Control System and User Interface for Hydraulic Tricycle

By

Iain Brearton

Final Report for ECE 445, Senior Design, Spring 2015

TA: Jacob Bryan

05 May 2015

Project No. 2

Abstract

This project involves designing, testing, and building a fully-functional user interface and control system for a hydraulic tricycle. The control system was built from scratch with an ATmega328p microcontroller, communicating with the user interface over I2C and interfacing with electromechanical valves that controlled a hydraulic circuit. The tricycle needed to be able to transfer hydraulic energy from the pedals to the rear drive wheel, store hydraulic energy, and ultimately compete in the 2015 Parker-Hannifin Chainless Challenge competition. As a result, the electronics needed to be able to interface with the mechanical systems and present an easy-to-use interface.

Done as a senior design project, the control system electronics were designed from scratch after research on how to robustly interface with external devices. The user interface electronics were a combination of commercially-available parts and custom circuitry as necessary. Main circuit blocks included a simple power supply and battery, the control system, the user interface, and pressure sensors.

Contents

1 Introduction 1

1.1 Objectives 2

2 Project Context and Background 3

2.1 Mechanical System Description 3

2.2 Hydraulic Circuits 3

3 Design 4

3.1 User Interface Design 4

3.2 Control System Design 6

3.3 Power Distribution Design 7

3.4 Pressure Sensors 7

4 Cost 8

4.1 Parts 8

4.2 Labor 8

5 Conclusion 9

5.1 Safety considerations 9

5.2 Ethical considerations 10

5.3 Future work 10

5.4 Acknowledgements 11

6 Figures 12

Reference 14

Appendix A Iterations of the User Interface 15

Appendix B Control System Schematic and PCB 18

Appendix C Requirement and Verification Table 22

Appendix D Proportional Valve Research 24

Appendix E Microcontroller Code 29

Appendix F Hydraulic Fluid Paths in Manifold 35

1 Introduction

Parker-Hannifin, a fluid power systems company, hosts an annual competition for the design of a chainless tricycle. A control system, user interface, and electrical power are needed for UIUC's entry into this competition. Previous control systems for these tricycles used either commercially-available systems [1] or discrete logic [2]. A new and unique approach to this problem is a custom-built control system and user interface. This has the possibility of being tailored to the end users, who are both accessible and on campus.

For this project, the Mechanical Engineering team members were clients. They approached me with a problem definition and a set of requirements. The Mechanical Engineering team was iterating their designs as I designed the electronics, so flexibility and quick turnaround times on new ideas was a requirement. Throughout this process, iterative designs were presented with feedback from the Mechanical Engineering team and changes to the problem definition as the mechanical system was developed.

Automation and control systems are of personal interest, so working on the control system and operator interface for a real product was a significant personal and professional benefit.

In this report, the context of the project will be discussed followed by the objectives, design requirements, design, and design verification.

1.1 Objectives

At the beginning of the project, the project goals, benefits to the end user, and system functions and features were examined to ensure the project would have an impact and stay on track. The goals listed below were set.

Goal 1. Electro-mechanically control flow of fluid in hydraulic circuits.

Goal 2. React to user input and give feedback to rider.

Goal 3. Stay within competition guidelines.

Goal 4. Help create a competitive or winning hydraulic tricycle.

Goal 5. No control failures during competition (happened in 2013-2014).

The benefit to the Mechanical Engineering team is automatic control of electromechanical valves, which improves the tricycle's ease of use when compared to manually-actuated valves.

More specifically, the functions and features provided are below.

Feature 1. Require user interaction to change hydraulic circuits (for safety of rider).

Feature 2. Interpret sensor data and ensure operation is within safe parameters.

Feature 3. Visually display current accumulator pressures.

Feature 4. Support functioning of mechanical systems on tricycle, in addition to above.

(i) Power and control valves.

(ii) Work around physical constraints on tricycle, such as dimensions and placement.

2 Project Context and Background

Having briefly mentioned the Objectives in section 1.1, it is now worth completing the picture. The 2014-2015 Chainless Challenge is a competition sponsored by Parker-Hannifin with the goal being to develop a fluid-powered tricycle, preferably without a chain. The competition consists of an endurance race, an efficiency race, and a sprint race. Points are awarded for placing in races as well as for various design features.

2.1 Mechanical System Description

A brief explanation of the mechanical system that is being controlled is necessary to give context to the rest of this document. Fluid power is provided by the rider and is input at the pedals. The fluid power can either be stored in the accumulators or sent directly to the hydraulic motor, as determined by the valves in the manifold. The hydraulic circuits to support this are listed below in section 2.2. The accumulators have valves that can either be setup to accept incoming fluid or expel outgoing fluid to the motor. Incoming fluid can either come from the rider (charge paths) or from the motor (regenerative braking path). Figure 1 is an overview of how components are connected. All figures and tables are in section 6.

2.2 Hydraulic Circuits

There are 7 hydraulic circuits that need to be supported by the electrical system, listed below. A set of diagrams generated by Ian O'Leary is attached at the end in Appendix F that show the various fluid paths in the latest mechanical design.

1. Direct Drive Path (Default)
2. Coasting Path
3. Charge Low Pressure Accumulator and Coast
4. Charge Low and High Pressure Accumulators
5. Discharge Low Pressure Accumulator
6. Discharge High Pressure Accumulator
7. Regenerative Braking Path

3 Design

The system designed was not the only solution to the problem presented, but was an attempt at one such design, heavily influenced by the designer and team's collective preferences.

The system as a whole was divided into four blocks, with the Power Distribution, Sensors, and Control System blocks physically located under the rider's seat and the User Interface block mounted on the center of the handlebars. Figure 2 shows a block diagram of the tricycle's electrical components and the electrical connections between each. Similarly, the software flow chart is in Appendix 6. The control system schematic and PCB are both in Appendix B.

We will discuss the user interface first as it had the greatest impact on the user (rider), followed by the control system it connected to and the control system's peripherals. A simplified description of the requirements and verifications for each block are presented in the sections for each block below. Further information about the numerical requirements of each system and how they were verified is available in Appendix C.

3.1 User Interface Design

The user interface module is the main way to facilitate user interaction and display information to the user. The user needs to be able to select between 6 operating modes, view the operating mode on the Character LCD, and view sensor data on the Character LCD, at the very least. The user interface was designed with Ian O'Leary from the Mechanical Engineering team. The user interface is mounted on the handlebars for accessibility and convenience. Pictures of iteractions of the user interface design are in Appendix A.

3.1.1 Inputs

Our main goal was to avoid having 7 buttons for each mode as a large quantity of buttons means that not all of them can be located in easily accessible places. Since the charging and discharging states had variations for different accumulator selections, we decided to use a selector switch to choose the accumulator(s) and one button each for charge, discharge, and regenerative braking. Indicator lights in the buttons provide feedback to the user that the control system has received their input, possible since the button indicators and contacts are separately powered. All buttons are momentary, since 'toggle' functionality is achieved in software as needed. Switch circuitry includes pull-up resistors.

Inputs and outputs are communicated with the microcontroller via an I^2C/TWI port expander IC. To verify that the inputs were working, an I^2C master was connected to the bus and the inputs were read with different buttons pressed and depressed. When buttons were depressed, the value read by the I^2C master for that input should correspond to logic HIGH. When buttons were pressed, the value read by the I^2C master for that input should correspond to logic LOW.

3.1.2 Outputs

Other than the indicator lights in the buttons, described in the Inputs section, the user interface outputs information with a 20x4 character LCD. The character LCD displays the current pressure of both accumulators and text indicating the current operating mode. The character LCD communicates with the microcontroller via I^2C/TWI using a commercially-available "LCD backpack".

The indicator lights were required to be controllable by an I^2C master. To test this, the I^2C master sent commands to blink each indicator light and a visual check confirmed the indicator lights changed states. The character LCD was required to be able to display arbitrary ASCII text in any position on its 20x4 screen. This was also tested with an I^2C master.

3.1.3 Main Power Switch

A main power switch was also desired, but it would get no use outside of turning on and off the system, so it was placed in a less accessible spot to make room for the other buttons and switches. Further details on the buttons and character LCD are below.

3.2 Control System Design

The control system's microcontroller is responsible for sending/receiving all signals to other modules. The valve control circuits are responsible for turning on and off the electromechanical valves given a logic HIGH or LOW from the microcontroller.

3.2.1 Microcontroller

The microcontroller is responsible for reading the button state on the user interface, sending display data to the character LCD, controlling the indicator lights, reading the pressure sensors, and turning on/off the valves. For rapid prototying, an ATmega328P is used since it is supported by the Arduino environment (the 328P is used in the Arduino UNO) and it is available in a 28-pin DIP, which is through-hole. The code for the microcontroller is located in Appendix E.

The microcontroller had to be able to communicate with all of the I^2C slaves (the port expander and the LCD backpack) as well as output logic signals that the valve control circuits would correctly react to. For details on the voltage tolerances, see Table C.2 in Appendix C.

3.2.2 Valve Control Circuits

The electromechanical valves require 12V at 1.2A in order to actuate. Since an ATmega328P can only source 200mA across all pins, either transistors or small power relays are required. Power relays offer the advantage of cheaply and effectively isolating the input from the output. In addition, since turning on the valves will create a momentary drop in supply voltage, future separation of the power supplies including their grounds may be desired. Thus, power relays were chosen. The power relays currently in use require 45mA to turn on, which a pin on the ATmega328p cannot sink or source, so a transistor is being used as a buffer, reducing the microcontroller pin current to approx. 1.23mA. All outputs for valves have indicator LEDs to aide in troubleshooting, though the LEDs can be left unpopulated if that feature is not desired.

The valve control circuits had to connect the "positive" sides of the valve coils to 12V based on the logic input from the microcontroller. The valves have no polarity, so "positive" is a term internal to the valve control circuit used to differentiate from ground, which is connected to the other side of the valve coils. Connecting the valve coils to power involved actuating the relay, so this was also tested.

3.3 Power Distribution Design

A battery is used to power the electromechanical valves and the custom electronics. The electromechanical valves require 12VDC and since they are the highest-current devices, the battery was chosen based on them. Since the control system and user interface electronics require 5VDC, a simple power supply was required.

3.3.1 Battery

The battery voltage was chosen based on the highest-current devices, the electromechanical valves, requiring the battery voltage to be 12V. Additionally, the battery needed enough capacity to power the entire system for 1 day of the competition. While exact power usage is unknown since it depends on the number of valves actuated, it has been estimated that 20Ah is the bare minimum, based on 8 hours usage at 2.5 Amps. These numbers were chosen based on a 6 hour competition and the current requirments of the valves, which use 1.2A each. Since the system remains in direct drive mode for most of its' operation and direct drive mode doesn't require any valves to be turned on, an overly-high estimate of 2.5A continuous draw was used. A lead-acid battery was used due to tradeoffs between availability, cost, capacity, and volume. The battery that was chosen was a Universal Battery UB12200, with a 22Ah capacity.

The lead-acid battery was tested by discharging at a rate of 2.4A for 6 hours and needed to remain above the discharged voltage of 11V. More details are available in Table C.2 in Appendix C.

3.3.2 Power Supply

The power supply steps down the 12V from the battery to 5V. While the valves do not have a polarity, it is important to protect against reverse-polarity for the control system and user interface electronics, so a protection circuit needs to be included. In addition, the power supply will need to continuously provide up to 200mA, so a power supply that can provide 2A continuously provides a very large safety margin that is useful for such a critical system on the tricycle.

With the requirements in the previous paragraph, a 7805 voltage regulator was used with passive cooling. A heatsink is attached to the 7805 and the ground pin (pin 2) is attached to the ground plane, providing enough cooling for this purpose. The reverse-polarity protection was done with a power PMOS, based on a TI Application Report [3]. While there is a voltage drop across the MOSFET during forward operation, the voltage drop is before the voltage regulator, so the microcontroller still receives a constant voltage. With the currently installed regulator, this is about 4.96V regardless of the number of relays actuated (the largest current load using 5V). During reverse operation, the power MOSFET creates a voltage difference of 0V or just over 0V, ensuring no damage to the 5V electronics.

The power supply was operated at the 2A current rating specified for 1 hour and stayed within its specified voltage output range of 5 $\pm$ 0.25 V. The power supply also properly protected against reverse polarity, outputting 0.00V on the +5V pin when reverse polarized.

3.4 Pressure Sensors

The pressure sensors used were provided by the Mechanical Engineering team. They output 0 - 4.5 V which was linearly proportional to input pressures from 0 - 6000 PSIG. They required +12V power. For testing, there were no calibrated analog gauges available, so the reading at atmospheric pressure was checked to be within 20

4 Cost

The total cost of the project is approximately $30109.90, with labor and parts costs, for 1 unit. Sections 4.1 and 4.2 show the calculations for the parts and labor costs, respectively.

4.1 Parts

The parts list is given in Table 1. The retail cost is shown as well as the actual cost with free parts subtracted. Overall cost is totalled at the end.

