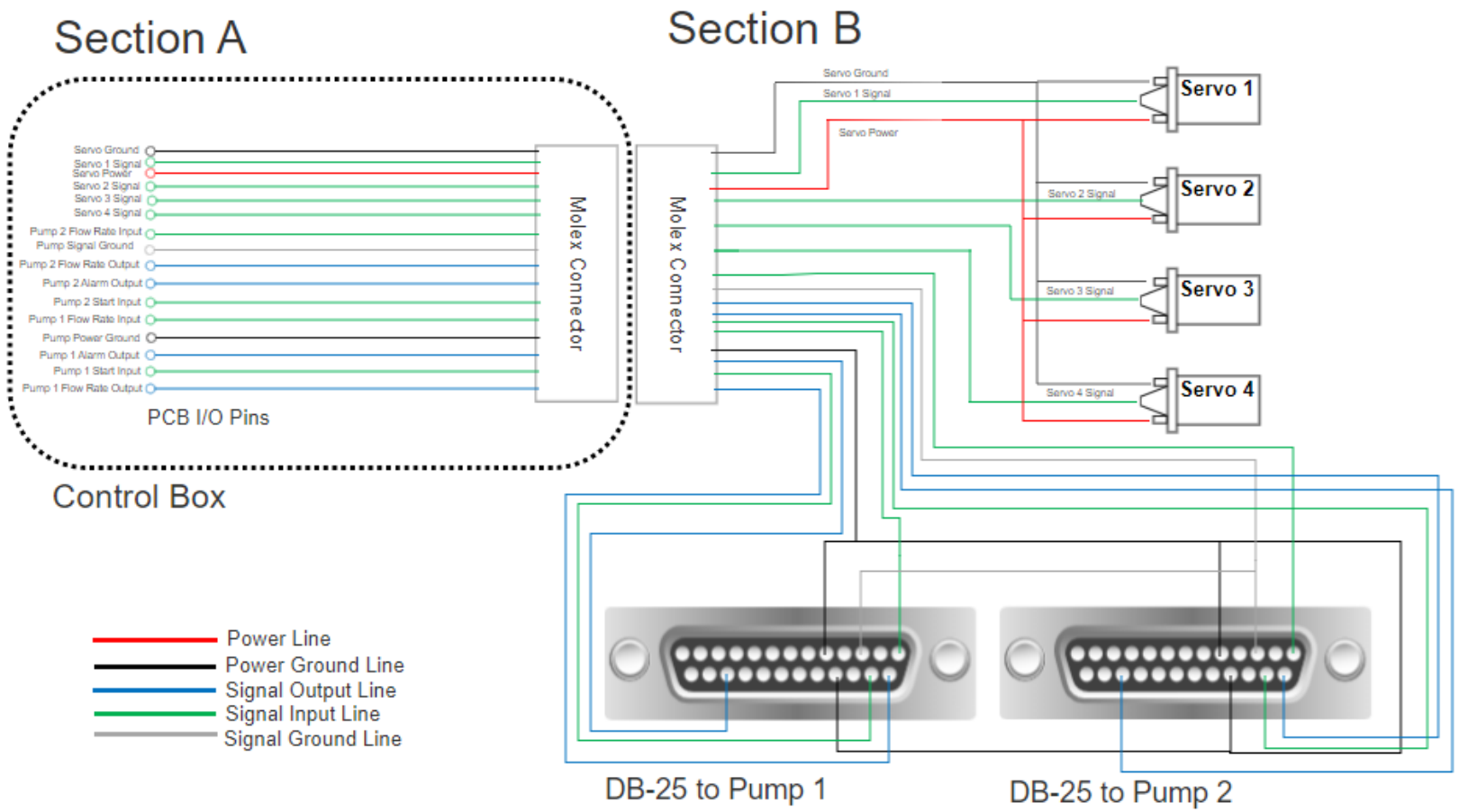

Figure 40 - Wiring Harness Diagram

WIRING HARNESS DIAGRAM For Control Box - Cart Interface	DRAWN BY	CHECKED	DATE	SCALE	SHEET NO.
	Jason	N/A	Nov - 2019	N/A	N/A

Table 6 - Wiring Harness Organization

PCB I/O Pin Number	Wire Color	Connector Pin Number	Wire Endpoint	DB-25 Pin Number
TBD	Black	1	Servo Ground	N/A
TBD	Orange/White	2	Servo 1 Signal	N/A
TBD	Red	3	Servo Power	N/A
TBD	White/Yellow	4	Servo 2 Signal	N/A
TBD	White/Brown	5	Servo 3 Signal	N/A
TBD	White/Green	6	Servo 4 Signal	N/A
TBD	White/Blue	7	Pump 2 Flow Rate Input	2-1
TBD	Grey	8	Pump Signal Ground	2-3, 2-4
TBD	Blue	9	Pump 2 Flow Rate Output	2-14
TBD	Green	10	Pump 2 Alarm Output	2-23
TBD	Orange/Green	11	Pump 2 Start Input	2-15
TBD	Orange/Purple	12	Pump 1 Flow Rate Input	1-1
TBD	Black/White	13	Pump Power Ground	5, 13, 17, 18
TBD	Brown	14	Pump 1 Alarm Output	1-23
TBD	Yellow	15	Pump 1 Start Input	1-15
TBD	White/Red	16	Pump 1 Flow Rate Output	1-14
N/A	Yellow/White	Unused	Tach Output	19
N/A	Blue/White	Unused	Prime	20
N/A	Green/White	Unused	Input 0-20mA; 4-20mA	2
N/A	Brown/White	Unused	Output 0-20mA; 4-20mA	4
N/A	Green/Orange	Unused	Local.Remote Indicator	24
N/A	Purple/Orange	Unused	CW/CCW	16

Table 6: Wiring Harness Organization

6. Project Prototype Construction and Coding

The following section outlines the chosen vendor for our PCB board as well as a final coding plan for the software. Both are essential for prototype construction, and thorough planning during this phase will save our group time during the later stages of construction.

6.1 PCB Vendor and Assembly

Criteria for choosing a PCB vendor will be based on manufacturer reliability, lead time, and cost.

Our choice of a vendor to facilitate fabrication of our PCB is one that we must make with extremely careful consideration. The PCB is the heart of our design and reliability, lead time, and cost are all considerations that we must take into account when choosing a vendor. Upon initial research we had stumbled upon JLCPCB, an experienced PCB manufacturing enterprise that is rooted in China. Looking further at their services, the starting price for a 4 layer board given a size of 50x50mm is $13 dollars (excluding shipping and handling). Given that this manufacturer is based in China, unfavorable lead time due to shipping, and the unfavorable pricing we were forced to look to other manufacturers. Further research into PCB manufacturers brought us to OSH Park, which offers PCB fabrication services based in the United States (reasonable lead time due to domestic shipping) and at a reasonable price. Seeing as this is a much smaller manufacturer the sizing and fabrication options are limited, but seeing as our board is not an industrial size board with limiting specifications, we can trust OSH Park with meeting our needs.

OSH Park only offers services related to manufacturing either two-layer boards (6 unique service options) and/or four-layer boards (only 2 unique service options). Due to the fact that we will be manufacturing a 4-layer board (top layer, ground plane, power plane, and bottom layer), we can ignore the service options related to two-layer boards. The service that we are focused on related to the four-layer board is the prototyping service, as this fulfills all our needs at a reasonable cost and with a good lead time. The service plan is as follows:

- Price: $10 per square inch, per set of 3
- Time To Ship: 9-12 Calendar Days
- Board Thickness: 63mil (1.6mm)
- Copper Weight: 1oz outer and 0.5 oz inner

I

In addition, the service plan on their website offers many different specifications related to copper, drill, stackup, and materials. As mentioned before, our PCB is not industrial grade, with limiting specifications, so many of the specifications related to copper and drill measurements will be perfectly fine for our design. Some specifications that we must note are:

- Material substrate: 190Tg FR408-HR (good thermal performance and lead-free assembly)
- Minimum Board Size: 0.25in (6.35mm) by 0.25in (6.35mm)
- Maximum Board Size: 16in (406.4mm) by 22in (558.8mm)
- PCB Finish: ENIG (slightly more expensive than conventional finishes, but is needed to ensure reliability and sustainability of our PCB).

OSH Park offers exactly the PCB manufacturing service that we need, but they do not offer any services related to assembly of the board. To cut down on costs and to ensure that our design comes to fruition the way we envisioned it, we will be personally populating and soldering the board ourselves. The assembly should not be too difficult of a process as we all have personal experience soldering through UCF extracurriculars and internships.

