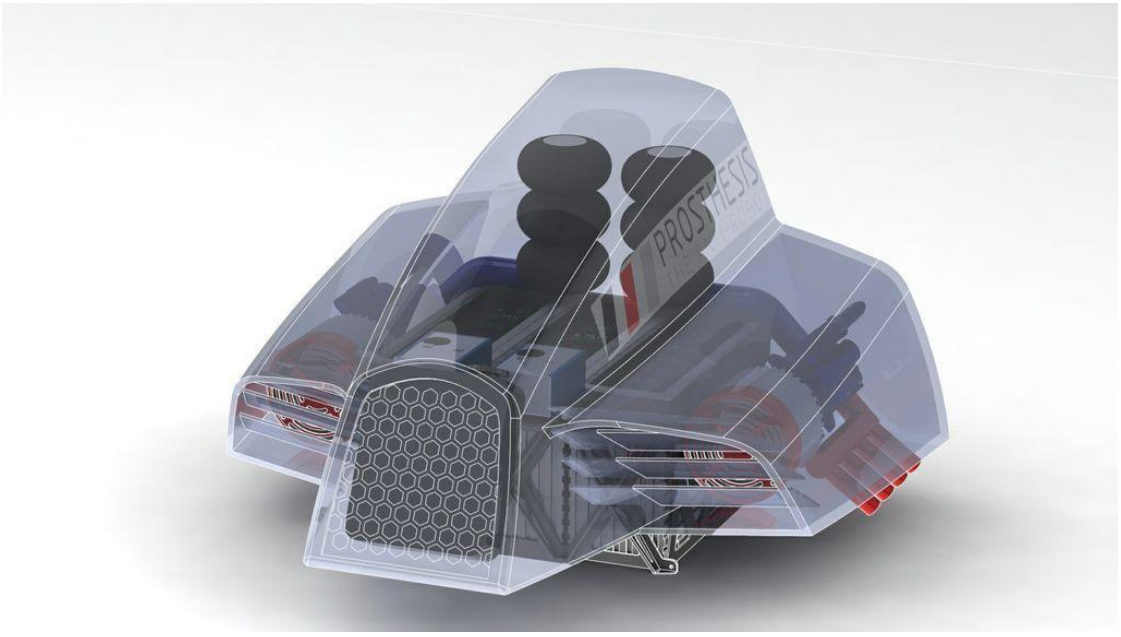
Engineering Recommendation Report - Prosthesis Electric Power Pack Control System



Edson Sanchez
Francesco Anzovino
Thomas Watanabe

Project Sponsor:
Jonathan Tippett

Project 1301
Applied Science 459
Engineering Physics Project Laboratory
The University of British Columbia
April 2, 2013

Executive Summary

Prosthesis is a two story tall, 3000 kg, four legged wearable walking machine intended to be human controlled. Its movements will be controlled by a pilot standing inside of the machine. The pilot's movements will be mapped to the legs of Prosthesis via an exoskeletal interface. The final machine will require approximately 60 kW of hydraulic power to reach top speed and will be 100% electric. The current prototype leg, also known as the Alpha leg, requires a 12 kW continuous power supply, which will come from a pump and electric motor combination.

Our team developed the control system for the electric power pack of Prosthesis. The work involved computer programming, mostly in C, as well as the processing of sensor data. In terms of the physical components of the pack, we helped our project sponsor by providing him with a set of calculations and performance curves detailing the possible states of the electric motors that will be used on Prosthesis. To perform tests on the controller that we developed, we chose a gear pump proportional to the electric power pack available to us at the time of testing. In other words, we determined the capacity of the pump we needed to be able to move the Alpha leg with a different power pack.

The control is done via an Arduino microcontroller that is programmed in C. It is in charge of keeping the pressure as close as possible to a user-specified pressure, known as the setpoint. This was done by implementing a software PID controller on the Arduino. The controller has two operating modes, one of which follows a simplified PID control scheme, using a constant pressure as the "target" or setpoint. The other controller setting, referred to as Load Sense, keeps the pressure at the pump outlet some constant amount higher than the pressure at a predetermined spot in the hydraulic line. Simply put, the pump outlet pressure adapts to the load in the hydraulic line. The setpoint based PID control was tested and found to be within acceptable operating parameters. An initial version of the Load Sense control was created, but it was neither completed nor tested by the submission of this report.