Table 1: Parts List and Costs

Item	Part Number	Quantity	Retail Cost ($ ea)	Actual Cost ($ ea)
Microcontroller	ATMEGA328P-PU	1	3.74	3.74
Power switch	100SP1T1B4M2QE	1	2.11	2.11
Fuse and holder	Unknown	1	5	0.00
Reverse polarity transistor	IRF9540PBF	1	1.72	1.72
Capacitors, 10uF 20%*	ESK106M050AC3AA	2	0.16	0.00
Voltage regulator	LM7805-ACT	1	0.67	0.00
LEDs (red, green, blue)*	Cree C503B-XXX	11	0.15	0.00
Status LED transistor*	Low current NPN, ie. SS8050	1	0.25	0.00
Valve control transistors*	SS8050DBU	9	0.29	0.00
Resistors 5%*	CF14JTxxxx	23	0.10	0.00
Terminal block header, 2 pos	284518-2	17	0.67	0.67
Terminal block plug, 2pos	284507-2	17	1.28	1.28
Ceramic resonator*	CSTLS16M0X53-B0	1	0.40	0.40
Capacitors, 22pF 20%*	FK28C0G1H220J	2	0.29	0.29
Pushbutton switch*	B3F-1000	1	0.35	0.00
Diode*	1N4004	9	0.11	0.00
Relay*	T77S1D10	9	2.28	0.00
Character LCD, 20x4	Adafruit 198	1	17.95	17.95
I2C port expander	MCP23017-E/SP	1	1.44	1.44
I2C LCD backpack	Adafruit 292	1	10.00	10.00
I2C ADC	ADS1015	1	9.95	9.95
Illuminated pushbuttons	Adafruit 1439	4	1.50	1.50
Selector switch	AutomationDirect GCX3370-22	1	14.50	14.50
Battery	UB12220	1	43.96	43.96
Wire	–	1	10.00	0.00
		TOTAL	**190.09**	**145.50**

Items with asterisks were available free from the ECE Service Shop.

4.2 Labor

Throughout this project, an estimated 500 hours were spent. At an hourly rate of $60.00 with a multiplier of 2.5, the total labor would be $75000. For an engineering consultant, the average fee is about $110 per hour, according to an IEEE survey from 2004 [4]. Thus, the labor rate chosen seems quite conservative, but the multiplier compensates.

5 Conclusion

In summary, a custom control system and user interface were built for a client, the Mechanical Engineering team. Within their constraints, a suitable implementation was designed, built, and tested in the intended real-world environment. The tricycle and its riders won 2nd place overall as well as placing in the top three for 10 out of 11 categories. The monetary awards benefitted both the university and the individual team members.

A discussion of this project would be incomplete without safety considerations, ethical considerations and future work, all covered immediately below.

5.1 Safety considerations

This project both interacts with a human being and controls the bike that a human being rides. Moreover, the system controls the flow of hydraulic fluid that is used to store energy and can be discharged by actuating the correct electro-mechanical valves.

An alternative to an electromechanical control system is a system that is entirely mechanically-actuated with levers. However, depending on the order the valves are opened and closed, the user could release stored energy in an unintended way (ie. discharge both accumulators instead of just one). Since the tricycle rider needs to change states several times during a race, while steering a 200 lb bike, the risk is relatively large. An electrical control system turning on and off electromechanical valves eliminates this risk and can be programmed to reject unsafe changes in valves, based on the sensor and user input it receives. In this way, it reduces the risk of operator error by limiting the choices of valve states. However, the electrical control system processes data faster than a human can react, so if it outputs an unsafe combination of valves, the tricycle may abruptly accelerate or decelerate without warning to the user. In addition, care must be taken that loss of power to the control system or all valves does not create an unsafe condition.

With this in mind, the valves and hydraulic circuit have been designed by the Mechanical Engineering team so that their default state (that in which no valves are powered/actuated) is the Direct Drive circuit, which does not involve storing or releasing energy. The user will have the ability to remove power from the control system and the valves in order to return to this state if the control system fails, via the main power switch on the user interface. In addition, a safety relay has been added. One side of the load goes to the battery and the other goes to the other relay loads (see schematic in Appendix B). In this way, IFF the safety relay is on, the valves can be actuated by turning on the other 8 valve relays. If the safety relay is off, turning on the other relays will not actuate the valves. When designing the microcontroller code that switches valve states, care was taken to examine undesirable combinations of valves that may occur when switching between states.

5.2 Ethical considerations

This project is closely related to several items of the IEEE Code of Ethics, namely:

1. to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;
3. to be honest and realistic in stating claims or estimates based on available data;
5. to improve the understanding of technology; its appropriate application, and potential consequences;
7. to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others;

Item 1 is very relevant, for the reasons discussed in the previous section, Safety Statement.

Item 3 is relevant because making false claims regarding the capabilities of the control system or user interface could lead the rider to trust the control system and user interface with their life in a situation they would not have been comfortable with when presented unbiased facts. Making false claims also creates moral and legal liability for negative outcomes of using the user interface and control system.

Item 5 is important equipment that fails could have somewhat catastrophic failures and it is important to understand the technical limitations of the technology being used, how to use it safely within those limitations, etc.

Item 7 is important because I am working with a team of mechanical engineers, and while they do not contribute directly to the reports, they create content I choose to include in my reports. Their work must be understood as their own with proper citations.

5.3 Future work

The electronics systems work as intended. In the future, the following additions or changes would further improve the end product:

- The code could be improved, as explained in Appendix E.
- The PCB could be updated so the status LED workaround was no longer required, as explained in Appendix B.
- If desired by future teams, the work already done on proportional valves could be extended, as explained in Appendix D.

5.4 Acknowledgements

I would like to acknowledge the following parties for their help with my Senior Design project:

- Professor Scott Carney, TA Jacob Bryan, and the rest of the ECE 445 course staff, for providing resources and help.
- Project advisor Professor Hsiao-Wecksler and TA Conrad Smith, for guiding our cross-disciplinary team.
- Sandy Parker and her co-workers at Parker-Hannifin, for organizing and hosting the Chainless Challenge 2015 competition.
- The National Fluid Power Association, for their generous grant which enabled us to create this project.
- David Ruxton at Hydraforce, for technical help with the electromechanical valves.
- Mark Smart of the ECE Service Shop, for answering PCB questions and making the PCBs.
- The ECE Service Shop, for providing advice and parts.

6 Figures

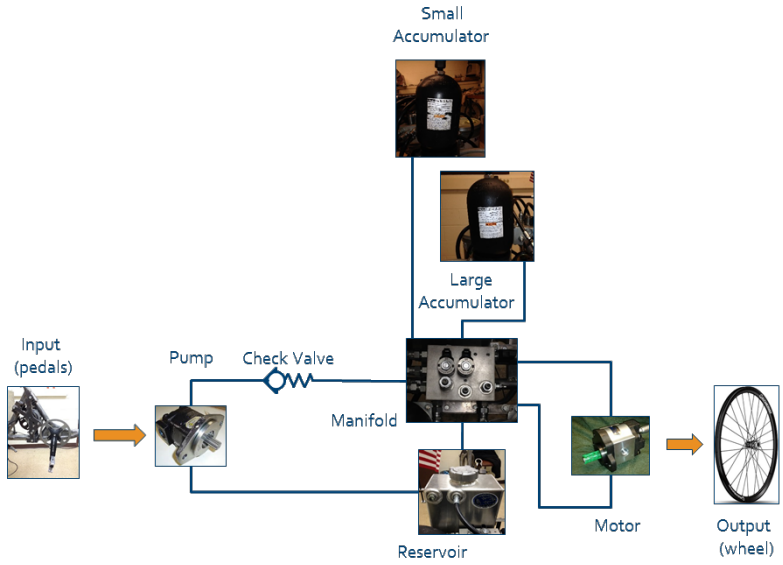


Figure 1: The mechanical system requiring electronic control. Blue lines represent hosing for fluid.

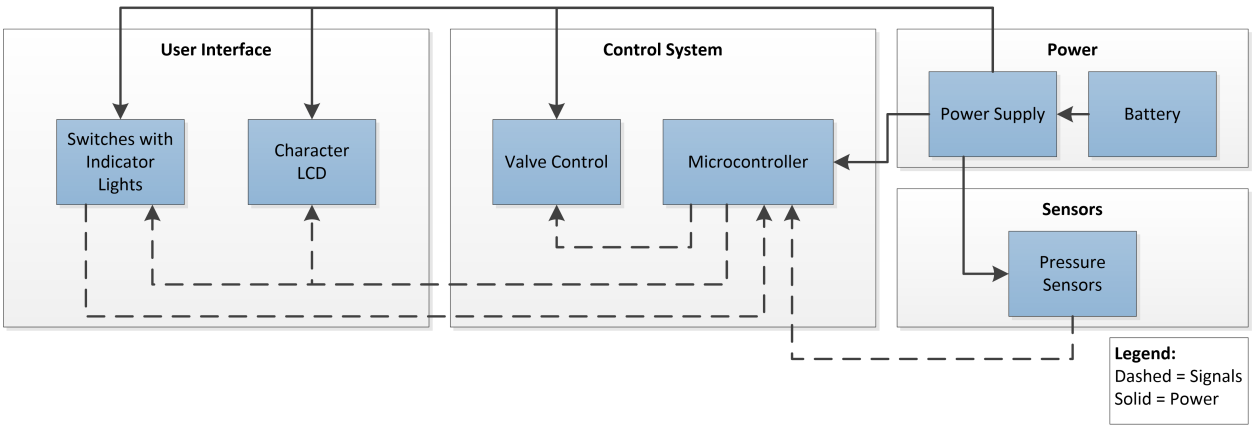


Figure 2: Block diagram of electrical system.

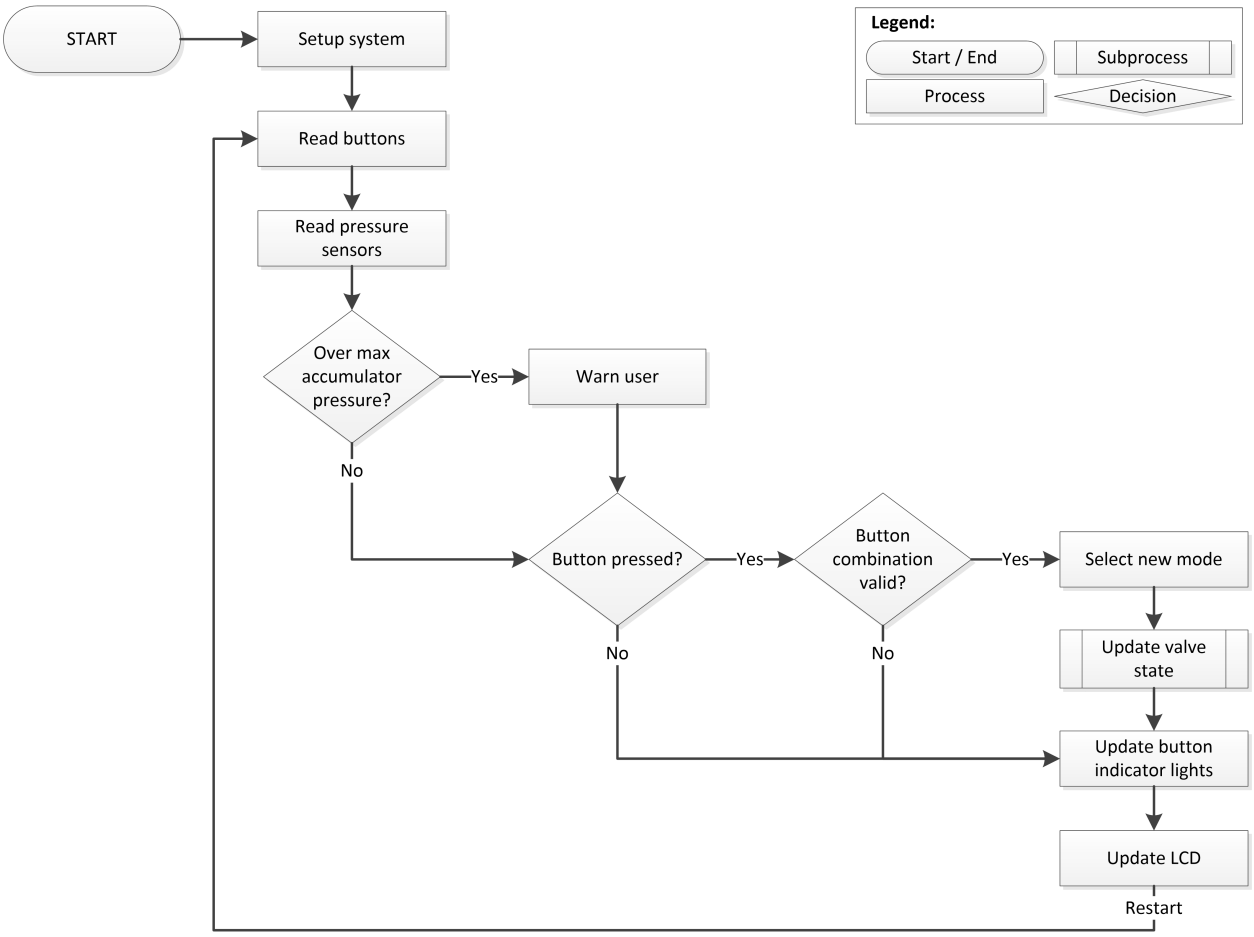


Figure 3: Software flow chart for the control system's microcontroller.