We have also considered the possibility of running into trouble with mounting the components ourselves. For this reason we have contacted an acquaintance, Jeff, a UCF EE alumni, who currently works as a PCB designer for a start-up company. Jeff gave us a tour of his workstation and he has all the capabilities to mount components. He offered his assistance if we needed it.

6.2 Final Coding Plan

This section will define the coding plan for the protoboard as well as adaptations to the final prototype. All team members have experience working in C and Assembly, so some software tasking will be broken up. This section covers the software environment, software components, and adaptations for the final prototype.

6.2.1 Software Environment

This section describes the software environment for the final coding plan. It will be used for both the protoboard as well as the final prototype. The environment includes the code management software and Arduino IDE.

6.2.1.1 Code Management

Git will be used to maintain version control of the software. While there will primarily be one group member working on code, and more than likely only one testing rig configured, version control is always a good idea. Git is commonly used in the workplace and there are many available resources to learn how to use it. Our chosen site to host the remote repository will be GitHub since it is free to set up private repositories for students and it has good documentation on how to use all the tools.

Branches will start with development, and each member that wishes to work on the code must make their own branch to work off of. Before merging their branch into development, we will have another group member review the code and test it with the existing code base. This will keep all group members up to date on the software while maintaining a high level of software quality.

6.2.1.2 Arduino IDE

The latest version of the arduino IDE, Arduino 1.8.10 will be used, in conjunction with a USB to micro usb cable for programming the prototype boards. The arduino IDE comes with several libraries pre-installed, and our group will be using the LiquidCrystal, SPI, RTC, and SD libraries from these.

6.2.2 Software Components

This section describes the four main components of the software. The setup function is important as it will set the groundwork for the other three functions. These other three functions provide the control framework that all other tasks will be built off of. Having three functions outside of the main loop strikes a good balance between readability and efficiency of the code. Each function can then be divided into further statements, expressions, and helping functions. Subdividing the software this way also allows more than one person to work on it without rewriting code.

6.2.2.1 Setup, Libraries, and Constants

The first step is to include all the necessary libraries, like SD, LiquidCrystal, and SPI. This is done similarly to C, where it is a '#include<SPI.h>' statement at the head of the code, outside of any functions. Use the following IDE menu sequence to include libraries: sketch, include library. This ensures that the IDE linker can find the library.

Next, we define all pin assignments as global variables. Global variables are defined outside of all functions and at the top of the code to make readability and maintainability easier, and so that they are within the scope of other functions. These values need to be given descriptive variable names to make code readability easier. For example, instead of naming pin 17 'abc' which tells us nothing about its function, we will name it something descriptive like 'button1'. Among these variable names, we will need a chipSelect pin assignment, or something similar for the SD initialization.

The setup function is called once at the time the board is powered up. In this function we will initialize the SD library and card using begin() with the chipSelect pin. Error handling will need to be employed here as the user could forget to insert an SD card causing this code to crash. We will also initialize the RTC library with hard coded values for date and time.

The loop() function takes the place of a main() function in traditional C code, with the code being repeatedly run while the board is on. First the loop() function must declare all the variables it needs at the start of the loop. This is good coding practice because it makes readability and maintainability easier. loop() will also create a name for the file at the head of the code.

To create a name for the file, we will be using the RTC library. It will use a real time clock to grab the date and time, and convert this into an eight character string that will be used later as the name of the file to write log data to.

The loop() function must declare and initialize the four values to their minimum values. loop() will call menu() and pass these four values in by reference. After the menu() function is done executing, the loop() function will open the file with the name created earlier and print to the log these updated values. loop() will then close the file so that these values will be saved and available if there is an emergency shutdown. Following this, loop() will display to the LCD a question asking if the user has primed the system. loop() will use a while loop to wait for a confirmation via a reading of high on the pin correlating to select before moving on.

loop() must check for emergency flags before calling the process() function in case the user pressed the emergency button at some point in the menu() function. If loop() detects the flag has been set, then it will call the emergency function and then goto a label at the end of loop(). This label at the end of loop will reset the flag.

After checking for this flag, loop() will call process(). Process() returns an integer which loop() will store in a variable to check for errors. If process() returned a 1, indicating an emergency shutdown needs to take place, then it will do as above, calling the emergency shutdown function and then goto a label at the end of the function. If process() returns a 0, then that means the process has been completed. loop() can write this to the file here, as well as the current date and time using the RTC library get functions and some string manipulation.

The last thing in loop is a goto label. Under this is an if statement that checks the state of the flag and if it is set, resets it back to boolean false for the next iteration of loop(), which represents another process.

6.2.2.2 Menu Function

The menu function will encapsulate all the user input and LCD menu code. Its critical function is to set the two values necessary for each pump, the volume and flow rate. The form that these values are stored and manipulated in is important. They should not be saved as global variables as this allows them to be edited by any function, and they should only be editable by the menu function. They should be initialized to their minimum values in the loop() function before any function calls are made. They will be passed in by reference to the menu() function so that modifications made in that function will be reflected in loop(). From there, these values can be passed by value to the process() function where any modifications will not be visible from loop(). Since the menu() function will not return any values, it must be a void function.

Timing is critical to the menu() function. We cannot write to the LCD too often since it won't be able to keep up. To achieve this, small time delays will have to be added using the delayMicroseconds() function. It will take a little trial and error to add a delay that is small enough that it does not interrupt the user experience, but large enough that it gives the LCD enough time to make changes.

The structure of the menu function will be a switch statement. The menu has to have the ability to backtrack incase the user would like to go back and change values they have already entered. Allowing the user to do this makes our system easier to use. The switch statement will be easier to follow than a series of nested loops, making it easier to maintain in the future. Each menu page on the LCD will correlate to a case in the switch statement. From each case, it is possible to update the variable the switch statement depends on and thereby navigate to other pages. For example, the initial landing page of the menu will simply ask the user if they would like to select parameters for the process. This menu screen correlates to a value of 0 in the switch statement. The case 0 will output to the LCD this question, and then enter a loop that is continuously checking for a button press in the form of a pin reading as high. When this condition is met, the switch statement variable will be set to 1, we break out of the case statement, and then in the next iteration of the switch statement we will move to case 1, which asks the user to select the flow rate for pump 1. From this menu screen, the user has the ability to hit cancel, which the program interprets by reading another pin as high. When this happens, the switch statement variable can be reset to 0, taking the user back to the first menu screen on the next iteration of the switch statement.

Each of the four cases that correspond to setting a value also needs to be able to monitor the potentiometers and translate this into values for the variables that will be on screen. Inside each of

these cases, the program will check to see if the select button has been pressed, which will save the current value being displayed to one of the four variables being set, and then set the switch variable to the next menu screen, at which point the program breaks out of the switch statement. If the select button is not pressed, the program will then check to see if the cancel button has been pressed, which will cause the program to set the switch variable back to either the initial menu screen or a halfway point, depending where in the menu sequence the user is. It is worth noting that hitting cancel from the halfway point should then take the user back to the initial menu screen. This way the user has the ability to go all the way back and reset any of the variables they need to up until they exit the menu function.

When the user gets to the last case in the switch statement, which corresponds to a menu screen showing the user the values they have selected, the user again has the chance to confirm or cancel. As a side note, these values are populated by dereferencing the values that were passed into the menu function and set in earlier menu screens by saving values to these references. Therefore these are the values that will be used by other functions so it is doubly important for the user to make sure they are correct. Pressing the cancel button will cause the program to set the switch variable back to 0 correlating with the initial menu case, but confirming will cause the program to return from the menu function. Even though the function is a void function, c still allows us to return. Before returning from the menu function, we can either clear the LCD or leave the four variables up, depending on what Helicon would prefer. Further consultation with Helicon will be needed to provide this fine tuning of the menu system and LCD display.

A total of eight case statements will be used, correlating with each blue square in Figure 41 below. The figure also illustrates the text that will be displayed on the screen during the menu sequence. The eighth square in the figure illustrates the type of shorthand writing that will have to be employed to get all text displayed on an LCD with a 20x4 character restriction.