Table of Contents

List of Figures

List of Tables

Introduction

Background and significance of the project:

From September to December 2012, we planned the control system of the electric power pack of Prosthesis. Prosthesis will be a 3000 kg, 2 story tall, hydraulic walking machine. The power pack that provides power to this gigantic walking machine, consists essentially of 2 DC electric motors, each one providing power to a pump which controls the pressure and flow in the hydraulic lines of Prosthesis. From January 1 to April 1 2013, we executed a third of the originally proposed plan and then modified our goals to better accommodate with the major project timeline and concentrate on the operation of the electric power pack. Our contribution is significant to the overall project because we setup the control parameters that will run the machine. This report summarizes the work done and provides our project sponsor or anyone working on this project in the future with useful recommendations for construction of the Electric Power Pack.

Statement of the Problem/Project Objectives:

The main objective was to have a controller for the power pack of Prosthesis. Prosthesis requires varying amounts of power for the different kinds of movements a pilot controlling it can execute. The Power Pack mainly consists of 2 DC electric motors, 2 motor controllers, 1 battery pack, 2 hydraulic pumps as well as an Arduino microcontroller. As the Power Pack Control Systems team, our main objective was to set up the available components in order to provide the power required for every movement. In addition we had to develop a way to register users for the power pack. The reason for this is to allow registered users to operate Prosthesis.

For testing our controller, it was important to choose an adequate pump that could be used with the motor from an existing walking machine, the Mondo Spider. In other words, we had to choose a pump capable of moving the Alpha leg using the Mondo Spider's electric power pack. Later on, once our project sponsor had decided on an optimal DC motor and pump configuration for Prosthesis, we had to provide a spreadsheet containing the possible configurations of the DC motor. Finally, we were tasked with fabricating a control panel that could be easily used by other engineers to calibrate or change the control parameters of the power pack controller we developed.

Scope and Limitations:

At the beginning of our project, our project sponsor asked us to develop the control system using gear pumps to control the pressure in the hydraulic lines. Once we developed this system, we realized that using different kinds of pumps would be more adequate for the system we were developing. Prosthesis is a machine that will most likely require sudden spikes in power from the power pack. In terms of hydraulic power, this requirement translates into being able to provide fluid flow really quickly. An axial piston pump (also known as variable flow pump) is the best choice for this kind of application. Unfortunately, neither the variable flow pumps nor the DC motors for Prosthesis would get purchased until the end of our project timeline. For this reason we developed a controller for the gear pumps and the DC motors of the Mondo Spider. If we had the variable flow pumps, we would have developed a PID controller that uses RPM as a controlled variable instead of pressure. Though the code we wrote is easily transferable to an RPM PID controller, this report contains information about the development and testing of the pressure PID controller, and it doesn't go into detail on the RPM PID controller.

After developing a regular pressure PID controller, we added another operating mode to our controller called LoadSense mode. Due to time constraints, we were only able to test it a couple of times and we were unsuccessful in getting it working. However, we provide useful suggestions in the recommendation section of this report for troubleshooting this issue. Since this part of the project is not yet functioning, we were not able to capture test results for this operating mode of our controller.

Discussion

Theory:

DC motors: A DC motor produces mechanical power by rotating a shaft with certain torque and angular velocity. It is powered from a DC Power source. The following equations were used to determine the power output and efficiency of the EMC-RT 200, a permanent magnet DC motor, at different operating points.

$$\tau_e = k \times I \text{; where } \tau_e = \text{electrical torque}, k = \text{torque constant}, I = \text{current}$$

$$\text{Power Output Motor} = \tau_m \times \omega \text{; where } \tau_m = \text{mechanical torque}, \omega = \text{speed}$$

$$\text{Efficiency} = \frac{\text{Power Output}}{\text{Power Input}} = \frac{\tau_m \times \omega}{VI}$$

Hydraulic systems: In a hydraulic circuit, the power is given by the following equation:

$$\text{Power} = \text{Pressure} \times \text{Flow rate}$$

In a gear pump, a geared rotor pushes fluid between its teeth and an enclosure as shown in Figure 1. This method of forcing the fluid results in the pressure depending on the torque of the rotor and in the flow rate depending on the rotor's speed.