References

[1] J. G. et al., "Parker Hannifins Chainless Challenge 2012," web page, apr 2012, accessed 06 May 2015. [Online]. Available: https://engineering.purdue.edu/ABE/InfoFor/CurrentStudents/SeniorProjects/2012/GeskeLamneckSparenbergEtAl

[2] A. B. et al., "Final Report," web page, dec 2009, accessed 06 May 2015. [Online]. Available: http://deepblue.lib.umich.edu/bitstream/handle/2027.42/86206/ME450?sequence=1

[3] Texas Instruments, "Reverse Current/Battery Protection Circuits," web page, jun 2003, accessed 06 May 2015. [Online]. Available: http://www.ti.com/lit/an/slva139/slva139.pdf

[4] IEEE USA, "Profile of IEEE Consultants, 2004," web page, dec 2004, accessed 06 May 2015. [Online]. Available: http://www.ieeeusa.org/business/files/2004FeeSurvey.pdf

[5] K. Fodor, "Microcontroller Input Protection Techniques," web page, mar 2012, accessed 05 May 2015. [Online]. Available: http://www.kevinmfodor.com/home/My-Blog/microcontrollerinputprotectiontechniques

[6] S. Martin, "Arduino I/O Protection," web page, accessed 05 May 2015. [Online]. Available: http://www.designer2k2.at/mods/arduino/63-arduino-io-protection

[7] Rugged Circuits LLC, "10 Ways to Destroy An Arduino," web page, accessed 05 May 2015. [Online]. Available: http://www.ruggedcircuits.com/10-ways-to-destroy-an-arduino/

[8] teho Labs, "KiCAD Tutorial," web page, accessed 05 May 2015. [Online]. Available: http://teholabs.com/knowledge/kicad.html

[9] C. Rohrbacher, "Quick KICAD Library Component Builder," web page, accessed 05 May 2015. [Online]. Available: http://kicad.rohrbacher.net/quicklib.php

[10] J. Hill, "KiCad Hierarchy Tutorial x01," web page, aug 2010, accessed 05 May 2015. [Online]. Available: http://uhawebpage.hartford.edu/kmhill/suppnotes/KiCadDia/KiCadHierarchy01/KiCadHierarchy01.pdf

[11] Mithat, "Stitching layers between copper zones in KiCad," web page, nov 2011, accessed 05 May 2015. [Online]. Available: https://biro.wordpress.com/2011/11/25/stitching-layers-between-copper-zones-in-kicad/

Appendix A Iterations of the User Interface

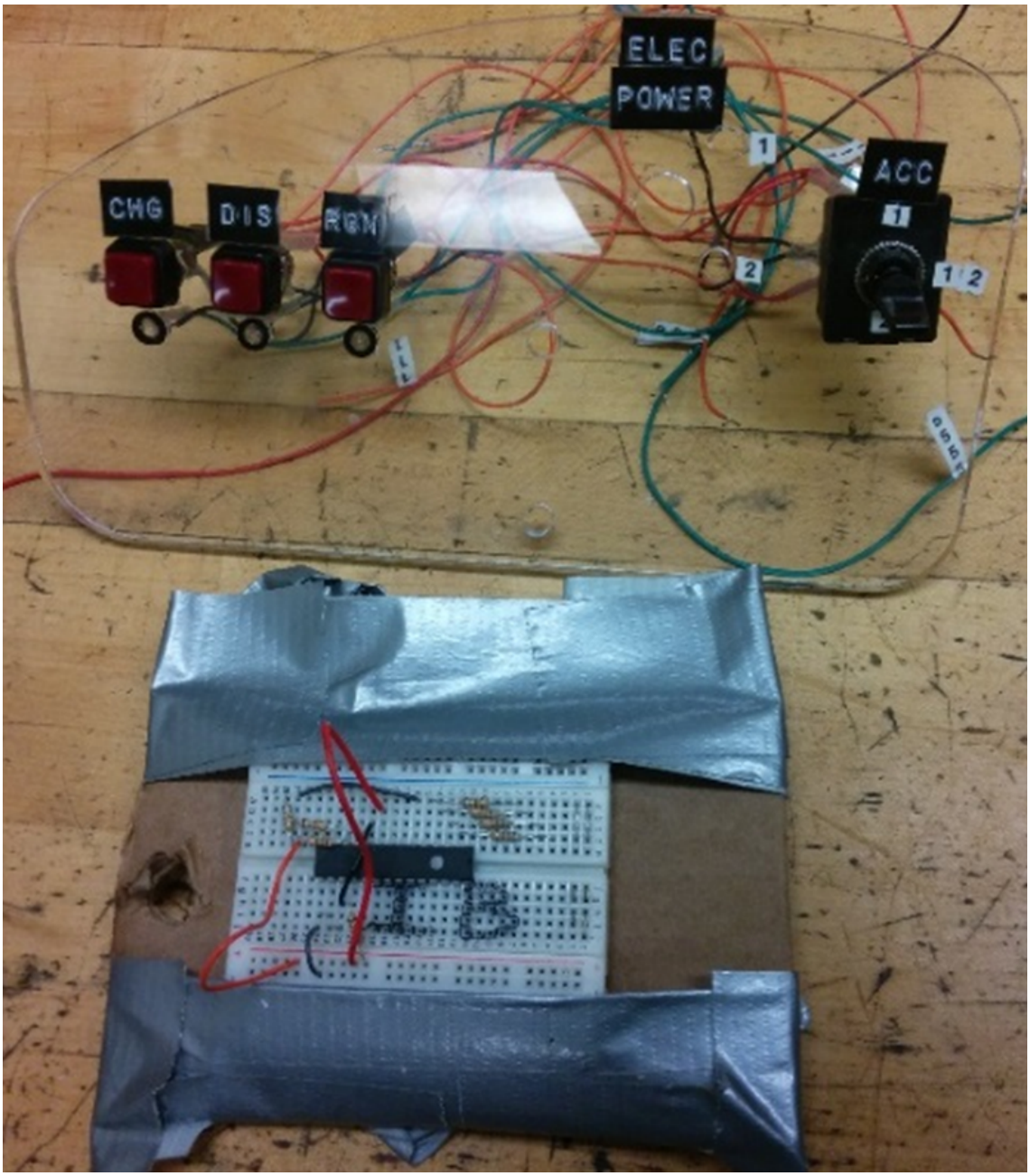


Figure A.1: First iteration of user interface. Needed external protoboard with I^2C port expander.

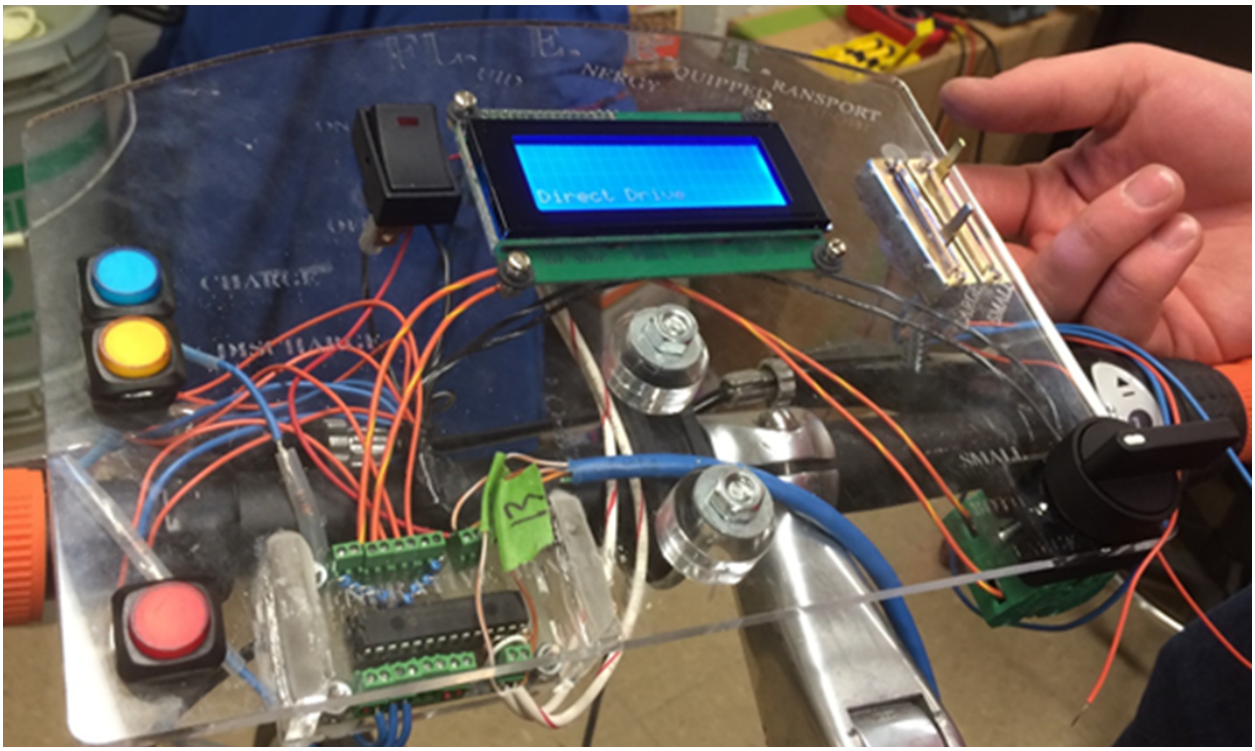


Figure A.2: Second iteration of user interface. Added PCB and LCD support.



Figure A.3: Final user interface.

Appendix B Control System Schematic and PCB

The CAD files for the electronics were made in KiCAD. The schematic was created and updated as features were prototyped, ensuring the greatest chance of first-time success when a PCB was made from the schematic. As it turned out, only the status LED did not work with the PCB as designed, but reuse of the status LED traces in a different arrangement produced a working status LED. Thus, revision 0.6 of the PCB was the only version ever made. The final versions of each file are revision 0.6.

The circuit schematic for the control system is documented on the next page. The control system PCB follows.

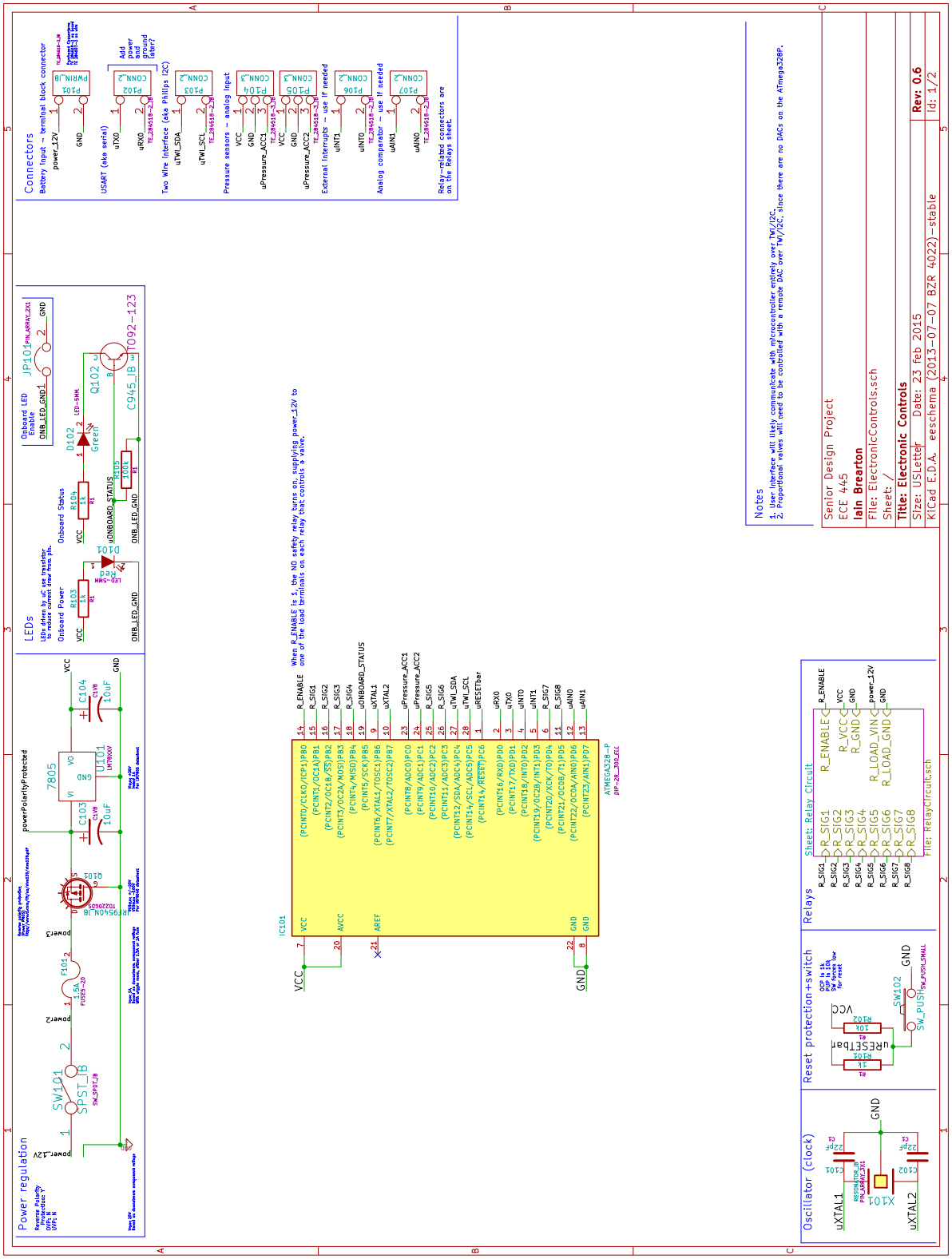


Figure B.1: Final schematic design for control system, page 1.