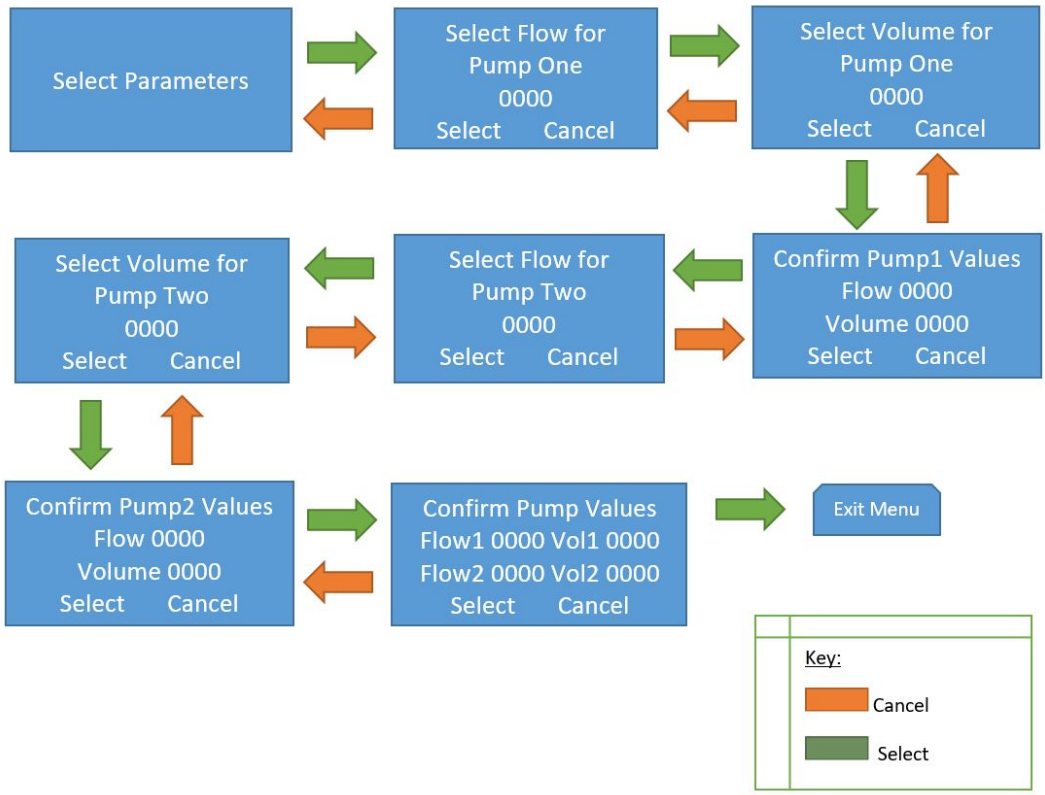

Figure 41 - Menu System

6.2.2.3 Chemical Process Function

The process() function will be passed four integer values, which are the volume and flow rate for each pump. This function will return an integer, which will correlate to either a run with or without errors. The function will return 0 for no errors and 1 if it detected an emergency shutdown.

The function will first calculate a runtime based off of the values it was passed. Using the setAlarm() and enableAlarm() functions from the RTC library, we can trigger an action to happen when the process should have had enough time to finish, as calculated in the previous step. We create a small helper function called processComplete() to compartmentalize the code that will notify users when a process should have had enough time to finish. In the processComplete() function, we will set the Green led pin to High so that it turns on as well as using the analogueWrite() function on the pin for the speakers to play a melody. Having this code compartmentalized makes it easy to maintain in the future, and could be easily removed as well if necessary.

Before setting any alarms, the pump and valves must be ready to start the process in order for the most accurate possible finish time. After calculating a runtime, the process() function will then signal the valves to open to the position that allows inert gas to flow, using the analogueWrite() function on each of the motor pins. There will be a different value for each valve, and they are stored as const values in a custom library. Storing them in a library instead of just declaring them

at the top of the code makes them slightly less accessible and should be a good signal to anyone maintaining the code that these values should not be changed. After signaling the valves, we will use a timed delay to give the pressurized inert gas enough time to clear the lines. The amount of time will not vary between different processes, but we will need to work closely with our POC for Helicon to get an accurate time for this, and it will require a bit of back and forth tweaking of the code for this phase. During the delay, periodic checking for the emergency flag will take place by breaking up the delay into smaller delays. If a flag has been set, then the function will return 1 and loop() will handle the emergency function from there.

After the set time has elapsed, the valves will be turned to their other position that allows the reagent to flow using the analogueWrite() function to each of the motor pins. These values will again be different for each valve and they are stored in the custom library. After all the valves are open, the pumps then need to be set and started by sending a signal using the analogWrite() function to the two pumps that is equivalent to the user set flow rate for each. Following this, a delay is used to give the program enough time to complete the process.

During the delay for the process, the delay will be broken up into smaller delays that allow the program to periodically check the emergency flag and also to check that the pump feedback is equivalent to the pump flow rate. Since the delay needs to be as long as the calculated runtime, breaking up the delay into smaller delays that are then equivalent to the larger delay needs to be done. This will be accomplished by setting the small delays to the smallest unit of time that we will be calculating for, which is seconds.

After the process has been run, the pumps will be signaled to stop by using the analogWrite() function again and the valves will be returned to their neutral states by writing the correct value from the custom library to each pin. The processComplete() function will be called and the LCD will prompt the user for an acknowledgement. Following this, the function will return a zero back to loop().

6.2.2.4 Emergency Shutdown Function

The emergency shutdown function will take the name of the file it is modifying as a string input. It will not return any values, so it must be a void function.

The emergency shutdown function will be called after checking that the emergency flag has been set. This function will first stop the pumps by setting the control pins to high for each pump. After this, the function then signals all the valves to turn to their neutral positions by taking the values hardcoded in the custom library and using analogWrite() to set the pins to these values. The emergency shutdown function will then write to the file name it was passed that an emergency shutdown has taken place as well as logging the current time using the RTC library get functions. Some string manipulation will have to be done to get the date and time into an easily write-able format. The function will then close the file so that this information will be saved.

After logging this important information for later review, the function will signal a red LED to turn on by setting the pin it is connected to to high. The function will turn off the green LED by setting the pin it is connected to to low. The function will then play an alert noise via the speaker by writing certain values to the pin this is connected to. We could choose to make this noise play continuously

until the user presses the select button, or simply have it play for a predetermined length of time using the RTC library alarm functions, depending on Helicons preferences.

Lastly the function displays to the LCD an emergency shutdown warning. We can also add instructions, like 'Check SD for more info." The user will have to press select to clear this warning, which will exit the function from a loop monitoring user inputs. After receiving this user input, the function will set the red LED back off by setting the pin to low, turn the green LED back on by setting the pin to high, and optionally turn off the alarm noise if it is still playing. The function will then end. We do not want to reset the global flag variable in this function because even though after it return back to loop(), loop () will basically jump down to the end of it's function so that it can iterate again, there is still time for an external interrupt to set the flag. This would cause the flag to be set heading into the menu function for the next process, which would in turn cause the emergency shutdown to be logged to the wrong file.

6.2.3 Prototype Adaptations

For the final prototype, pin numbers are likely to change from the protoboard. Declaring pin assignments at the top of the code will make this a quick code change from the protoboard to the prototype.

6.2.4 User Manual

This section outlines the user manual for the prototype. It defines the required user inputs to the software as well as gives information on required user actions.

After switching the microcontroller on, the user will be brought to the menu landing pages asking if the user would like to set parameters. Before entering into the menu, it is a good idea for the user to setup the reagents they would like to mix. They have the opportunity to do this up until the system prompts them to acknowledge whether they have primed the system. After entering the menu, the user will be prompted to enter the flow rate for pump one using the two potentiometers. After this the user will be prompted to enter the volume for pump one. Following this, the LCD will display the two selected values and give the user the opportunity to go back and change values or move forward. Moving forward, the user will enter the flow rate for pump two. Following this, the user will be prompted for the volume for pump 2. The LCD will then display the two selected values for pump two and give the user the opportunity to go back and change values or move forward. Moving forward, the LCD will display all four values that the user has chosen and prompt the user for confirmation. The user has the opportunity to go back and change values from this screen. Moving forward, the LCD will prompt the user for acknowledgment that the pumps have been primed. The user should make sure the reagent containers are properly connected to the system at this time, and also prime the two pumps. After this is done, the user should press select and acknowledge the prompt. The system will run on its own at this time until either it hits an error that causes an emergency shutdown or the process completes.

If the process completes, the user must acknowledge that the system is complete for the process to officially be stopped and all data logged. If an emergency shutdown occurs, the user will be alerted via sounds and a red LED, and the user must press select to acknowledge the emergency shutdown.

If the user needs to stop the process while it is running for any reason, they may press the red button located on the user interface. This will then prompt them to acknowledge the emergency shutdown via the select button. The user may safely disengage reagent containers after acknowledging an emergency shutdown, and also after acknowledging a process has been completed.