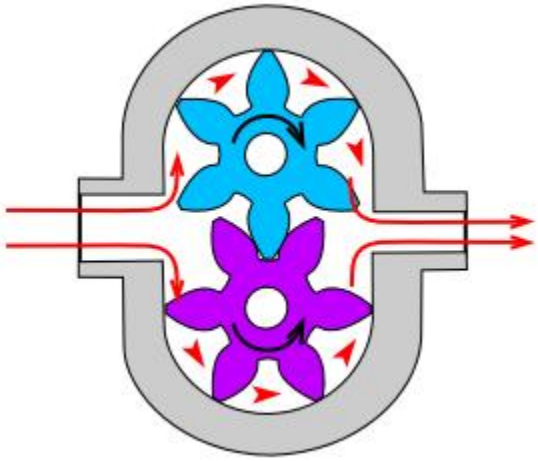


Figure 1: Typical External Gear Pump, http://upload.wikimedia.org/wikipedia/commons/c/c4/Gear_pump.png

The equations below were used to calculate the torque required and flow rate produced by the gear pump that was used on the Mondo Spider.

$$\text{Pressure} = \frac{T_p \times 2\pi \times \text{efficiency}}{\text{pump displacement}} \text{ ; where } \tau_p = \text{pump torque}$$

$$\text{Flow rate} = \text{pump displacement} \times \omega \text{ ; where } \omega = \text{angular velocity}$$

An axial piston pump behaves in a slightly different way. Figure 2 shows the side view of a typical piston pump. Multiple pistons are connected to a swash plate, which can be adjusted to make different angles to the shaft. The angle of the swash plate affects how much the pistons can move laterally. By rotating the shaft, the pistons move from right to left displacing the fluid outwards. As the piston movement in the lateral direction increases, more fluid is drawn in and then forced out, increasing the pump displacement. This variable pump displacement allows for another method of control. A hydraulic connection can be made that will adjust the swash angle automatically according to a difference in pressure.

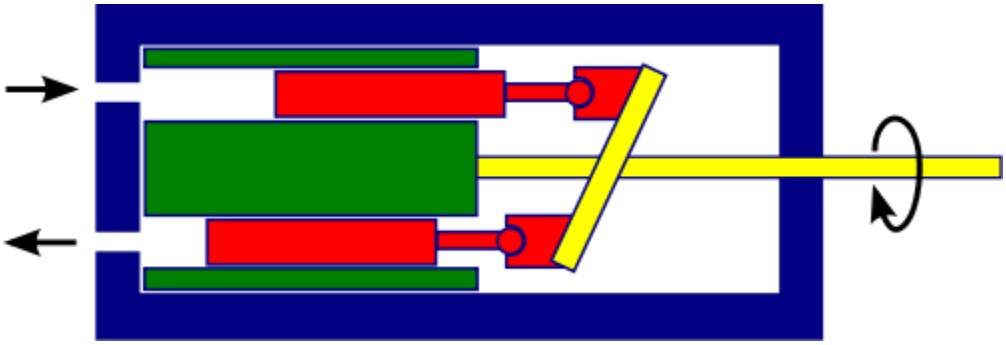


Figure 2: Axial Piston Pump, http://upload.wikimedia.org/wikipedia/en/0/0d/Axial_piston_pump.svg

PID control: A PID controller calculates an Error based on an Input value and some Setpoint that the Output should reach. A weighted sum of the present, past, and future errors is calculated to determine the Output. The weight of these terms is represented by three values, the proportional (P), the integral (I), and the derivative (D). This is represented in Figure 3.