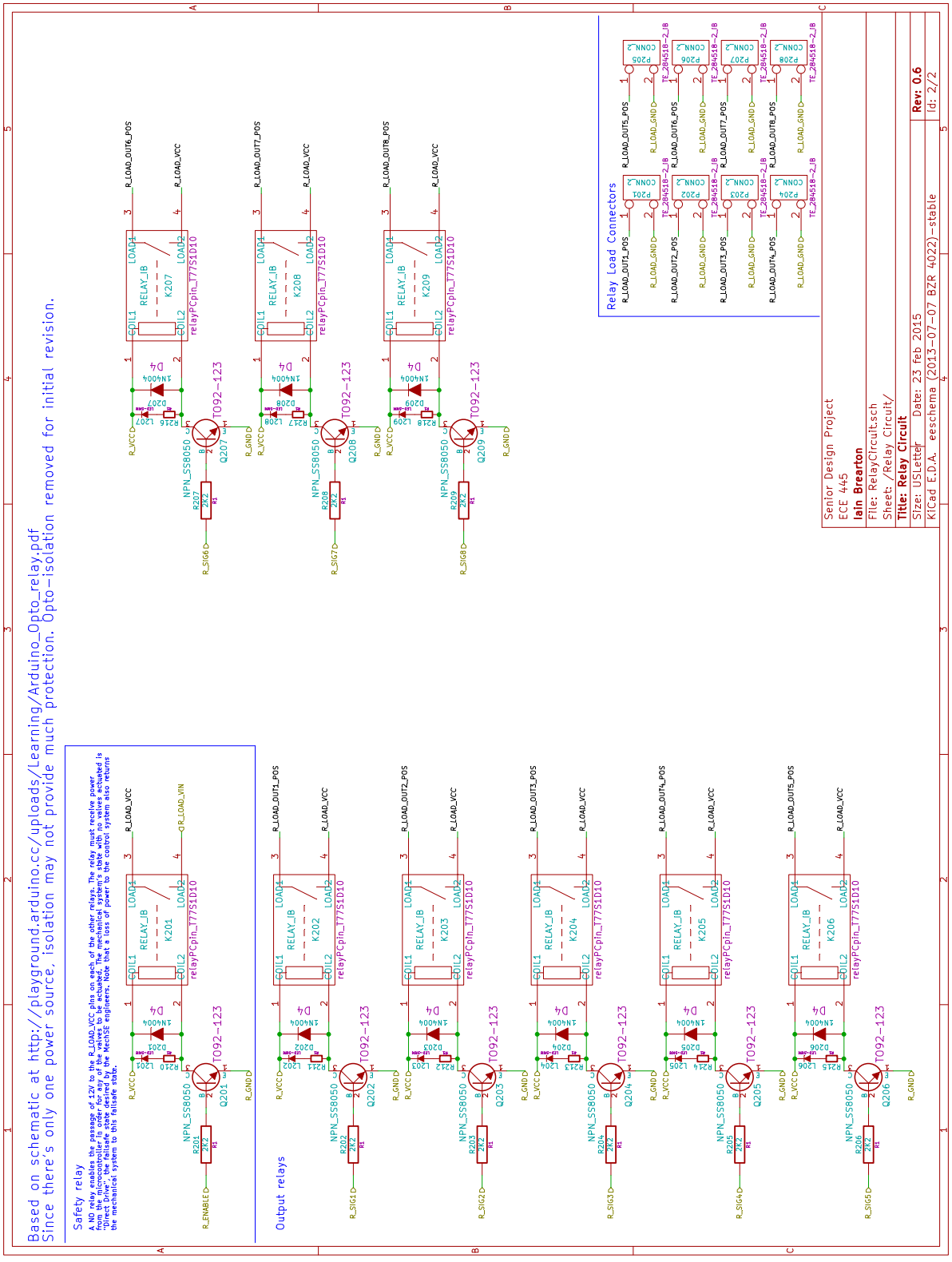


Figure B.2: Final schematic design for control system, page 2.

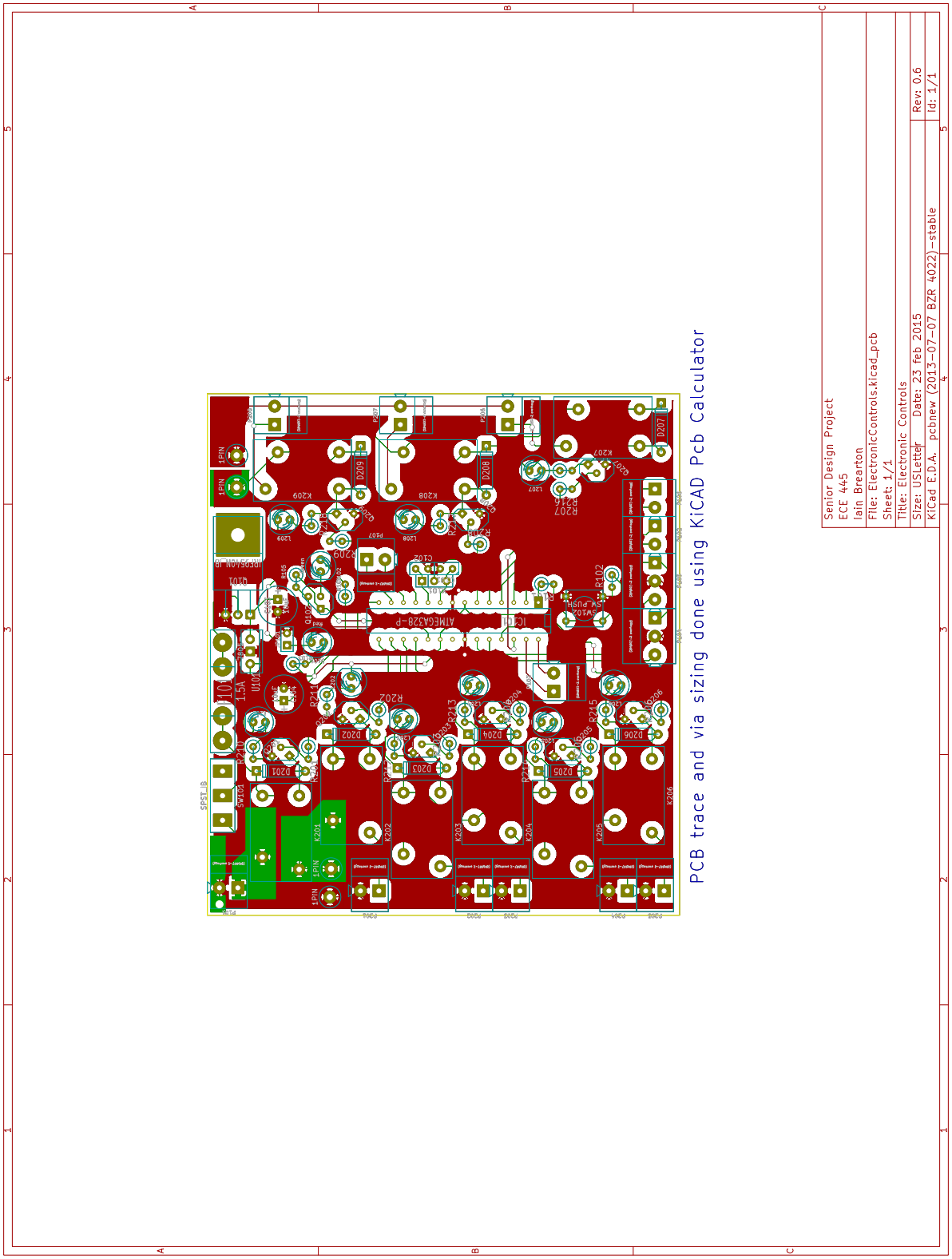


Figure B.3: Final PCB design for control system.

Appendix C Requirement and Verification Table

The requirements defined for the electronics as well as the methodology for verifying each requirement is listed in Table C.2. As noted in the right column, all requirements were successfully verified.

Table C.2: System Requirements and Verifications

Block	Requirement	Verification	Verification Status (Y or N)
1. Switches with indicator lights	a. HIGH or LOW button state is read over I2C	a. Connect I2C master to bus and read from the I2C port expander with all buttons unpressed. Press one button at a time, continuing to read the state over I2C. Data should indicate logic HIGH when buttons are unpressed and logic LOW when buttons are pressed. 5 button presses and depresses should be properly registered.	Y
	b. Indicator lights are controlled over I2C	b. Start with all indicator lights off. Connect I2C master to bus and write to I2C port expander the data required to turn on each indicator light, then turn off each indicator light. Only the indicator lights specified in the I2C packets should change state on each write. Each indicator light should be tested twice in this manner.	Y
2. Character LCD	a. Character LCD displays ASCII text from microcontroller	a. Connect I2C master to bus and send ASCII text to LCD backpack over bus. Ensure ASCII text sent fills character LCD and that all characters are displayed where expected.	Y
3. Microcontroller	a. Microcontroller can communicate with all I2C slaves	a. Connect microcontroller to I2C bus as master. Verify microcontroller can perform all I2C tasks mentioned elsewhere in this table read switch states, control indicator lights and read pressure sensors.	Y
	b. Output a low signal 25% below the turn-on voltage (1VDC) of the valve control circuit (ie. 0.75VDC)	b. Program the microcontroller to send a low signal. Check voltage is 25% below threshold voltage of transistor (0.75V) relative to ground using DMM in parallel with microcontroller.	Y
	c. Output a high signal 25% above above the saturation voltage (1.2VDC) of the valve control circuit (ie. 1.5VDC)	c. Program the microcontroller to send a low signal. Check voltage is 25% above the threshold voltage of transistor (1.5V) relative to ground using DMM in parallel with microcontroller.	Y
4. Valve control	a. Turn on output given an input 25% above transistor saturation voltage (1.2VDC)	a. Connect a variable power supply to the input of the circuit. Sweep the voltage from 25% below Vsaturation (0.9V) to 25% above Vsaturation (1.5V). Relay should turn on and indicator should light.	Y
	b. Turn off output given an input 25% below transistor turn-on voltage (1VDC)	b. Connect a variable power supply to the input of the circuit. Sweep the voltage from 25% above Vturn-on (1.25V) to 25% below Vturn-on (0.75V). Relay and indicator should turn off.	Y

Table C.2: System Requirements and Verifications cont.

Block	Requirement	Verification	Verification Status (Y or N)
5. Power supply	a. Provide 5 ± 0.25 VDC at 2 ± 0.25 A	a. Place DMM in parallel with power supply output. Measure voltage difference should be between 4.75 and 5.25 V. Next, add a constant-current load to the power supply output and set it to 2 ± 0.1 A. Let run for 1 hour. At the end of the hour, if the DMM still is within the voltage requirements, power supply passes. Do not use heatsinks or airflow beyond normal application.	Y
	b. If input voltage is connected with reversed polarity, output 0 V +0.1 -0.	b. Disconnect power supply output from rest of circuits. Connect battery in reverse polarity. Connect DMM to power supply output and measure voltage should be within 0 V and 0.1 V.	Y
6. Battery	a. Check battery can last 1 day at event. Expected use is 2.4A for more than 6 hours.	a. Charge battery to full capacity. Discharge battery with constant current load of 2.4 ± 0.1 A and time how long it takes to discharge to 11 ± 0.5 V. Must take more than 6 hours. Recharge battery after full discharge to prevent permanent damage.	Y
7. Pressure sensors	a. Check pressure reading shows atmospheric pressure ± 20% when fluid is at atmospheric pressure	a. Discharge both accumulators. Disconnect hosing to each accumulator at manifold end. Let fluid equalize with room temperature. Read pressure sensor output with microcontroller and verify that it reads atmospheric pressure ± 20%.	Y
	b. Check pressure readings increase as each accumulator is charged	b. Set assembled and otherwise functional bike to charge desired accumulator. Pedal bike to charge accumulator and read pressure sensor output with microcontroller to verify pressure is increasing. Test other accumulator.	Y
	c. Check pressure readings decrease as each accumulator is charged	c. Set assembled and otherwise functional bike to discharge desired accumulator. Read pressure sensor output with microcontroller to verify pressure is decreasing as bike moves forward. Test other accumulator.	Y

Appendix D Proportional Valve Research

The mechanical system supports proportional poppet valves, an type of electro-hydraulic valve that can vary flow rate depending on the current flowing through the coil. Research on how to use proportional valve controllers to control these valves was done, but it was decided to use these valves as normal poppet valves for several reasons. Our team was close to shipping the tricycle when the proportional valve control was realized. Not only does a last minute design change have the potential to introduce problems, but our team did not have time to test the proportional valve control in the real world – only bench testing had been completed.

Proportional poppet valves can be used as normal poppet valves if they are directly connected to their rated coil voltage with no current control. In this project, they were switched by a relay.

Below and on the pages that follow is the research we finished on proportional valves and their control.

Hydraulic Hardware

The proportional valve controllers we used are model EVDR-0201 from HydraForce Inc. These prop valve controllers are configured with HydraForce's ECU-Impulse software. The prop valves themselves are model SP12-20 which are a Proportional Poppet, 2-Way, Normally Closed Valve.

It is important to note that each prop valve controller can actuate two prop valves, but that it only has one input, so the two outputs cannot be independent. In our case with two accumulators, two prop valve controllers would have been desirable with separate linear potentiometers.

Electronic Hardware

The valve configuration software allows you to configure two lines with one intersecting end, mapping a 0-10 V input to a current output for the valves. The voltage available was 12 V from the battery. In order to have a max voltage of 10 V going in to the valve controllers, a resistor divider was used with a regular resistor and a linear potentiometer. While necessary, this creates a non-linear relationship between the position of the linear potentiometer and the voltage output.