7. Project Prototype Testing Plan

Hardware and software will be tested both separately and as an integrated unit whenever possible during development. This will allow testing of specific requirements that have to do with either hardware or software separately, as well as checking that the system as a whole fulfills requirements. The following section will outline both hardware and software test environments which discusses where each component will be thoroughly tested in order to meet requirements and properly simulate the final environment our hardware and software will be kept in. The hardware and software specific testing will emphasize the specific procedure and methods used for the main hardware and software components that need to be tested.

7.1 Hardware Test Environment

Many hardware components on our device require detailed and specific testing in order for the project to work as intended. As discussed in section 5 of this paper, our project consists of many subsystems, half of them pertaining to hardware specific modules and the other half of them pertaining to software specific modules. For the purpose of this section, the hardware specific subsystems will be discussed. These hardware specific subsystems include our CCU, which includes our power, processor, and data logging module and our control loop subsystem which includes hardware controlled elements that will be part of our control loop structure like our motors, pumps, and additional peripherals.

It is important to note that as we are not experts in any of the devices that we are working with, aside from the PCB that we will be designing for our specific purpose, we will not be going out of our way to design hardware components and features on our board or in our project that are far out of our scope. This means that hardware components such as motors, AC-DC power transformers, and LCD will not be designed by us and therefore will be purchased and tested by us in a normal consumer fashion. The details of our testing of specific components can be found in section 7.2 below.

7.2 Hardware Testing Overview

This section discusses in further detail the specific components discussed in section 7.1 along with additional peripherals. Included is how and why each section must be successfully tested and prototyped in order for the final outcome of our project to be a success.

7.2.1 Emergency Shut Off Switch Testing

In the case of a catastrophic failure in our system, we will have two possible responses. One is automatic recognition of alerts or alarms from our microcontroller, in which normal procedures are halted an an automated emergency procedure is implemented. In this procedure all pumps are turned off and the valves are set to their default position. The second is a manual shutoff switch in series with the power supply, which would cut power to the system via isolation. The first response has been outlined in a few other sections, so the second will be discussed here.

There are two primary scenarios in which an emergency shutoff switch can be beneficial in an electrical system. One is to physically cut power to a system such that a failure does not cause damage in the form of high currents, thermal runaway, or electric shock. The second is to manually initiate the software shutdown procedure, such that the controller does not continue to attempt to control a system that has undergone a failure.

When considering our necessary shutdown procedure, a switch that just cuts the power is not sufficient. The servo motors still need to run long enough to reset the valves. The only way to do this is to have the manual switch trigger the software emergency shutoff procedure. It's important to note that while the valves will take time to reset, the pumps need to stop immediately upon throwing the switch. So this action should feel like the power is being cut, without actually cutting the power.
In order to test this system we can start with just the microcontroller, and monitor the *Pump ON/OFF* output pin while we throw the switch to verify that the signal is correct. This step will also let us see the delay between throwing the switch and receiving the signal. The microcontroller should be able to send this signal in the microsend scale, so any perceptible delay could be cause to evaluate the software optimization.

If the signal is detected when throwing the switch, the emergency shutoff switch can be tested as part of the general pump interface testing. As the pump connections are tested and verified, the switch can be thrown periodically to determine the time to turn off the pumps. Anything longer than one second will be considered a failure of the emergency shutdown procedure.

7.2.2 Central Processor Module Testing

The central processor module, as it is implemented in our system, refers to the two microcontroller processors that we have running in our system. In terms of testing, most of the limitations/shortcomings can be tested through our prototype arduino boards as structurally they rely on the same base schematics, the main difference being that the arduino board contains a lot of useless peripherals and does not have any way to interface with the pumps. In terms of hardware testing, the main parameters to test in our central processor is the master slave connection along with absolute maximum and minimum ratings.

As previously mentioned the master-slave connection is the main parameter that must be thoroughly examined. The unfortunate part of this parameter is that in order to properly test this, we must have our PCB fully built. If the connection is not properly made then our PCB will fail as the master processors, SAMD21, will not be able to communicate with the slave processor, AMD11. The only way to ensure that our master-slave system on our PCB has proper communication is to properly

route and solder the PCB when the time comes. This along with the routing and soldering of our power module is of utmost importance as these facilitate operation of our PCB.

Another parameter to keep in mind are the absolute maximum and minimum operation conditions. Fortunately for us, this information is already given to us and tabularized by ARM© in the datasheets that accompany the respective processor. Fortunately for us, the minimum and maximum operational conditions for both the microcontrollers is the same, which means that similar operational circuitry can be used. These values listed above indicate they have been thoroughly tested by ARM and that it would be wise to follow them to avoid any breaking of components. As for testing procedure we must ensure that these values remain well within the ranges provided. To ensure that this happens, our PCB will be kept in an enclosure (specific design information found in section 5.1.1) that is electrically stable. What this entails is that the PCB and components soldered to the board are enclosed and will not remain in contact with any stray or loose wires that are in contact with our system. The temperature of our microcontroller will remain at a stable temperature slightly above room temperature (due to an increase in temperature from operation). Again the PCB will remain in our designed enclosure so a large temperature fluctuation due to environmental changes (if the cart system were to be moved to a new facility) will not play a major role as the maximum operating temperature is well above the boiling point of water. In addition to avoid any complication in power input and output maximus that occur from the 5V micro-usb input, we using the capacitor banks provided by the open-source arduino schematics that use these same microcontrollers. The schematics for these capacitor banks can be found in Figure 42 below.

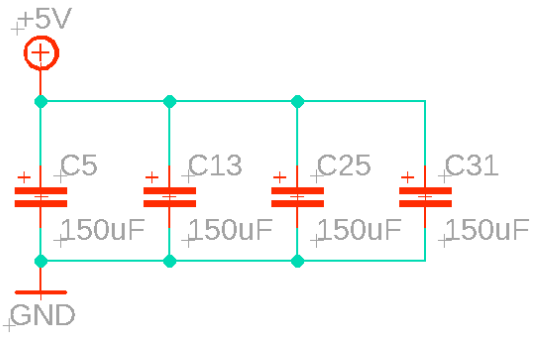

Figure 42: Arduino Capacitor Bank

7.2.3 Power Module Testing

As discussed in section 5.2.1, our CCU will contain 2 main voltage signals. These signals are the 3.3V used to power our microcontrollers and the 5V received as an input from our micro-usb power source and to power our servos/motors. As mentioned in section section 7.1, due to the danger of getting hurt physically testing this product as it is implemented in our PCB will be difficult. The two methods of testing we will be us using are the same methods we will be using for our data logging module testing, testing the power module as it is implemented in the arduino prototype boards and checking operational testing conditions located in the devices' data sheet.

Our first method of testing is regarding our prototype arduino boards. The main DC-DC 5V to 3.3 volt conversion occurs in the arduino MKR motor carrier board which contains our slave microcontroller and connector peripherals. This board along with our master microcontroller, the

MKR Zero, will be our testing prototypes as they contain the same processors and power regulators that we will be using. According to arduino, the board contains both reverse current protection (in the form of two P-Channel MOSFET connected in series shown in Figure 43. The orientation of the MOSFETS and directions of the diodes will prevent current from flowing back into the input power pin of our 5V voltage regulator.

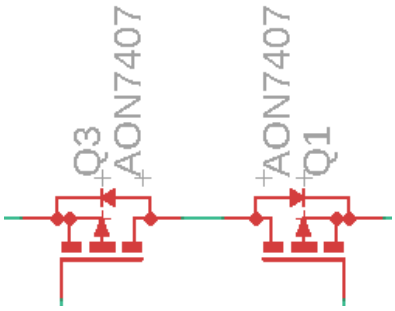

Figure 43: Reverse Current Protection P-Channel MOSFETs

This along with the automatic temperature shutdown protection for the DC motor drivers ensures that the 5V input voltage DC-DC regulator will remain protected in the form of a temperature and current malfunction. To properly and safely test the automatic driver shutdown we conducted a test in which we connected four DC motors (the maximum we will have in our final system) to our MKR motor carrier and left it running until the temperature of the drivers reached a peak temperature of 80 degrees celsius. This temperature is well below the maximum operating temperature of drivers at 150 degrees celsius. It is safe to assume that the final storage and operational conditions of our PCB inside of its enclosure will not reach this temperature.