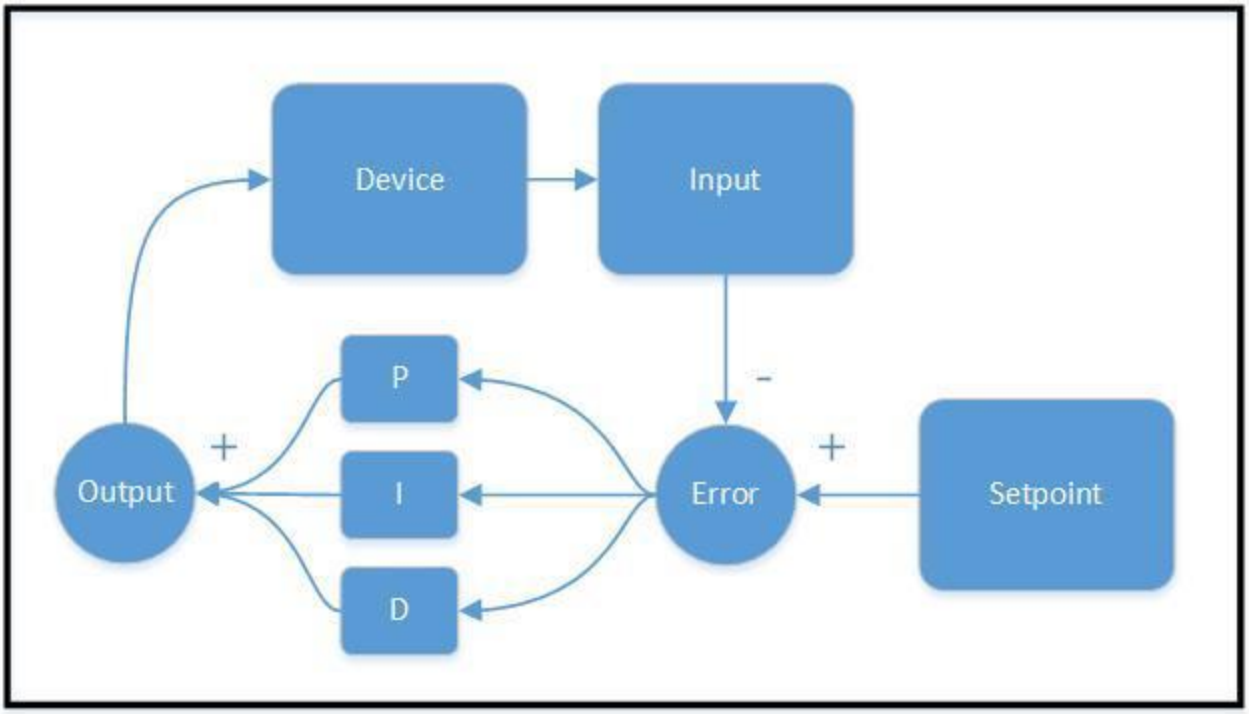


Figure 3: Diagram of PID Control

By changing the values of the P, I, and D, the behaviour of the Output can be altered. This is referred to as 'tuning' the PID. In general, the tuning for the PID control will be system dependent, as different responses might be desired for different circumstances.

Methods / Testing Protocol:

DC motors: Calculations were made to provide our project sponsor with useful information about the motors chosen for Prosthesis. The calculations aim to provide more information about the output of the motors than the information provided in the datasheets. The documentation that accompanied the EMC-RT 200's only provided data up to a torque of 40 Nm. The stall torque of the EMC-RT 200 is 108 Nm, so we extrapolated the performance curves with the assumption that the relationships remain the same upto the stall torque.

For instance, we wanted to know the speed torque relationship from zero torque and maximum speed all the way up to stall torque. Figure 4 below, shows the extrapolated speed and torque relationship for the EMC-RT 200.