In Figure D.1 is the circuit for the resistor divider. The relationship between the voltage output and the resistance of the linear potentiometer can be seen in Equation Equation D.1. The voltage output versus resistance input is shown in Figure D.2.

$$V_{out} = \frac{R_{pot}}{R_{pot} + R_{res}} * \ 12\text{V} \qquad \text{(Equation D.1)}$$

Ideally, the resistance of the regular resistor (R_{res}) used would be $\geq$ 8 kΩ as this limits the output voltage to just under 10V to allow for an input voltage slightly higher than 12V. For an 8 kΩ resistor, you could have an input up to 12.36V and still have a max output of 10V.

If a battery were to be used, care must be taken; most batteries are at a higher voltage than stated on the battery when fully charged. For example, the lead-acid battery used by the 2014-2015 team (Universal Battery UB12200) was a 12V battery, but lead-acid batteries are fully-charged at about 14.5V. In addition,

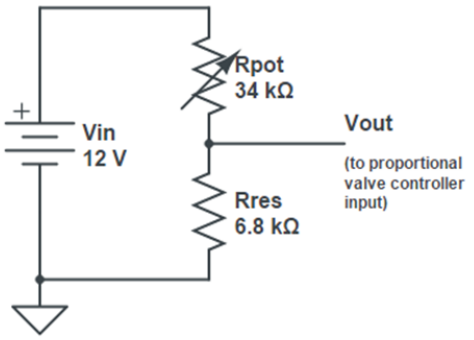


Figure D.1: Schematic for resistor divider.

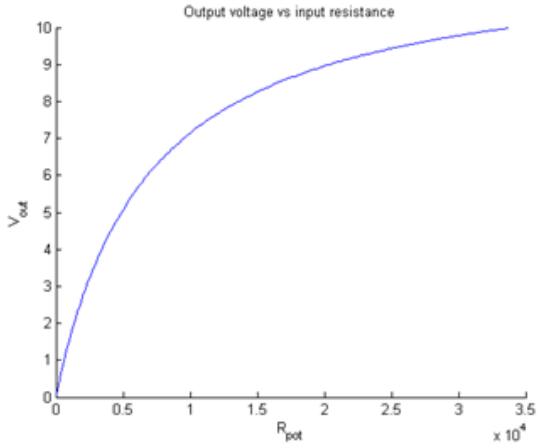


Figure D.2: Voltage output versus resistance input.

the battery voltage decreases as the battery is discharged, but the controller reference of 10V does not. It might therefore be better to change to the 0-5V input range and use a 5V regulator or similar, though this was not ideal for our setup since we had finalized the other electronics and ordered Printed Circuit Boards by this point.

Also note that resistors are rated to dissipate a certain amount of power. Power is calculated per Equation D.2. Thus, the power for either resistor (R_{pot} or R_{res} is shown in Equation D.3, where $P_{\{x\}}$ is the power through $R_{\{x\}}$.

$$P = VI \qquad \text{(Equation D.2)}$$

$$P_{\{x\}} = \frac{R_{\{x\}} V_{in}^2}{(R_{pot} + R_{res})^2} \qquad \text{(Equation D.3)}$$

Calculating fit for entry into ECU-Impulse

We found an approximate fit with two intersecting lines using MATLAB. For our bench setup, with a power supply that output almost exactly 12 V, we used the following resistances based on the hardware available to us:

$R_{pot} = 34\ \text{k}\Omega$

$R_{res} = 6.8\ \text{k}\Omega$

Code to approximate the inputs for the valve controllers

```
close all; clear all; clc;

Rpot = 0:200:34000;
Rres = 6800;
Vout = Rpot./(Rpot+Rres)*12;

trans = 25; % Index of transistion between two fit lines
valve_controller_break = 6; % Volts

figure; hold on;
plot(Rpot,Vout)

% Segment 1
p1 = polyfit(Rpot(1:trans),Vout(1:trans),1);
fit1 = polyval(p1,Rpot(1:trans));
plot(Rpot(1:trans),fit1,'r--')

% Segment 2
p2 = polyfit(Rpot(trans:end),Vout(trans:end),1);
fit2 = polyval(p2,Rpot(trans:end));
plot(Rpot(trans:end),fit2,'r-.')

% Approximation we can program into valve controller
line([0,Rpot(trans)],[0,valve_controller_break],'Color','g')
line([Rpot(trans),Rpot(end)],[valve_controller_break,10],'Color','g')

title(['Fit for transistion at ', num2str(trans),'th data point'])
xlabel('R_{pot}')
ylabel('V_{out}')
legend('V_{out}','Fit 1','Fit 2','Valve controller programming','Location','Best')
```

The transistion point trans selects the input resistance index where the two linear fit lines start and end. The variable valve_controller_break selects the output voltage where the break is on the valve controller programming line. The input resistance the break is at is selected by trans. Figure D.3 is the result of the MATLAB code.

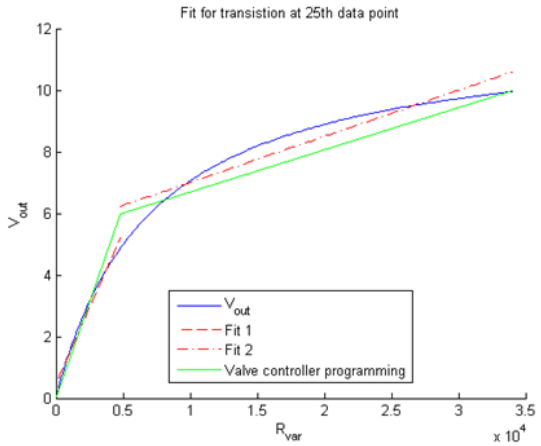


Figure D.3: Fit calculated with MATLAB code.

Controller Software - Configuration

Based on the endpoints of the chosen fit lines, we will program the valves to match the green line. The settings shown in Figure D.4 were used in the ECU-Impulse software. The Error Signal settings determine what inputs are considered an error. In an error, the valve controller will output nothing. See the manual for more details.

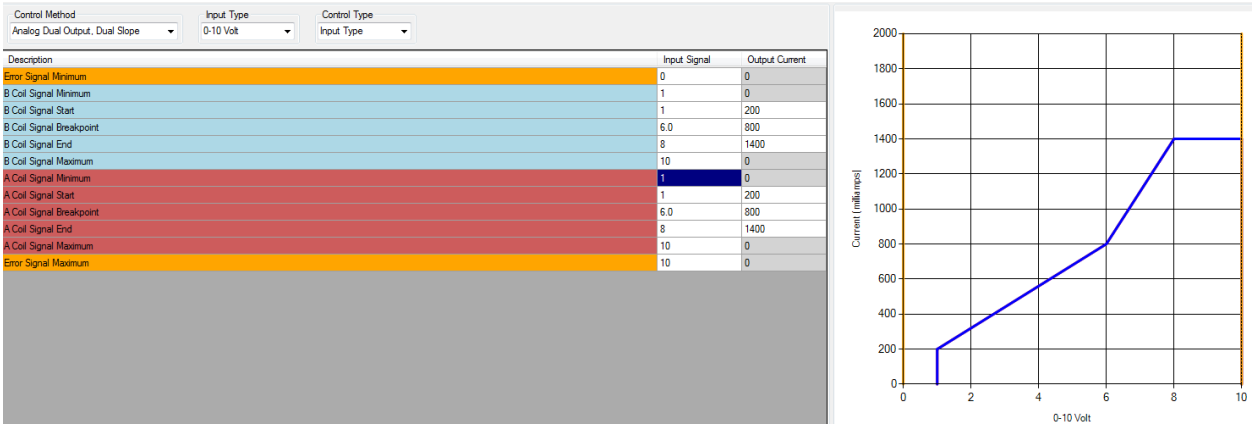


Figure D.4: Proportional valve controller settings.

Empirical Results

While we decided not to forge ahead and add proportional valve control to our tricycle, we did use the Monitor function of the ECU-Impulse software to verify our linearization. Figure D.5 shows the valve output current with respect to time. The linear potentiometer was moved at an approximately constant rate from 0 kΩ to 34 kΩ, with both outputs connected to proportional valves (coils A and B).

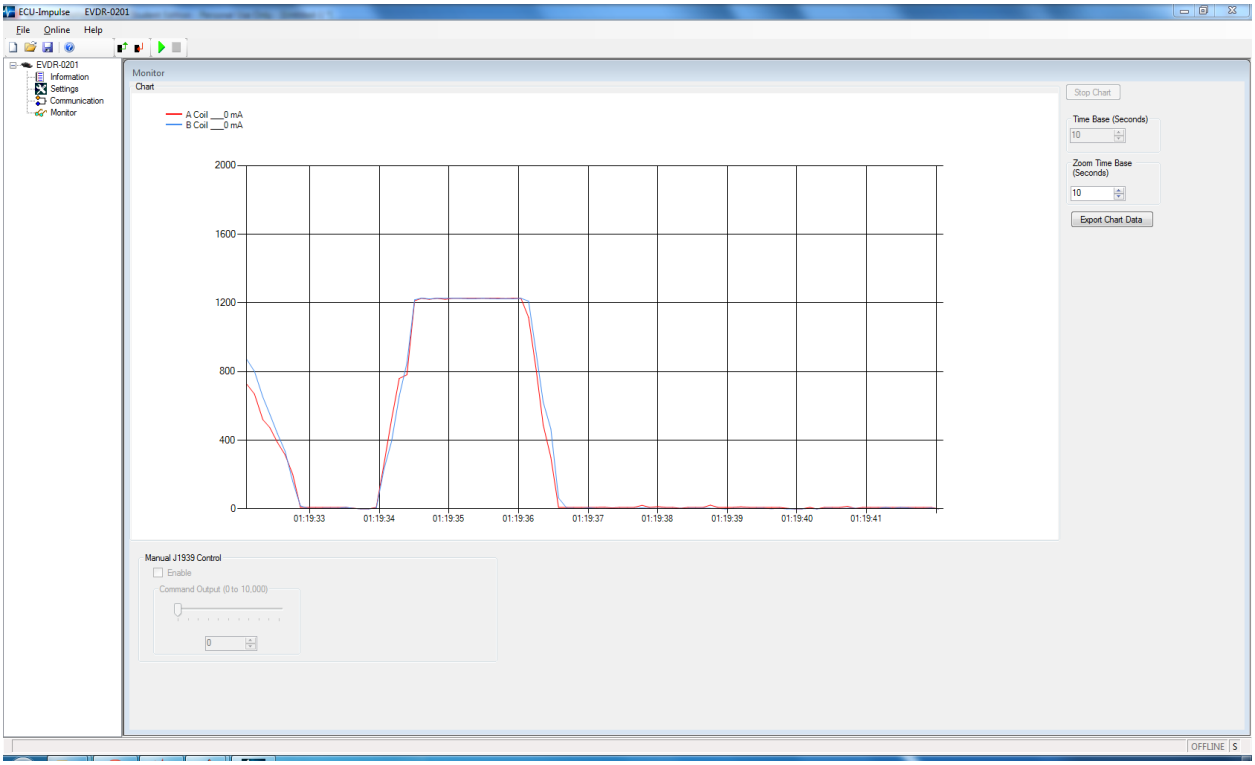


Figure D.5: Proportional valve controller monitor showing approximate linearity.

Appendix E Microcontroller Code

The code for the control system's microcontroller appears below. It has room for improvement in the following areas:

(i) The UI outputs are currently updated every time the main loop executes, even if no outputs on the user interface needs to change.