Another important operating condition to take note of is the operational efficiency of the power module. The 5V buck regulator shown in section 5.2.1 is the general schematic footprint for the Semtech TS30013 which has 3 operation currents. For our design, we will be using the smallest of the three currents (1A) to avoid any high current complications that may occur. This current value is more than enough for an output and operation of the regulator. As stated, the efficiency of the regulator is dependent on both input voltage and operational current. Through Semtech's device testing, they have identified that given and Iout of 1A and the switched output voltage of 3.3V (which is the exact output voltage we need to switch to from 5V) the devices efficiency drops linearly as input voltage increases. This however is perfect for power cases such as ours in which the input voltage to our system is 5V from our micro-usb input.

7.2.4 Data Logging Module Testing

The data logging module, as it pertains to our system, serves as a way to log data regarding the chemical profile system we plan to implement in our system. What this entails is collecting data regarding sensor data along with the data and time when the chemical mixing process is started stopped. In addition to this, the date and time of the emergency shutdown procedure must be recorded. While the mechanics of these tests reside more in the software domain, the test scenarios/cases are something that must be created in the hardware domain by alteration of the operating conditions of the pump system. There are also additional hardware testing concerns that

relate to the data logging module and how it physically interfaces with our board and the micro-SD card that must be done as a precaution.

The first set of tests, and also the most straightforward to validate, are the tests related to how our data logging module physically interfaces with the board and the choice of micro-SD card. As discussed in section 5.2.3: Data Logging Module, our data logging module will communicate to our master microcontroller (SAMD21) through a set of 12 pins. Five of these pins will actually be communicating with our microcontroller (NSS, MOSI, MISO, SCK, CD1 pins) and the other pins will be routing signal planes into the micro-SD reader (3.3V and GND). To ensure that our micro-SD reader is working as intended the first step of testing would be to ensure that both the schematic and PCB has, respectively, routed and soldered these pins correctly. In addition to the choice of micro-SD is also important as the data collection module is responsible with collecting sensitive information like faults, profile runtime errors, and other sensitive data. While there are monetary incentives to buy cheap micro-SD cards, it would be wise to not go this route and buy trusted and warranty backed brand name retailers such as SanDisk, Lexar and Samsung to avoid any possible errors that occur. Read and write speeds are also important to us as we are dealing with sensitive information, and to test whether the read/write speed is accurate to what the purchased micro-SD card indicates we will write an error log to the system and attempt to read the same error log back. If the read/write speed matches what was promised by the purchased micro-SD card then we are set to begin our second set of SD card testing scenarios.

The second set of tests includes accurate and precise data logging as they specifically pertain to our system's emergency shutdown procedure. As previously mentioned this step can only be undergone after the micro-SD card and micro-SD card reader have properly been checked and wired respectively. The emergency shutdown procedure can be due to a variety of factors, very few of these factors our outside of control, therefore it is our responsibility to account for as many of these scenarios as possible. These scenarios can be attributed to both hardware and software problems, in this case the hardware problems and testing cases will be discussed. The hardware causes for emergency shutdown can be attributed to the motors and pump. The manual emergency shutdown, as discussed in section 7.2.1 and 7.4.4, in unison with the data logging module must be tested as well.

Once the motor has been properly calibrated, tested, and implemented into Helicon's system, we test to see whether the motor's valve position automatic emergency shutdown is working. To test this we must first create an 'error testing' profile in which the motor does not properly open/close the valve (for testing purposes we will not be using actual hazardous chemicals as this is extremely dangerous and incurs many safety violations). Given that the 'error testing' profile tells the motor to open/close the valve within the 'go-zone' parameters we will manually operate the motors until the valve resides in the 'dead-zone' (180-270 degrees). Next we press the red button and follow the procedure identified in section 7.4.4. If the data logging module correctly logs the intended position and the error position then the data collection module can successfully log position related valve errors. In a similar testing procedure, we can test whether or not the motors are responsive (instead of manually operating the motors we can disconnect the motors from our system and see if the data collection module logs that the motors are unresponsive).

The pump error data collection is another main function of our data collection module and requires its own set of testing parameters. Of these parameters the main one to test is pump flow rate. The testing procedure for this is extremely similar to that of the motor in which we instruct the data collection module to collect pump flow rate information that exceeds or is less than a specific range. Next we manual set the pump in two separate tests to exceed and be less than the predefined flow rate range. Next we press the red button and follow the procedure identified in section 7.4.4. If the data collection module notes this and logs this information then the tests are seen as a success. This same testing procedure involving an operating range and testing outside of said range can be done for sensor error logging.

7.2.5 Motor Testing

The motor as it is implemented in our project is one of the most important parts of our project as it is part of our project that will be facilitating automation of Helicon Chemical Company's chemical pump system. As outlined in section 5, our CCU will be controlling our motor through our feedback loops implemented through our PCB hardware and software. For the purposes of this section only the hardware testing specifics will be discussed as the pertain to the motors of our system. There are many tests that must be done thoroughly to ensure that our motors not only work as intended, but also have the longevity that Helicon Chemical Company desires. These tests include, but are not limited to:

1. Motor Driver Testing
2. Motor Coupling Testing
3. Motor Torque Testing
4. Motor Position Testing

7.2.5.1 Motor Driver Testing

The quantitative testing for the drivers of the motors motors is a fairly straightforward task. As emphasized in paragraph two of section 7.2 Hardware Testing, it would be highly advisable for us to use resources and tools that are already exist in similar products for our project. In addition, the MKR Motor Carrier open-source Arduino board contains premade schematics for motors used for similar purposes to our project. For this reason we are using the given motor-related schematic parts as the prototype arduino board confirms the intercompatibility between parts. These parts include the pre-chosen and pre-tested drivers (DRV8871 and MC33926), shown in figure 44 and figure 45, and the 3-pin 5V compatible connectors.

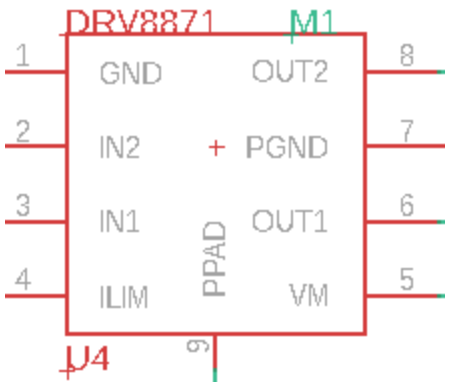

Figure 44: DRV8871 Driver

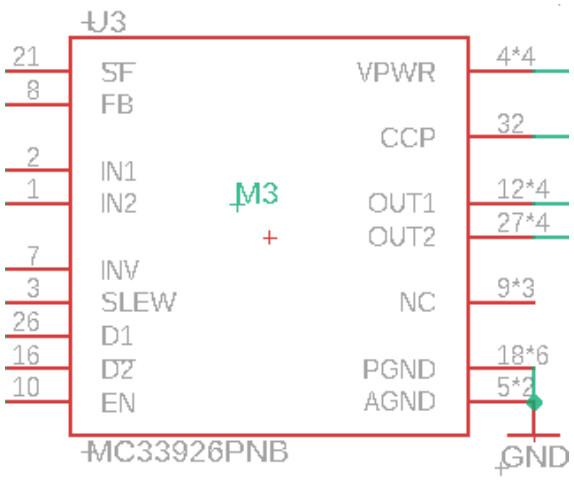

Figure 45: MC33926 Driver

To ensure that there are no issues in terms of driver compatibility, input voltage, current draw, and other quantitative testing issues that may occur during our many other motor-related tests or operation, we aim to use motors/servos that are compatible with these parts. The two drivers listed above will be what our motors are going to be using as their primary drivers and in terms of input voltage, the motors must be compatible with the 3-pin 5V connector used in our schematic. To properly test that these quantitative measures are compatible and functional, we must first, before physically connecting the motor to our system, ensure that the datasheet of the purchased motor says that it is compatible with the DRV8871 and MC33826. In addition we must also check in the datasheet that the motor relies on a 3-pin 5V connector to function. Once quantitative measures are checked, we will connect the motors to our PCB and turn the motors on, preferable running them for a full minute to ensure no issues, and then check to see if the motors can be turned off properly with not to large of a delay as precision is important (more details in 7.2.5.4 Motor Position Testing).