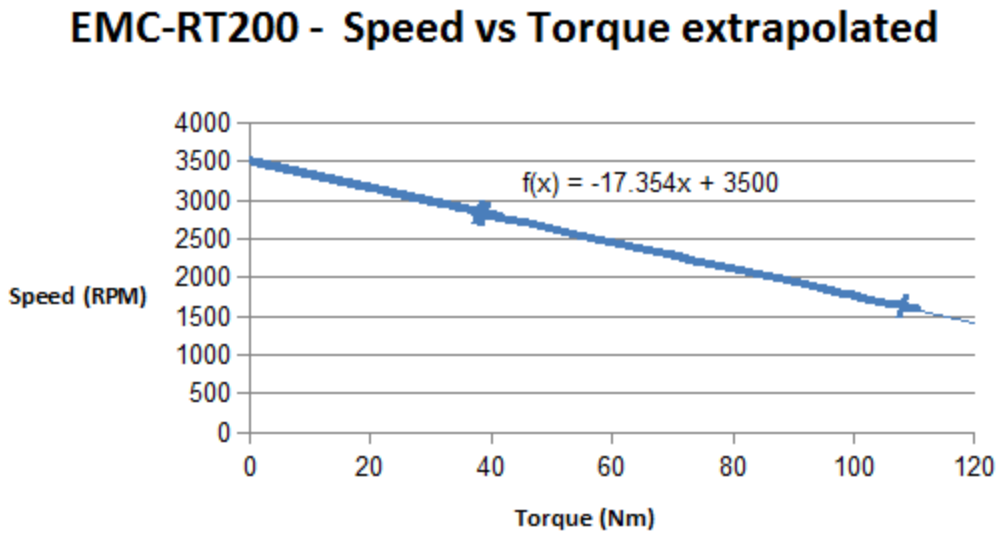


Figure 4: Speed vs. Torque Graph

In addition, it was desirable to determine the efficiency of the motor at different speeds and torques. Figure 5 shows the efficiency of this motor as a function of speed.

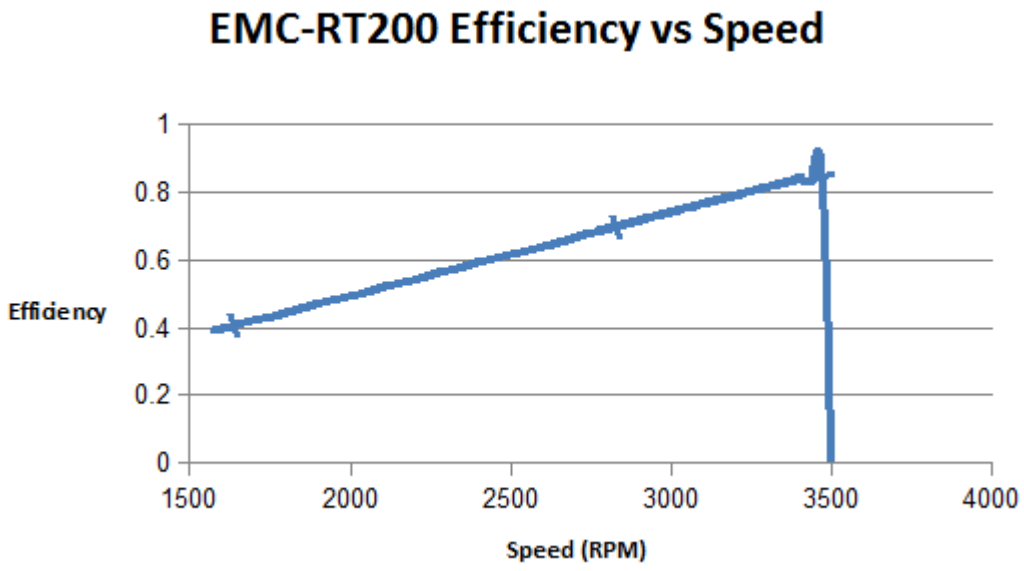


Figure 5: Efficiency vs. Speed Graph

A similar analysis was performed to understand the performance of the DC motors used on the Mondo Spider, an existing hydraulic walking machine with an electric power pack, which we used for testing. Please refer to Appendix B for the complete

spreadsheet containing all data and performance curves for the Mondo Spider's DC motor.