(ii) The pressure sensor measurements could be averaged. There was low enough noise that this wasn't a large issue, but this would be a good idea if making a commercial product.

```
// IO expander code based on Adafruit MCP23017 button example

#include <stdarg.h> // For LCD sprintf wrapper
#include <Wire.h> // For Adafruit_MCP23017 and LiquidCrystal
#include "Adafruit_MCP23017.h"
#include "LiquidCrystal.h" // Modified version of library for LCD backpack, available from Adafruit on Github
                           // https://github.com/adafruit/LiquidCrystal
Adafruit_MCP23017 mcp;
LiquidCrystal lcd(1); // Address 0x21 [0x20 + (~A2 ~A1 A0)]

// MCP definitions
#define MCP_BTN_CHG     1
#define MCP_BTN_DISCHG  2
#define MCP_BTN_REGEN   3
#define MCP_BTN_SEL1    5 // Large accumulator
#define MCP_BTN_SEL2    4 // Small accumulator

#define NUM_MCP_BTNS 5
uint8_t mcp_btns[NUM_MCP_BTNS] = {MCP_BTN_CHG, MCP_BTN_DISCHG, MCP_BTN_REGEN, MCP_BTN_SEL1, MCP_BTN_SEL2};

#define MCP_LED_CHG     9
#define MCP_LED_DISCHG  10
#define MCP_LED_REGEN   11
#define MCP_LED_SEL1    12 // Large accumulator
#define MCP_LED_SEL2    13 // Small accumulator

#define NUM_MCP_LEDS 5
uint8_t mcp_leds[NUM_MCP_LEDS] = {MCP_LED_CHG, MCP_LED_DISCHG, MCP_LED_REGEN, MCP_LED_SEL1, MCP_LED_SEL2};


// Status LED
#define LED_STATUS 13 // Arduino pin number
uint8_t ledBlinkState = LOW; // Initial state is off
uint8_t ledBlinkInterval = 1000; // Time between state changes in millis.
unsigned long ledBlinkPrevMillis = 0; // Millis starts at 0 when uC is turned on.

// Relay definitions
#define REL_SAFETY 8 // Arduino pin number
#define NUM_OUTPUT_RELAYS 8 // Output relays - does not include safety relay
#define REL_1       9 // Arduino pin numbers. With "A" in front, is analog pin
#define REL_2       10
#define REL_3       11
#define REL_4       12
#define REL_5       A2
#define REL_6       A3
#define REL_7       4
#define REL_8       5
uint8_t relays[NUM_OUTPUT_RELAYS] = {REL_1, REL_2, REL_3, REL_4, REL_5, REL_6, REL_7, REL_8}; // Does not include
    safety relay

// Valve definitions
#define VAL_SV1  REL_1
#define VAL_SV2  REL_2
#define VAL_SV3  REL_3
#define VAL_SV4  REL_4
#define VAL_SP1  REL_5
#define VAL_SP2  REL_6
int valves[6] = {VAL_SV1, VAL_SV2, VAL_SV3, VAL_SV4, VAL_SP1, VAL_SP2};

// Button state and mode typedefs
typedef struct {
  uint8_t chg:1;
  uint8_t dischg:1;
  uint8_t regen:1;
  uint8_t sel1:1; // Large accumulator
  uint8_t sel2:1; // Small accumulator
```

```
} buttonState_t;
buttonState_t b;

// Button debouncing variables
uint8_t btnStates[NUM_MCP_BTNS]={0,0,0,0,0,0}; // array of raw btn states
uint8_t btnPrevStates[NUM_MCP_BTNS]={0,0,0,0,0,0,}; // array of previous raw btn states
uint8_t btnDebouncedStates[NUM_MCP_BTNS]={0,0,0,0,0,0}; // array to store the state after debouncing
uint8_t btnPrevDebounceStates[NUM_MCP_BTNS]={0,0,0,0,0,0}; // array to store the previous state after debouncing
long prevTimes[NUM_MCP_BTNS]={0,0,0,0,0,0}; // array of previous times the btn has been checked
int debounceDelay=100; // time to debounce

typedef enum {
  MODE_DIRECTDRIVE,
  MODE_REGEN,
  MODE_CHARGE,
  MODE_DISCHARGE
} mode_t;
mode_t mode;

// Pressure sensors
#define PS1 0 // Arduino analog pin for use with analogRead
#define PS2 1
#define PS_MIN_PSI 0
#define PS_MAX_PSI 6000
#define PS_MIN_V 0.5
#define PS_MAX_V 4.5
int   pressures_ADC[2] = {0,0};
float pressures_V[2]   = {0,0};
float pressures_PSI[2] = {0,0};

// End of global variable definitions
// Start of printf wrapper for LCD

// Based on Arduino example code at http://playground.arduino.cc/Main/Printf
// Assumes LCD is using LiquidCrystal library and var is called 'lcd'
// Uses same format as C++ printf()
// NOTE: Does not do floats as Arduino stdio.h specifically excludes float support
void lcd_sprintf(char *fmt, ... ){
        char buf[20]; // resulting string limited to 20 chars
        va_list args;
        va_start (args, fmt );
        vsnprintf(buf, sizeof(buf), fmt, args);
        va_end (args);
        lcd.print(buf);
}

void lcd_sprintf(const __FlashStringHelper *fmt, ... ){
  char buf[20]; // resulting string limited to 20 chars
  va_list args;
  va_start (args, fmt);
#ifdef __AVR__
  vsnprintf_P(buf, sizeof(buf), (const char *)fmt, args); // progmem for AVR
#else
  vsnprintf(buf, sizeof(buf), (const char *)fmt, args); // for the rest of the world
#endif
  va_end(args);
  lcd.print(buf);
}


// End of printf wrapper for LCD
// Start of helper functions

// Stop boot setup if a button is pressed. Let the user know which button.
void startupButtonPressedError() {
  char msg[] = "Button press at boot";

  mode = MODE_DIRECTDRIVE;
  updateValves();

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("BOOT ERROR:");
  lcd.setCursor(0,1);
  lcd.print(msg);
  lcd.setCursor(0,2);
  lcd.print("["); // Row 3
  if(b.chg)     lcd.print("Chg ");
  if(b.dischg)  lcd.print("Dischg ");
  if(b.regen)   lcd.print("Regen ");
  lcd.print("]");
  lcd.setCursor(0,3);
  lcd.print("Reset to continue");

  while(1); // Wait for user to cycle power
}
```

```
void readButtons_debouncePins(){
  volatile long _millis=millis();

  for(uint8_t btn=0;btn<NUM_MCP_BTNS;btn++){
    if(btnStates[btn]!=btnPrevStates[btn]){
      prevTimes[btn]=_millis;
    }
    if(_millis-prevTimes[btn]>debounceDelay){
      prevTimes[btn]=_millis;
      btnDebouncedStates[btn]=btnStates[btn];
    }
    btnPrevStates[btn]=btnStates[btn];
  }

}

void readButtons_checkStateChangeAndToggle(){
  // Check if b.chg needs to toggle
  if(btnPrevDebounceStates[0] != btnDebouncedStates[0]) {
    if(btnDebouncedStates[0] == HIGH) {
      b.chg = b.chg ^ 1;
    }
  }

  // Set previous
  for(short btn=0;btn<5;btn++){
    btnPrevDebounceStates[btn]=btnDebouncedStates[btn];
  }
}

// Based on Ubi De Feo s debounce code from http://hackaday.com/2010/11/09/debounce-code-one-post-to-rule-them-
    all/
void readButtons() { // With debouncing
  // Get raw inputs
  for(uint8_t btn=0;btn<NUM_MCP_BTNS;btn++){
    volatile int pin=mcp_btns[btn];
    volatile int mpPin=pin;
    volatile int pinPosition=btn;

    btnStates[pinPosition]=mcp.digitalRead(pin) ? 0 : 1;
  }

  readButtons_debouncePins();

  // Use debounce function on each button
  //b.chg    = btnDebouncedStates[0];
  b.dischg = btnDebouncedStates[1];
  b.regen  = btnDebouncedStates[2];
  b.sel1   = btnDebouncedStates[3];
  b.sel2   = btnDebouncedStates[4];

  // Make the Charge button a toggle
  readButtons_checkStateChangeAndToggle();
}

// No debouncing
void readButtonsRaw() {
  b.chg    = mcp.digitalRead(MCP_BTN_CHG)    ? 0 : 1;
  b.dischg = mcp.digitalRead(MCP_BTN_DISCHG) ? 0 : 1;
  b.regen  = mcp.digitalRead(MCP_BTN_REGEN)  ? 0 : 1;
  b.sel1   = mcp.digitalRead(MCP_BTN_SEL1)   ? 0 : 1; // Large accumulator
  b.sel2   = mcp.digitalRead(MCP_BTN_SEL2)   ? 0 : 1; // Small accumulator
}

void waitNoButtons() {
  lcd.setCursor(0,2);
  lcd.print("RELEASE ALL BUTTONS,");
  lcd.setCursor(0,3);
  lcd.print("INCL. TOGGLE BUTTONS");
  uint8_t but = 1;
  while(but) {
    readButtons(); // With debouncing
    but = b.regen || b.chg || b.dischg;
  }
  lcd.setCursor(0,2);
  lcd.print("No buttons pressed! ");
  delay(500);
  lcd.clear();
}

// Based on BlinkWithoutDelay tutorial at http://arduino.cc/en/Tutorial/BlinkWithoutDelay
void blinkStatusLED() {
  unsigned long ledBlinkCurrMillis = millis();
  if(ledBlinkCurrMillis - ledBlinkPrevMillis > ledBlinkInterval) {
    ledBlinkPrevMillis = ledBlinkCurrMillis;
```

```
    ledBlinkState = (ledBlinkState == LOW) ? HIGH : LOW;
    digitalWrite(LED_STATUS, ledBlinkState);
  }
}

// Changes state of all LEDs. Accepts HIGH or LOW. Invalid inputs are interpreted as HIGH.
void setAllLEDs(uint8_t state) {
  state = (state == LOW) ? LOW : HIGH; // If value is not HIGH OR LOW, make it HIGH.
  digitalWrite(LED_STATUS,state);
  setMCPLEDs(state);
}

// Changes state of all LEDs connected to the MCP. Accepts HIGH or LOW. Invalid inputs are interpreted as HIGH.
void setMCPLEDs(uint8_t state) {
  state = (state == LOW) ? LOW : HIGH; // If value is not HIGH OR LOW, make it HIGH.
  uint8_t i;
  for(i=0;i<NUM_MCP_LEDS;i++) {
    mcp.digitalWrite(mcp_leds[i],state);
  }
}

// Sets LEDs based on mode
void setLEDsForMode() {
  // Turn off all LEDs
  setMCPLEDs(LOW);
  // Turn on LED for current mode.
  switch(mode) {
     case MODE_CHARGE:
      mcp.digitalWrite(MCP_LED_CHG,HIGH);
      break;
     case MODE_DISCHARGE:
      mcp.digitalWrite(MCP_LED_DISCHG,HIGH);
      break;
    case MODE_REGEN:
      mcp.digitalWrite(MCP_LED_REGEN,HIGH);
      break;
    default:
      break;
  }
  // Set accumulator selector LEDs
  if(b.sel1) mcp.digitalWrite(MCP_LED_SEL1,HIGH);
  else if(b.sel2) mcp.digitalWrite(MCP_LED_SEL2,HIGH);
  else {
    mcp.digitalWrite(MCP_LED_SEL1,HIGH);
    mcp.digitalWrite(MCP_LED_SEL2,HIGH);
  }
}

void SRon() {
  digitalWrite(REL_SAFETY,HIGH);
}
void SRoff() {
  digitalWrite(REL_SAFETY,LOW);
}

// Based on assumption that if you are turning on valves, you were previously in Direct Drive and
//  that switching on the valves doesn't create any edge cases.
//  ### TODO - ABOVE STATEMENT WILL NEED LATER RE-EVALUATION ###
// valves: VAL_SV1, VAL_SV2, VAL_SV3, VAL_SV4, VAL_SP1, VAL_SP2
void updateValves() {
  switch(mode) {
    case MODE_CHARGE:
      lcd.setCursor(0,3); // Start of bottom line
      lcd.print("Charging ");
      digitalWrite(VAL_SV3,HIGH);
      digitalWrite(VAL_SV4,LOW );
      digitalWrite(VAL_SP1,LOW );
      digitalWrite(VAL_SP2,LOW );
      if(b.sel1) { // Large accumulator only
        digitalWrite(VAL_SV1,HIGH);
        digitalWrite(VAL_SV2,LOW );
        lcd.print("    ACC   L");
      } else if (b.sel2) { // Small accumulator only
        digitalWrite(VAL_SV1,LOW );
        digitalWrite(VAL_SV2,HIGH);
        lcd.print("    ACC S  ");
      } else { // Both accumulators
        digitalWrite(VAL_SV1,HIGH);
        digitalWrite(VAL_SV2,HIGH);
        lcd.print("    ACC S+L");
      }
      break;
    case MODE_DISCHARGE:
      lcd.setCursor(0,3); // Start of bottom line
      lcd.print("Discharging ");
      digitalWrite(VAL_SV1,LOW );
```

```
      digitalWrite(VAL_SV2,LOW );
      digitalWrite(VAL_SV3,LOW );
      digitalWrite(VAL_SV4,LOW );
      if(b.sel1) { // Large accumulator only
        digitalWrite(VAL_SP1,HIGH);
        digitalWrite(VAL_SP2,LOW );
        lcd.print(" ACC   L");
      } else if (b.sel2) { // Small accumulator only
        digitalWrite(VAL_SP1,LOW );
        digitalWrite(VAL_SP2,HIGH);
        lcd.print(" ACC S  ");
      }
      else { // Both accumulators
        digitalWrite(VAL_SP1,HIGH);
        digitalWrite(VAL_SP2,HIGH);
        lcd.print(" ACC S+L");
      }
      break;
    case MODE_REGEN:
      lcd.setCursor(0,3); // Start of bottom line
      lcd.print("Regen.      ");
      digitalWrite(VAL_SV3,HIGH);
      digitalWrite(VAL_SV4,HIGH);
      digitalWrite(VAL_SP1,LOW );
      digitalWrite(VAL_SP2,LOW );
      if(b.sel1) { // Large accumulator only
        digitalWrite(VAL_SV1,HIGH);
        digitalWrite(VAL_SV2,LOW );
        lcd.print(" ACC   L");
      } else if (b.sel2) { // Small accumulator only
        digitalWrite(VAL_SV1,LOW );
        digitalWrite(VAL_SV2,HIGH);
        lcd.print(" ACC S  ");
      } else { // Both accumulators
        digitalWrite(VAL_SV1,HIGH);
        digitalWrite(VAL_SV2,HIGH);
        lcd.print(" ACC S+L");
      }
      break;
    case MODE_DIRECTDRIVE:
    default:
      // Nothing powered
      digitalWrite(VAL_SV1,LOW );
      digitalWrite(VAL_SV2,LOW );
      digitalWrite(VAL_SV3,LOW );
      digitalWrite(VAL_SV4,LOW );
      digitalWrite(VAL_SP1,LOW );
      digitalWrite(VAL_SP2,LOW );
      lcd.setCursor(0,3); // Start of bottom line
      lcd.print("Direct Drive       ");
      break;
  }
}

  // Read analog inputs
void updatePressures() {
  // ### TODO - Take several samples to reduce effects of noise ###

  pressures_ADC[0] = analogRead(PS1); // Large accumulator
  pressures_ADC[1] = analogRead(PS2); // Small accumulator

  pressures_V[0] = ((float)pressures_ADC[0])/1023*5; // Large accumulator
  pressures_V[1] = ((float)pressures_ADC[1])/1023*5; // Small accumulator

  pressures_PSI[0] = (pressures_V[0]-(PS_MIN_V))*((PS_MAX_PSI)-(PS_MIN_PSI))/((PS_MAX_V)-(PS_MIN_V));
  pressures_PSI[1] = (pressures_V[1]-(PS_MIN_V))*((PS_MAX_PSI)-(PS_MIN_PSI))/((PS_MAX_V)-(PS_MIN_V));

  // Display on output, large on left, small on right
  lcd.setCursor(0,0);
  lcd.print("Large:    Small:    ");
  lcd.setCursor(0,1);
  lcd_sprintf("%.4d PSI  %.4d PSI  ",(int)pressures_PSI[0],(int)pressures_PSI[1]);
  /*
  // ### DEBUGGING ###
  lcd.setCursor(0,2);
  lcd_sprintf("PS1:%4d  PS2:%4d  ",pressures_ADC[0], pressures_ADC[1]);
  // ### END DEBUGGING ###
  */
}

// End of helper functions
// Start of main code

void setup() {
  // Status LED
  pinMode(LED_STATUS,OUTPUT);
```

```
409
    // Relays
    // Note that the valves are controlled by relays, so this is required for valve operation
    pinMode(REL_SAFETY,OUTPUT);
    uint8_t i;
414 for(i=0;i<NUM_OUTPUT_RELAYS;i++) {
      pinMode(relays[i],OUTPUT);
    }

    // Port expander
419 mcp.begin();
    // UI buttons
    mcp.pinMode(MCP_BTN_REGEN,INPUT);
    mcp.pinMode(MCP_BTN_CHG,INPUT);
    mcp.pinMode(MCP_BTN_DISCHG,INPUT);
424 mcp.pinMode(MCP_BTN_SEL1,INPUT);
    mcp.pinMode(MCP_BTN_SEL2,INPUT);
    // UI LEDs
    mcp.pinMode(MCP_LED_REGEN,OUTPUT);
    mcp.pinMode(MCP_LED_CHG,OUTPUT);
429 mcp.pinMode(MCP_LED_DISCHG,OUTPUT);
    mcp.pinMode(MCP_LED_SEL1,OUTPUT);
    mcp.pinMode(MCP_LED_SEL2,OUTPUT);

    // LCD backpack
434 lcd.begin(20, 4); // columns, rows
    lcd.print("Setting up system...");

    // If buttons pressed, instead of actuating that mode, throw a critical error.
    readButtonsRaw(); // No debouncing desired.
439 if(b.regen || b.chg || b.dischg) {
      startupButtonPressedError();
    }

    // Setup variables
444 mode = MODE_DIRECTDRIVE;
    updateValves();
    updatePressures();

    lcd.setCursor(0,0);
449 lcd.print("System initialized. "); // Pad to 20 chars so we don't have to use lcd.clear().
    lcd.setCursor(0,1); // Force line ordering, otherwise order is row 1, 3, 2, 4
    lcd.print("For accumulators,   ");
    lcd.setCursor(0,2);
    lcd.print("S = small, L = large");
454 delay(1000);
    lcd.clear();
}

void loop() {
459 blinkStatusLED();
    readButtons(); // With debouncing
    if(b.regen ? (b.chg || b.dischg) : (b.chg && b.dischg)) {
      // More than two buttons pressed
      SRoff();
464   mode = MODE_DIRECTDRIVE;
      updateValves();
      setAllLEDs(HIGH);
      waitNoButtons();
      setAllLEDs(LOW);
469 } else if (!b.regen && !b.chg && !b.dischg) {
      // No buttons pressed
      mode = MODE_DIRECTDRIVE;
      updateValves();
      SRon();
474 } else if (b.regen) {
      // Regen only
      mode = MODE_REGEN;
      updateValves();
      SRon();
479 } else if (b.chg) {
      // Charge only
      mode = MODE_CHARGE;
      updateValves();
      SRon();
484 } else if (b.dischg) {
      // Discharge only
      mode = MODE_DISCHARGE;
      updateValves();
      SRon();
489 }
    setLEDsForMode();
    updatePressures();
}

494 // End of main code
```