7.2.5.2 Motor Coupling Testing

The motor and its coupling with the valve is another aspect of the motor that needs to be tested. This coupling is key to the motors operation in our design. One one hand, if the motor is not coupled tight enough, then it will be both hazardous to the system and be difficult to precisely control. The hazard that this causes is by virtue of the motor slipping out of its coupling with the valve while it is on, which could damage the valves themselves or damage the tubing used to carry the chemicals through the system. In terms of precision control, the more loose the motor-valve coupling is the hard it will be to precisely control the motor positioning. In addition, if the motor-valve coupling is set too tight we may destroy Helicon's custom built valve threads. These valve threads were created as a means to fasten a nut on top of the lever used currently by the system, but they can create an easier time coupling the motor to the valve and creating a sturdy connection. We aim to not modify Helicon's system so that they can temporarily go back to the lever based system in case one of the motor's needs to be replaced in the distant future.

For testing purposes, a spare valve has been provided to us by Helicon. Our testing procedure includes coupling the motor our purchased 6mm to 10mm motor coupling. Once the motor coupling is securely fastened, the coupling can then be attached to the valve and secured through the custom built threads. It is important to not strip the threads located on the coupling and inside Helicon's valves, so this process will be done with precision, accuracy, and a relatively slow speed. Once the motor is securely coupled with the valve we will do 'wiggle test', which involves gently moving the motor side-to-side to see how much excess degrees of movement the motor has. It is also important to keep the motor's axis of rotation at the same plane as the valve to avoid any complication with torque related testing. If the motor-valve coupling relatively secure (not too much excess degrees of movement), then we can secure the custom motor mounting bracket mentioned in section 5. Given that this procedure is done successfully the motor should not have too much 'wiggle' and should be securely coupled with the valve.

7.2.5.3 Motor Torque Testing

In addition to the driver test discussed in section 7.2.5.1 and the motor coupling test discussed in section 7.2.5.2, the motor torque test is penultimate motor test. That is due to the fact that the driver compatibility and coupling mechanics of the motor must first be tested out and mastered before torque of the motor can be properly tested. As discussed in section 5.3.2.2 the motor-gearbox combination that we have selected allows for up for up to 300 Newton Meter torque which is outputs more than enough torque to rotate our valves. This however does not immediately validate our torque testing as the angle between the lever and the arm must be perpendicular to ensure maximum torque output. This relationship can be seen by the general torque equation represented below.

$$\tau = rF\sin\theta$$

Once the testing of the motor coupling with the valve is successfully completed and the mounting brackets for the motor are added, the motor will be stable enough in operation that we can effectively say the angle between the lever and arm is perpendicular. This means that the purchased motor's out of box torque reading can be accurately trusted, as the variables decreasing the torque in the system have been minimized the best they can be.

In terms of testing of this motor's torque as it is implemented in our system, we simply have to give the motor power and see if it can successfully turn the valves. If the motor is able to turn the valves without any struggle, then the motor's torque test is successful. The amount of time required to turn

the valve a specific degree is unimportant to us for the purposes of this project as Helicon has specified that the precision and accuracy in opening and closing the valves is of utmost importance. Also as long as the motor can actually provide enough torque to open and close the valves, then there are a couple ways to compensate speed. So long as we use a separate gearbox from the motor, we can always purchase a stronger gearbox to increase the maximum torque allowing for faster movement. We can also adjust the flow rate of the pump to increase or decrease the amount of a given chemical flowing through the valve proportionally to the speed that the valve opens/closes.

7.2.5.4 Motor Position Testing

In section 5.1 Figure 11 shows an image of Helicon's cart. Notice that there are physical barriers imposed on the ball valve handles. This is because if the valves ever get to these positions it would have disastrous effects on Helicon's equipment. Reagents could get sucked into the vacuum pump and no one will have a good time. These physical barriers prevent such an event. Figure 46 below shows our working zone (Green) and our danger zone (Red).

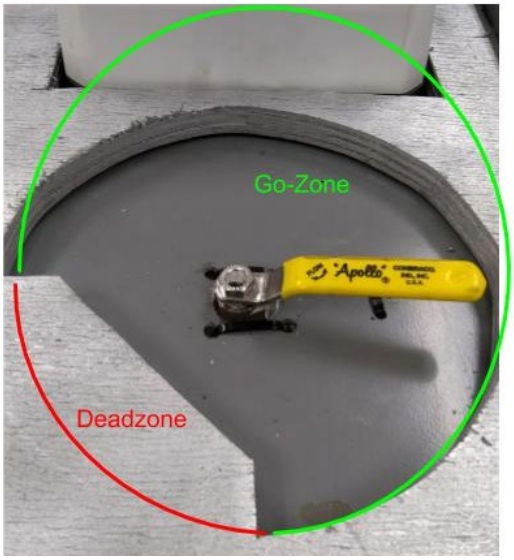

Figure 46 - Helicon Ball Valve Deadzone

Once the ball valves are modified, however, this safeguard is no longer valid. The torch of protection will be passed to our system, and along with it the trust of Helicon. There will be a roughly 90 degree deadzone where the motors can never enter. Once the motors are mounted they will need to be tested so we know exactly which positions are in this deadzone, and hardcode our software to only run in the acceptable positions.

The testing procedure will look like this: the valve is manually placed at the very edge of the deadzone, the servo is connected at this position and powered on. We then note the motor position from the feedback signal, then actuate the motor to move by one positive degree. If the motor moved toward the deadzone, then the initial position **plus** 90 degrees is the deadzone, and the initial position **minus** 270 is hardcoded as this motors possible positions. If one positive degree of movement sent the motor away from the deadzone, the bolded signs above are reversed.

The unfortunate part of this is that, due to Helicon's valve positioning, the deadzone for each of these valves is in a different quadrant. Therefore this calibration will need to be done for each of the four motors, and our programming will need to have very clear identifiers for each motors, such that motor position commands always reference the correct motor, with the correct 270 degree "go-zone".

7.2.7 LCD Testing

Seeing as the LCD implementation purpose in our project is to display output and input information, its testing must be done through both qualitative and quantitative testing.

Qualitative Testing: The appearance of our LCD must be visible and able to display all of the characters in each row that it allows. The LCD we have chosen is the RioRand's 20x4 white character on blue background LCD, which is a LCD display with 4 rows of 20 characters. In addition, this LCD allows manual soldering of a possible potentiometer (which will be included in our final design) to pins 1 and 2 of the LCD's pins to allow for manual control of contrast of the white characters on the display. The addition of a potentiometer to the display will allow us to fix any possible errors that occur during testing resulting in characters not displaying. To fully test that the display is able to show all 4 rows of 20 characters a simple test code can programmed to run on the display. This test code should include the essential characters such as all 26 letters of the alphabet (having both uppercase and lowercase does not matter for our specific purpose but for aesthetic purposes all letters will be the same case), all 10 digits, common special symbols, and a basic phrases such as "hello world". Figure 47 below is a basic diagram showing what our character test case should look like, where each cell represents one character.

H	E	L	L	O		W	O	R	L	D	!								
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z		0	1	2	3	4	5	6	7	8	9			
,	.	/	:	;	(	)	*	^	-	+	=	@	~	[	]	|	□		

Figure 47: Visual Representation of Successful Character LCD Test

In addition to this we must also have a test chase where every single character cell is occupied with a character to ensure that all of the character cells are functioning as intended. The best character to use for a test like this would be the ASCII symbol 254 '□' as on the LCD display this will fill up the entire cell, ensuring that every bit on interface is accessible. Figure 48 below is a visual representation of this test, as the '□' represents a cell being completely filled.

□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□

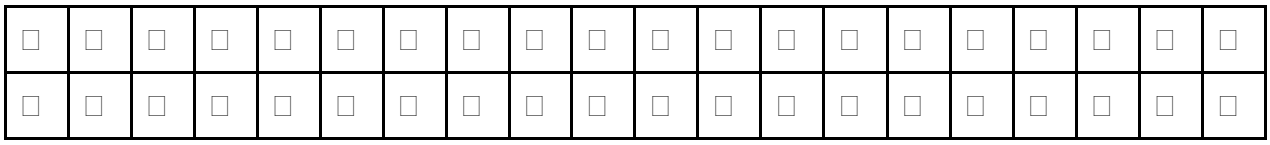

Figure 48: Visual Representation of Successful Square LCD Test

Quantitative Test: In addition to ensuring that the LCD is displaying characters at an appropriate brightness and that it is able to display all needed characters, we must also test quantitative measurements so that our LCD performs at the level that we want it to. Some of these measurable quantities include refresh rate and input voltage

In regards to refresh rate, for our specific project, this is not a major concern as the main purpose of the LCD is to display inputs (what chemical profile to run and ad what flow rate) and that the profile that ran is completed. For this reason, displaying said chemical profiles and flow rate faster on our LCD, while this would be preferable for the user, is not 100% necessary. If we do decide that this is necessary there are other displays to choose from that also fulfil our other requirements while only being slightly more expensive.