Hydraulic system: The spreadsheet we created of the Mondo Spider's DC motor was also used to specify an adequate gear pump to use for our testing of the PID controller we made. Based on the pressure requirements for our system, which will generally be around 2000 - 3000 psi, we calculated how much torque we would require from the motor and the power output of the gear pump. For example, if we want to operate our hydraulic system at 3000 psi, we will require 15.87 Nm from the motor. This means the speed of the motor will be 3248.65 RPM which will produce a flow rate of 13.31 litres/minute. At these conditions, the power output of the pump is 4588.1 W.The table below is a section of the spreadsheet in Appendix C, which computes all these values for different pressures.

Pressure (psi)	Pump Torque[with Eff.] (Nm)	Motor RPM (from other worksheet)	Flow rate (Gpm)	Flow rate (Lpm)	Pump Power (W)
3000.00	15.87	3248.65	3.52	13.31	4588.10

Table 1: Hydraulic Gear Pump specifications

PID control: We tested our PID controller in the interest of acquiring the most common results when characterizing a PID control system. It is common to determine specifications such as the rise time, settling time, percent overshoot as well as the type of the response of the system to a step input. Furthermore, we collected data to analyze its ramp response.

We collect our data in the following way: A pressure transducer outputs a current which is proportional to the pressure it reads. The current is translated into a voltage, which is then read by the Arduino microcontroller as the Input signal. The microcontroller is also reading four voltage signals from the PID Box that uses potentiometers to represent P, I, and D values and the pressure Setpoint. Using these values, the microcontroller calculates the necessary Output signal to send to the motor controller in order to reach the target Setpoint. Using a LabJack device, we measure the Input voltage that corresponds to the system pressure, the Setpoint voltage, and the Output voltage, which is a PWM signal. The Labjack can plot the values in real time in addition to saving the data in a file that can be graphed and analyzed later. The real-time plot allows us to tune the PID by observing the pressure response to the setpoint.

Designs / Rationale:

DC motors: One of the reasons our sponsor chose to use DC motors to power Prosthesis was that they allow for bursts of higher power than normal operation. This means that a motor can be chosen that will run at an optimal efficiency during normal operation and that will be able to exceed its normal output power when required.

Hydraulic system: Originally, Prosthesis was going to be using gear pumps. What this meant was that the motor would need to vary its operating speed as more power was needed. However, a DC motor cannot spin faster than its rated RPM without the risk of overheating and potentially melting. In order to safely use a gear pump, the chosen motor would need to be rated to continuously output a power that Prosthesis would only need for very short periods of time. This posed an issue as such a motor would be run inefficiently during normal operation.

The determined solution was to use an axial piston pump and the motor speed instead of the hydraulic pressure as the input parameter for our PID control. Not only did the axial pump allow its displacement to change, but the displacement could be adjusted by angling the swash plate during operation to control pressure and flow rate. A hydraulic feedback could be established to change the swash angle using a pressure difference. By holding the motor speed at a certain value, which could be chosen for optimal motor efficiency, the displacement would change according to the flow rate in Prosthesis' hydraulic lines. This change would have a direct effect on the fluid pressure, resulting in a closed loop control system that would maximize the total system efficiency.

It was decided by our project sponsor that the final system should use axial piston pumps. However, we could not get them before the end of our project term. For this reason, we worked on a pressure PID control system for gear pumps so that we could observe how a Load Sense configuration might work.

Control Panel: The control system we designed required an interface for easy calibration and tuning of the control parameters. In conjunction with our project sponsor, we designed a control panel that could be used as a practical readout and tuning module. Figure 6 shows the design for the general layout of components on the panel. Six 7 segment displays for each motor and pump are used to read critical values of the power pack. Two 7 segment displays are used to read the battery voltage and hydraulic fluid temperature. 2 Mode switches are used to switch between the constant pressure and Load Sense operating modes; LED's are used to indicate which mode is engaged. The switches placed beside the Set Pressure and Load Sense numerical displays are used to increase those values. The choice of the switches was not a trivial task since they have to be capable of enduring oil or any other kind of fluid spills. We chose

switches that are rated to be IP-5/6, which means that they are both resistant to spray from water and protected from dust particles. The panel also has two ports for connectors to be used for the PID box. Whenever the system is rebooted and the PID box is connected, the microcontroller will read the P, I, and D values from the PID box, otherwise it will use the most recent values stored in the code. There are various USB ports which can be used to reprogram the Arduinos if need be.