Appendix F Hydraulic Fluid Paths in Manifold

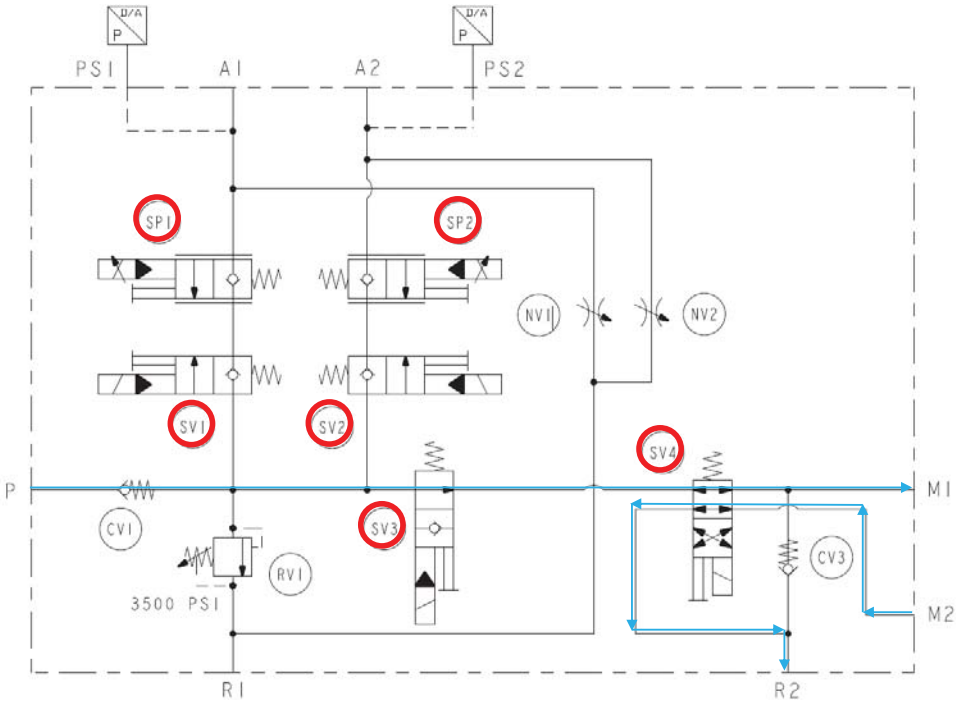
The next few pages show the various fluid paths in the manifold as well as which valves needed to be actuated for each state. Ian O'Leary, a Mechanical Engineer on our team, is the original author of the content.

Updated Manifold 2014-2015

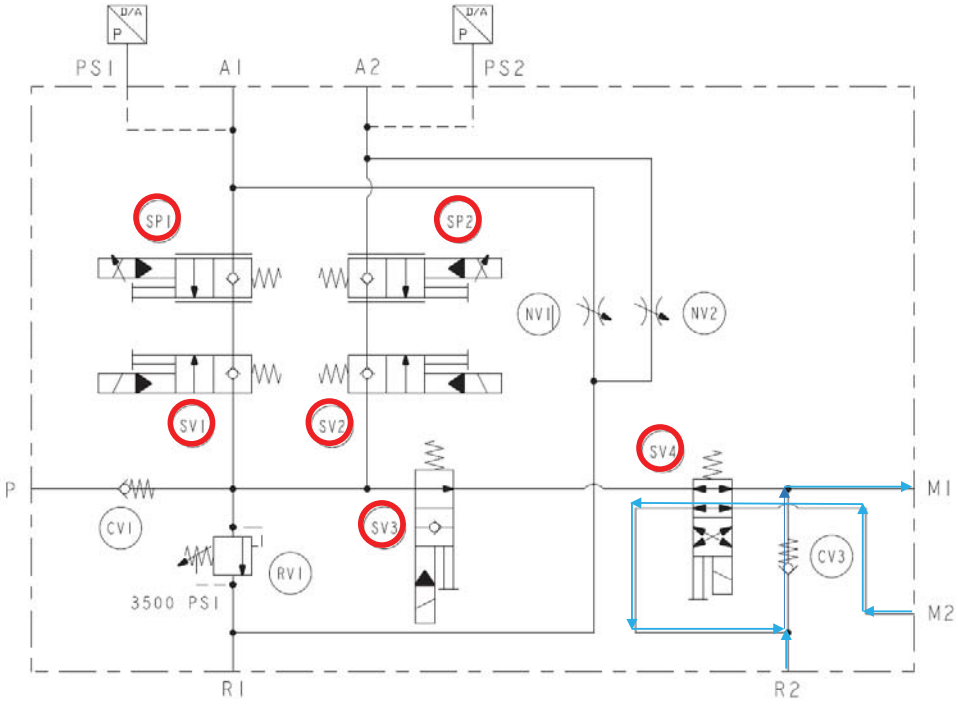
Made by Ian O'Leary for bicycle control team