The second quantitative test for us is whether or not the input voltage of the LCD matches to that of the main PCB. This is important as having a voltage that differs from what we are using in our PCB would mean that we would have to create a brand new DC-DC conversion circuit on our board for this sole purpose. Having a matching voltage to what we have on our PCB means that we have less work to do in terms of modification of our main PCB design. For our specific design we have two voltages to work with on our board and that is 5V which comes in as an input form our micro-usb connection and 3.3V which comes from a 5V to 3.3V DC-DC conversion module on our board. Upon research of the pins of the RioRand's 20x4 white character on blue background LCD, the input voltage required is 5V which matches the voltage that is being utilized on our board.

7.3 Software Test Environment

To increase ease of testing, a paired down hardware testing rig will be implemented for as much of the software testing as possible. The use of an oscilloscope to test output signals will be vital to test software driven outputs. Testing purely software capabilities, such as the GUI for the tactile screen, can be done with just the screen component.

The task of testing the software becomes more difficult when multiple hardware components must be used to test accurate responses by the software. For example, testing that the software can detect a problem in the signal returned from the pump will require the use of the pumps and being able to reliably recreate a pump 'error'.

We will utilize the senior design lab for equipment such as the as the oscilloscope, power supply, and function generator. We will also use it for miscellaneous tools such as multimeters, wire strippers, pliers, etc. in order to access the microcontroller pins to test software outputs and fake inputs for the software to read.

7.4 Software Specific Testing

The main focus of software specific testing is to see that the software behaves as it should and to find any weird edge cases in the program logic. It also serves to iron out any bugs in the software that might be exposed during testing. While traditional software development might employ the use of automated unit testing to check that software requirements are being met, since our software resides on an embedded system, we will instead be testing by hand.

Besides making sure the program compiles and is able to be written to the microcontroller, each component of the software should be tested. That includes the user interface sections, process code, and emergency shutdown functionality.

7.4.1 User Interface Testing

This section defines all the specific tests for the software components that deal with the user, either through the menu system or grabbing acknowledgments from the user via button presses.

7.4.1.1 GUI

Menu Accessibility Test: A user must be able to navigate between all eight menu pages. To test this, run the microcontroller, which should begin the menu sequence at the initial menu page. Check that pressing the select button transitions the LCD to the next menu page, and pressing the select button again transitions the LCD to the next, and so on through all eight menu pages.

Then test to make sure the user can backtrack through all 8 pages of the menu. To test this, run the microcontroller and use the select button to navigate to the second menu page. From there, hitting cancel should return the user back one page. Repeat this procedure for the seven menu screens following the initial menu page, which does not allow the user to press cancel.

This test is successful if a user can navigate to each of the eight menu pages in order, and backtrack through the last seven menu pages. Anything less is considered a failed test.

Menu Variable Test: A main concern is that the value being displayed by the menu should be what is actually getting saved by the software. To test this, run the microcontroller and navigate to the first of the four menu screens that allow the user to set a variable. Select variables for each of the four variables that can be set, making a note of the value that are selected for each. We will put in a line of code for testing that prints these values to the screen after returning from the menu function. Verify that the display values are the same as the ones set. Another way to verify this is to check that the correct values have been written to the log since the log gets updated after the menu() function returns.

A particular question that testing could answer is ‘how fast can the potentiometers be manipulated and still save the correct variable value’?’. Testing this will be harder as the LCD will be updating faster than a human could probably detect, so it would be hard to record the value being displayed on the screen with what the system is saving. It is possible though, due to the order of instructions in the code, that a user could see a value, quickly manipulate the potentiometers to get a new value

and press select before this value was updated to the screen. Testing to see how long a user should wait before pressing select after manipulating the potentiometers will have to be done to ensure that the value being displayed on screen is the same as that being stored by the code.

The user should also be able to scroll through the full range of variables currently set from 1ml to 4000ml using the two potentiometers. Test this for each of the four menu screens that allow the user to set a value. Also test that the software is correctly incrementing when turning either potentiometer since they both have different granularities.

These tests are successful if the variables being displayed that a user selects are the same ones being saved, and if a user can increment through the full range of values for each variable by the appropriate increments for each potentiometer. If these conditions are not met, then the test is failed.

GUI Acknowledge Test: The software will prompt the user for an acknowledgement via the LCD for three occasions, a primed system, a completed process, and an emergency shutdown warning. To test this for a primed system, run the code on the microcontroller and navigate through the menu sequence, setting any values for the variables. Upon exiting the menu, the system will prompt for a user to acknowledge priming the system, and will not move on until the user has done so. The user should be able to press confirm to acknowledge this. To test this for the completed process, run through a complete process to make it to the end of the program. Make sure that the acknowledge screen stays up until the confirm button is pressed. To test the acknowledge for the emergency shutdown, utilize the red emergency shutdown button on the user interface. After pressing this, the GUI should display a screen that informs the user of the emergency shutdown that should not go away until the confirm button is pressed.

This test is successful if all acknowledge screens remain up until the user acknowledges them. Otherwise, if the acknowledgements do not remain up, or do not even pop up at all under these testing conditions, then the test is failed.

7.4.2 Chemical Process Function Testing

This section outlines the tests that are particular to the chemical process function. In particular, it defines tests for the software interfaces to the motors and pumps.

Motor Testing: Using an oscilloscope, check that the software is outputting the correct signal to each of the valves, for all three positions the valves will be in: neutral, reagents flow, inert gas flow. These three positions should be tested for each of the four valves as they all have different values. To test this, set an oscilloscope up to one of the pins that a motor interface with. Make note of which motor it is to have the correct reference values found in the custom library. Verify that the oscilloscope is measuring the correct signal as compared to the values found in the custom library. Having the physical motor will be helpful to test that the software is moving the valve to its precise location.

This test is successful if the correct value is being sent to each pump for each of the three positions. If any of the values are incorrect, the test is considered failed.

Pump Testing: Use an oscilloscope to test the basic commands going to the pumps. To test that the start/stop command is being properly set, run the code on the microcontroller, and navigate through the menu sequence, making a note of the values selected, until you get to the user primed acknowledgement. After acknowledging this screen, the software will signal the pumps to start. Verify the signal for each pump with the oscilloscope. The next command to the pumps will be to set the flow rates, again using an oscilloscope to verify the signal being set for each pump by checking with the values that were set in the menu sequence. Verify that the correct signal is going to the correct pump.

Testing to make sure the software can pick up a pump error is a bit trickier. Since the software checks the signal being sent to the pump for flow rate against a feedback signal from the pump, we will be faking this feedback from the pump using a signal generator. That way we can set the feedback signal to something that does not match the flow rate being fed to the pump and reliably create a situation that should raise a pump error. This can be tested at any point the pump is started and getting a flow rate command from the microcontroller, so any time after the acknowledge system is primed up until the process completed acknowledgement pops up. To test this, trigger a pump error with the method described above and check that an emergency acknowledgment screen pops up. Alternatively, the log may also be checked as it should have a timestamped emergency shutdown message logged when a pump error occurs.

This test is successful if the correct start/stop signal is being sent to the pumps and if the correct flow rate is being set for each pump. This test is successful if the system registers a pump error after the conditions outlined above are met. If the correct signals aren't being set, or if the system does not catch the pump error, then the test is failed.

7.4.3 Emergency Shutdown Testing

Special testing of the emergency shutdown system will be done as it is considered a critical function of the project. The SD card must be checked for appropriate log data and times of emergency shutdowns, both initiated by the user pushing the big red button and also software instigated based on a pump error. The software must set the pumps to stop and signal the motors to set valves in neutral positions, as well as engage an alert. Integrated testing of software and hardware will be critical for this requirement since it is a vital function that touches on all systems in the project.

Initiate Shutdown Test: Test that the emergency shutdown sequence can be initiated by a pump error or user input via the red button. To test this, first try pressing the red button, and then check that the emergency function has been tripped, either by checking pump status, valve positions, red LED status, or the log. Using the testing setup defined in the pump test, create a pump error and then check that this causes the emergency function to be tripped by checking any of the things mentioned above.

This test will only be considered passing if both methods produce a flag that then allows the program to enter the emergency shutdown function.

Emergency Procedure Test: This is a large integrated test that checks that the logging is being done correctly, the pumps and valves are being set correctly, and the alert system reacts correctly when

an emergency shutdown function is triggered. It requires the whole system to be configured, including the hardware components. While these software outputs could be tested using readings from an oscilloscope instead of the actual hardware, this is a good time to make sure the hardware is behaving as expected.