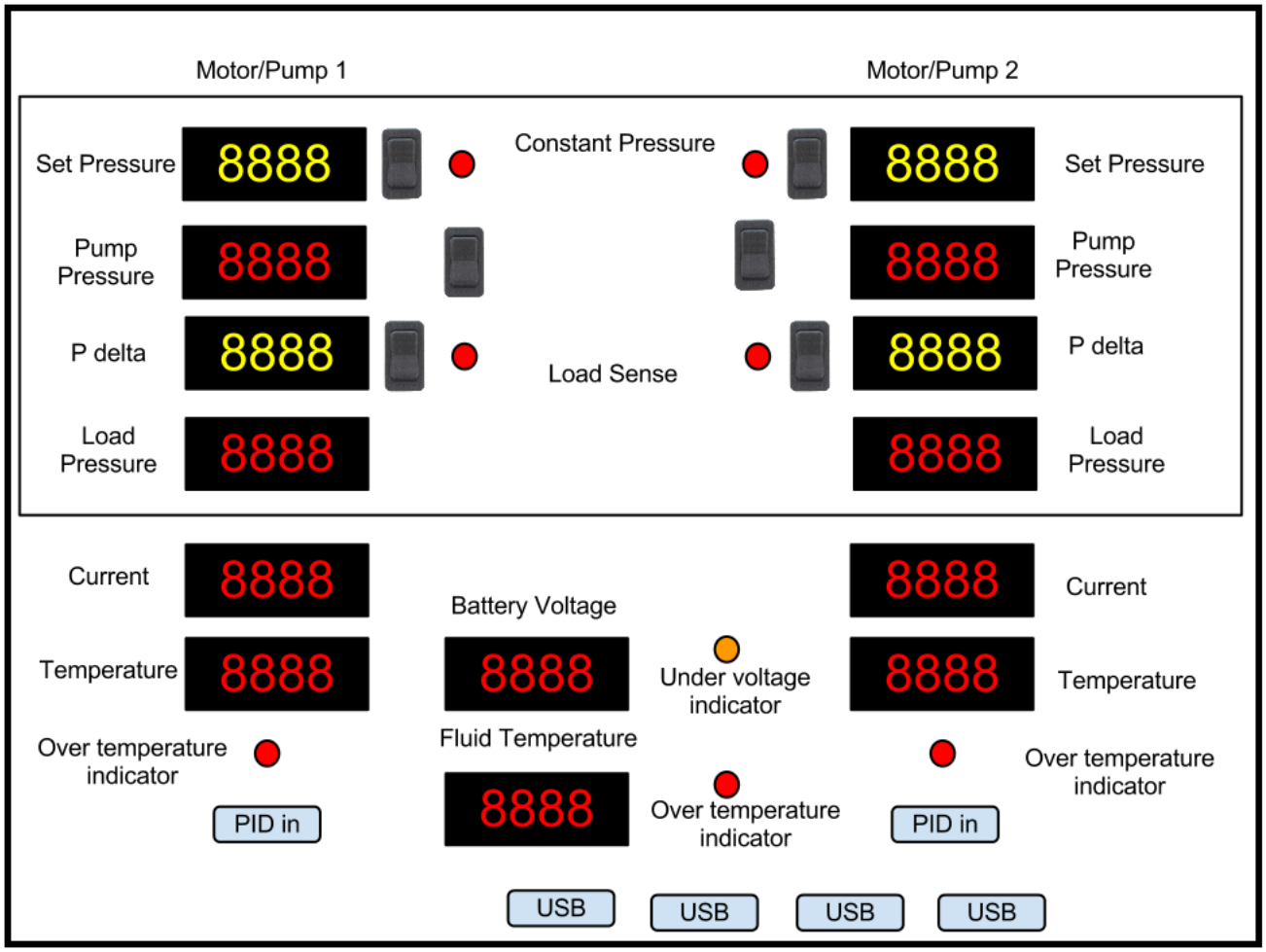


Figure 6: Design for the Control Panel

PID: The PID pressure control system uses a pressure sensor at the pump outlet, control buttons and knobs from the control panel, and, for one of the settings, a pressure sensor somewhere in the hydraulic line.
The PID pressure control system operates in two separate modes: Constant Pressure and Load Sense.
Constant Pressure mode is the simpler approach, maintaining some user-defined constant pressure at the pump outlet. This control option is primarily intended for testing purposes, as maintaining a high pressure during periods of low activity is inefficient. Constant Pressure is the simpler approach.
Load Sense mode, instead of keeping a constant pressure, maintains a constant (user-defined) pressure difference between the pump outlet and the other pressure sensor in the hydraulic line. In this way, the pump pressure adapts to the load seen by the line. In periods of reduced operation, the load is small, and thus the motor and pump are not required to work hard.