=OFF =ON =FLOW

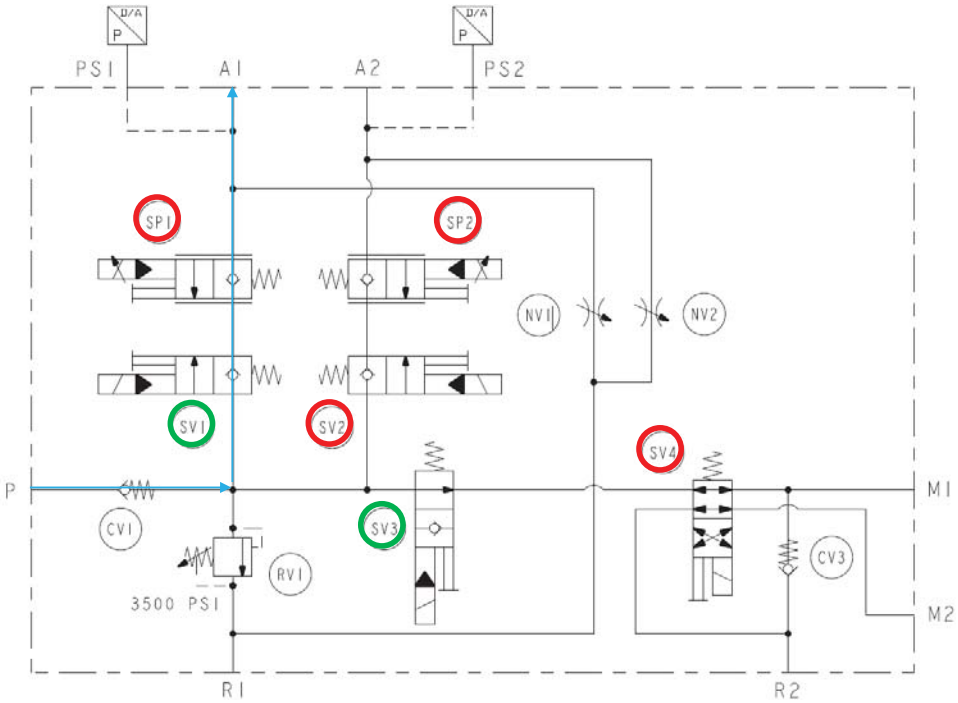
Direct Drive



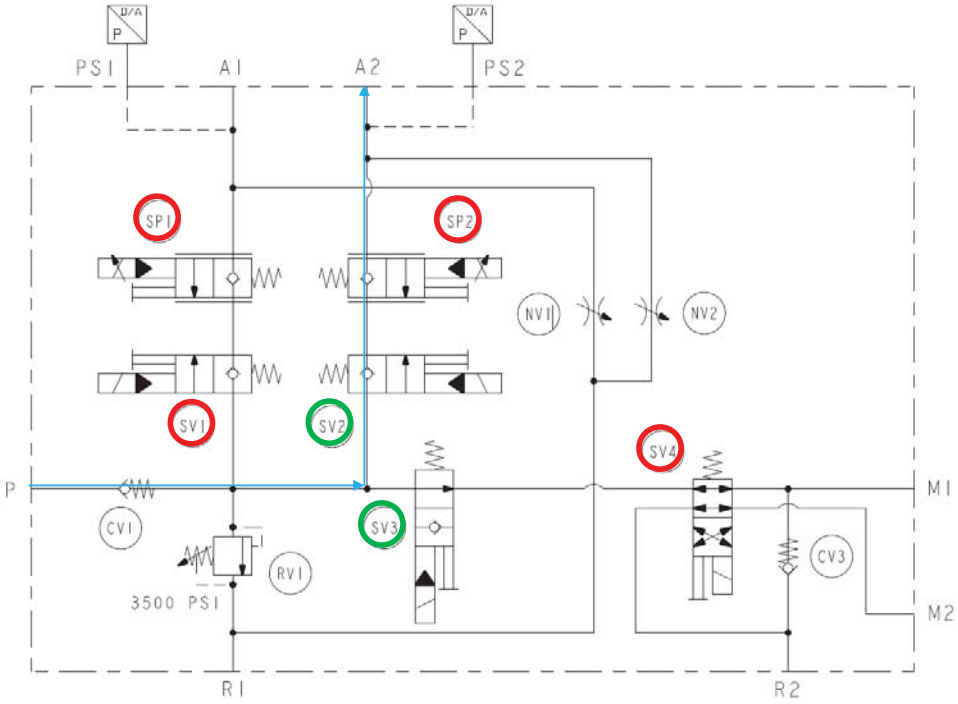
Coasting



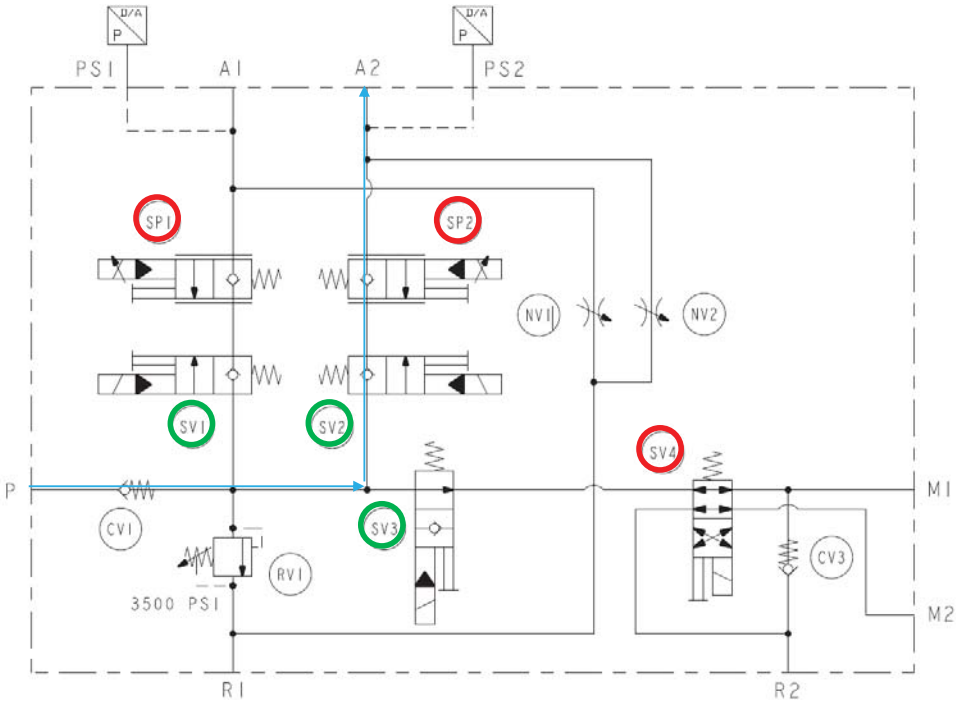
Charge A1



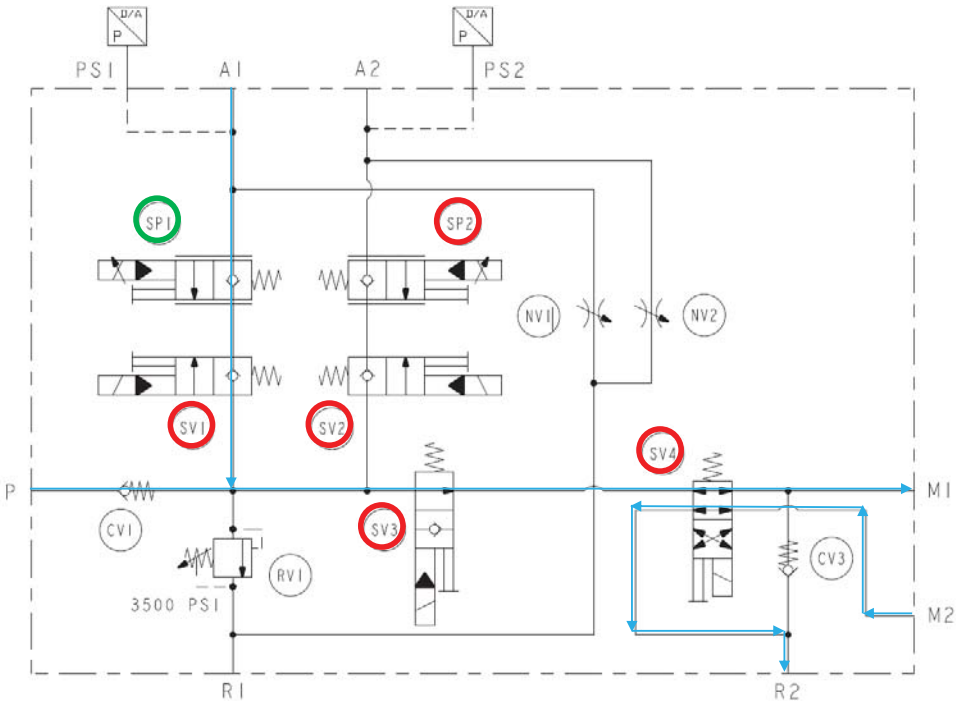
Charge A2



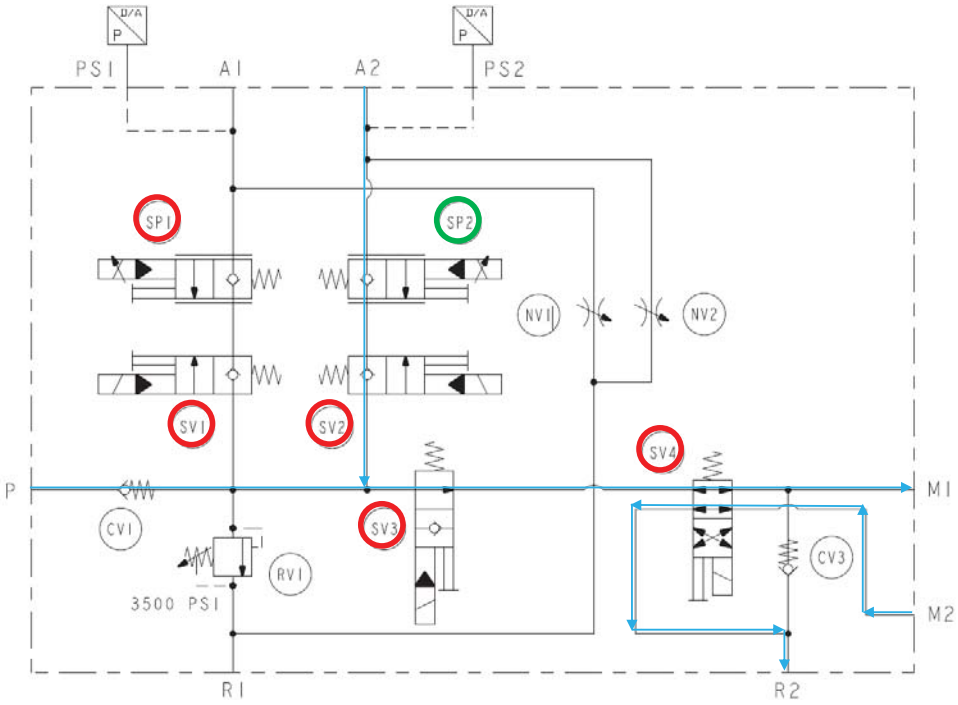
Charge A1 & A2



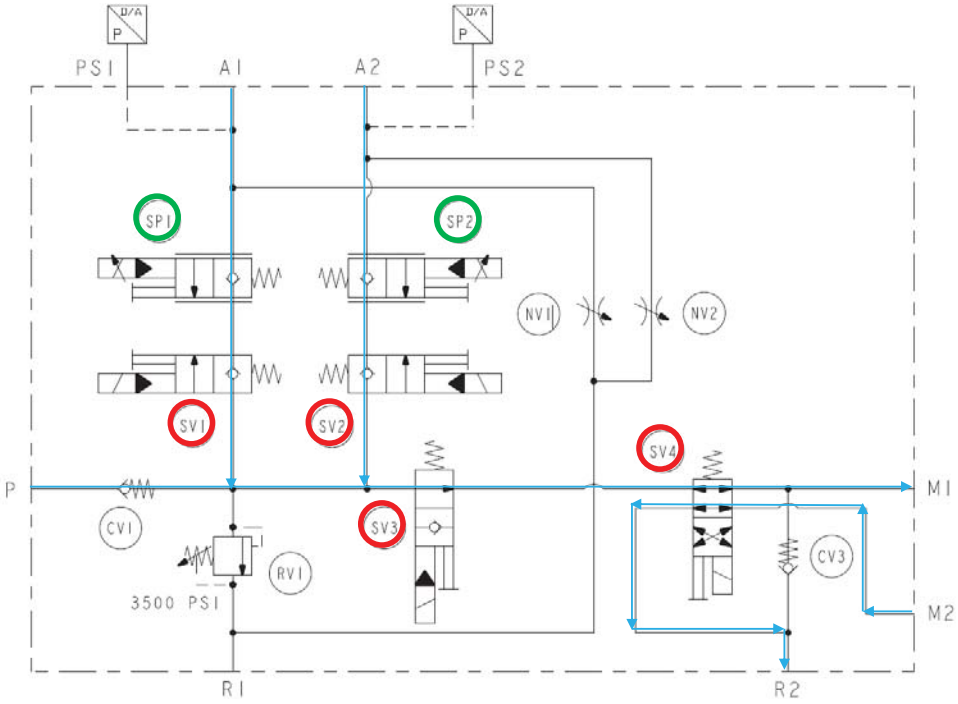
Discharge A1 (Optional Pedaling)



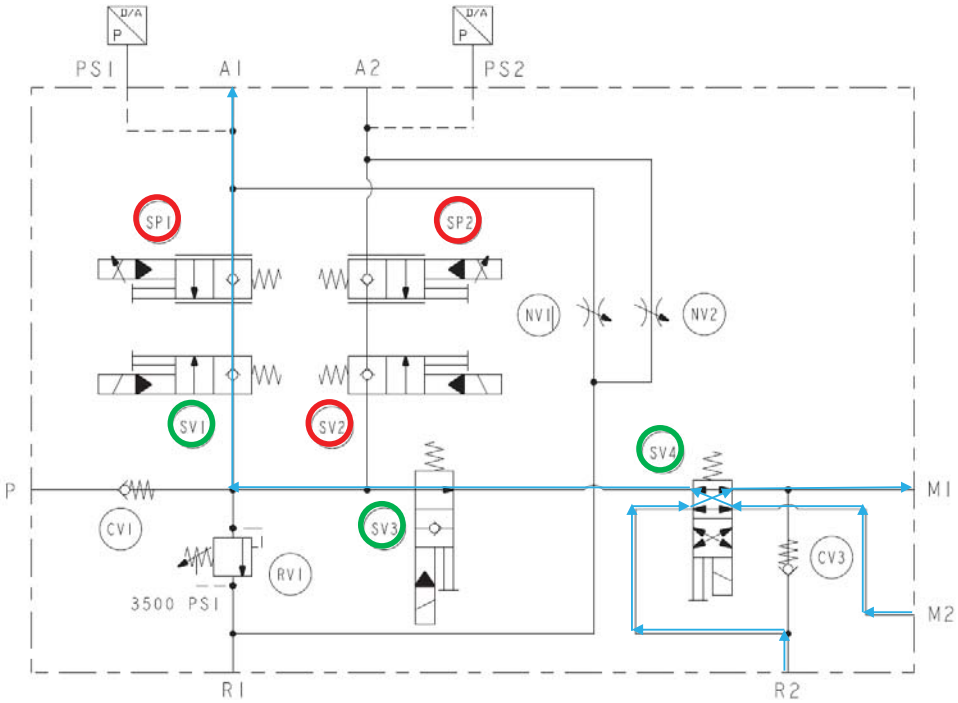
Discharge A2 (Optional Pedaling)



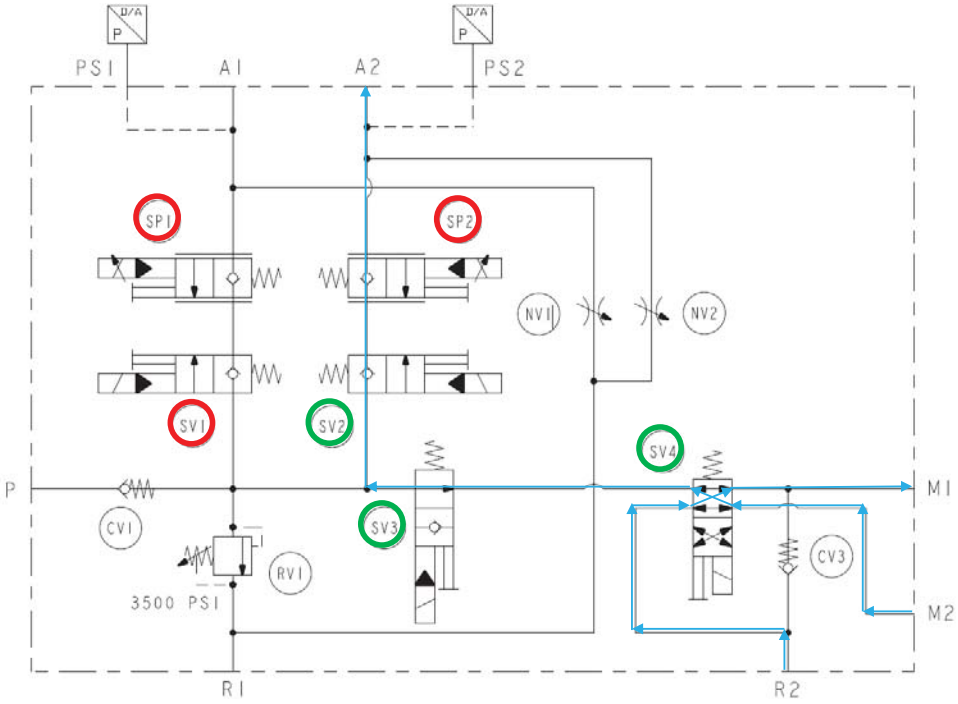
Discharge A1 & A2 (Optional Pedaling)



Regenerative Breaking A1



Regenerative Breaking A2



Regenerative Breaking A2