To run the test, produce an emergency shutdown by pressing the red button during runtime of the process. The LCD should then display an alert, the red LED should be lit, the pump set to stop, and valves returned to neutral positions. Verify that all these events have taken place, and then remove the SD card to read the log to verify that the log was updated correctly with the date and time the emergency shutdown occurred.

The test is considered successful only if all required actions occur, and anything less will be considered a failed test.

7.4.4 Log Testing

Particular testing of the log capability of the software needs to be done because it is a direct request by Helicon. The log should accurately record the date and time processes are started and completed, along with the values that are being set for the flow rate and volume for each pump. The log also records any emergency shutdown functions with the date and time. Besides these values, the file name creation logic needs to be tested as well.

File Name Testing: To test that the software is creating new file names based off of the date and time, run the microcontroller for a couple of iterations through the process. Each iteration should create a unique new file name, and this can be verified by checking the SD card on a computer. The names themselves should all have the same first 4 digits since they would have been created the same month and day, but the last 4 digits should distinguish between the files as they represent the hour and minute. Checking the file names against the current runtime by outputting this variable to the LCD using a line of code meant only for testing purposes.

This test is considered successful if a unique name is generated for every process and if the name is generated by using the month, day, hour, minute model. Not generating new files or naming them incorrectly will result in a failed test.

Process Time Log Test: To test that the process start and stop times are accurately recorded in the log, run through a process on the microcontroller, making a note of the time when the pump is signaled to turn on and off, which signals process starting and ending. After the process is complete, remove the SD card and check the log times that are recorded against the times from running the process.

The test is successful if the log times and test recorded times match to within a few seconds. If they differ by more than sixty seconds, the test is considered failed.

Variable Log Test: Recording accurate flow rate and volume for each pump is critical. Test this by running through a process, making a note of selected values in the menu. After exiting the menu

and returning to the pump primed acknowledge on the LCD, remove the SD card and check the log. The values saved in the log should exactly match those selected during the menu phase.

The test is considered successful if the logged variables match those selected during the menu phase. If they do not match exactly, the test is considered failed.

Emergency Shutdown Log Test: The log needs to track when emergency shutdowns take place. It should be the last item in a log if that process encountered an emergency shutdown. To test this, either push the red button during a process or use the method described in the pump test to create a pump error, making a note of the time. Remove the SD card and read the log file, checking that the emergency shutdown has been recorded in the log. Next, check the time recorded in the log against that recorded when the emergency shutdown was instigated.

This test is considered a success if the emergency shutdown was logged and the time logged is within sixty seconds of the time the emergency shutdown was caused. Any discrepancy over sixty seconds is considered a failed test. Failing to log that an emergency shutdown has occurred is also considered a failed test.

8. Administrative Content

This section outlines the administrative content, including an outline of the major project milestones for senior design 1 and senior design 2. It provides a budget discussion and quick table reference for the parts list. It ends with a section describing future goals for the project.

8.1 Milestone Discussion

Senior design 1 milestones focus mainly on deliverables in the form of paper report and presentation requirements. Senior design 1 also has the start of prototyping and testing individual components to work towards a completed design. Senior design 2 milestones will focus more on prototyping and also developing a finished product that fulfills the requirements. Table 7 below shows the milestones for senior design 1 while Table 8 following that show the milestones for senior design 2.

Senior Design 1 Milestones Table

	Start Date	Stop Date	Status	Lead
Divide and Conquer	9/20/19	10/4/19	Completed	Group 22

60 Page Document	10/4/19	11/1/19	Completed	Group 22
Select Parts for Protoboard	10/4/19	11/1/19	Completed	Group 22
100 Page Document	11/4/19	11/15/19	Completed	Group 22
Order Protoboard	11/4/19	11/25/19	Completed	Group 22
Final 120 page Document	11/25/19	12/4/19	Completed	Group 22

Table 7

Senior Design 2 Milestones Table

	Start Date	End Date	Status	Lead
Order Parts	01/06/20	02/01/20	Not Yet Started	Group 22
Prototype Build	01/06/20	02/01/20	Not Yet Started	Group 22
Preliminary Software Build	01/06/20	03/01/20	Not Yet Started	Group 22
Revise Design	02/01/20	03/01/20	Not Yet Started	Group 22
Re-Order Parts	02/01/20	03/01/20	Not Yet Started	Group 22
Refine Software	03/01/20	04/01/20	Not Yet Started	Group 22
Revise Design Stage 2	03/01/20	04/01/20	Not Yet Started	Group 22
Integration Testing	03/01/20	04/15/20	Not Yet Started	Group 22
Demo/Pitch	04/15/20	04/30/20	Not Yet Started	Group 22

Table 8

8.2 Budget and Finance Discussion

Helicon has given a soft commit for funding which should become more definite as design details and planning solidify. As long as justification can be given, funding is likely to be provided for a required component. Some components will be provided by Helicon, such as the pump and valves,

which will save funds. Effort will be made to utilize free samples from manufacturers for prototyping and testing. Further discussion will need to occur about funding for components necessary to meet stretch goals that are not essential to Helicon's needs but necessary for senior design requirements.

After considering all of our options and removing the ideas that either do not correlate with senior design requirements or Helicon requirements then the budget is definitely on the lower end of the spectrum. We now know that Sensors, valves and the higher echelon of CCUs being the PLCs are all exempt from our design. Our true budget is thus predicted to be under $600 unless there are unforeseen hurdles once implementation of the design is done. After research concluded, the enclosure, the motors, and the LCD will be our biggest budget investments and getting those wrong will be a large setback for the project financially speaking. The CCU being a PCB is our largest time sink as reordering and soldering all the parts onto it again will take a large amount of time. These components in particular must be assessed correctly the first time to avoid setbacks. The other materials are easier to replace and can be ordered quickly thanks to familial contacts and today's modern online ordering services. Below is a quick reference table for rough estimates of all component prices.

Budget Table

Item	**Estimated Cost**

CCU	$10-$350
Sensors	$50-$150
Valves	$40-$100 (Or provided by sponsor)
Misc. Circuitry Material (wires, transistors, contact blocks, etc)	$40-$60
Pump	Provided by sponsor
Push Buttons	$10-$30
Power Supply	$20-$50
Breaker	$30-$40
Enclosure and Din Rails	$60-$120
Din Rails and Backplane	$40-$80
LCD Display with/without Keypad	$100-$200
Motors	$80-$120
Motor Stands	$20-$50
Total:	$500-$1,300

Table 9 - Quick Reference for the proposed budget

8.3 Looking Forward

This system is designed for a small part of what is a large manufacturing process. As such, there is tremendous room for growth. Improvements could be made to increase the flexibility of the system

so that it works in a wider range of chemical manufacturing situations. Adding in the ability to mix more chemical components with different temperature requirements would also be incredibly useful to Helicon. Giving the user the ability to monitor the system from anywhere via a web application would be helpful for managing production on a fully automated system, and would be necessary for larger systems. This step would also present some level of security concerns that would need to be addressed as well.

Given more time, more money, more resources, a fully automated system that does not require human supervision from start to finish is possible, and is already being done for large scale manufacturing plants. A small scale, easily customizable version is therefore more than possible and presents an opportunity for any enterprising individuals with an engineering background.

Open Collector Outputs

Some remote outputs on this drive (Tachometer, Local/Remote, and Alarm) are "open collector" type outputs and cannot be wired in the same manner as relay outputs. An open collector output is not isolated and must be configured differently than a relay output.When the open collector output is active, the output is effectively switched to earth ground and if improperly terminated could result in damage to the drive and/or external equipment.

Recommendation

When connecting to open collector outputs, the output should be connected to a current limiting resistor and then to a positive supply source which is less than 28V DC. Typically this would be connected to a 24V PLC input (see Figure 3-14).

NOTE: when using the 24V supply on the interface connector, current draw must be limited to 150 mA.

Motor Running Contacts

On this drive the motor running outputs (Normally Open and Normally Closed) are relay outputs and do not require the 24V positive supply source.

NOTE: it is not recommended to attach 120V supply lines to relay contacts!

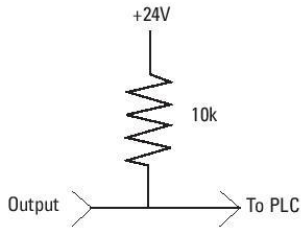

Figure 3-14. Terminating Open Collector Outputs to a PLC