Experimental Equipment/ Flow Diagrams/ algorithms:

Figure 7 shows how the basic components of the power pack are connected and the flow of information in our control system.

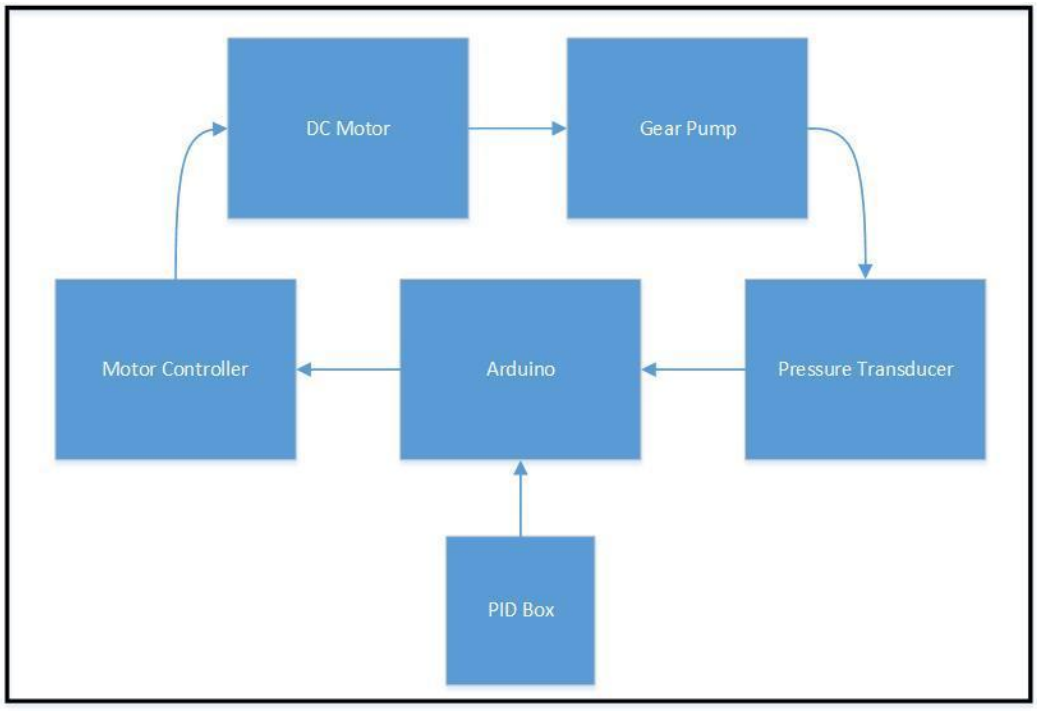


Figure 7: Diagram of Information Flow in the Control System

PID Control Algorithm:
1) Pressure transducer reads pressure in hydraulic lines and informs Arduino.
2) Arduino reads proportional, integral and derivative (PID) terms and setpoint from box
3) Arduino calculates an error and sends PWM signal to motor controller
4) Motor Controller (programmed to accept PWM signals) provides power from the batteries proportional to the PWM signal to the motor
5) Motor turns gear pump to increase pressure in the hydraulic lines
Note: If the pressure needs to decrease, the motor controller simply tells the motor to stop working, and the pressure falls accordingly.

As mentioned earlier in the report, there are two possible operating modes for the power pack controller. The diagram in Figure 8 demonstrates how the code for interfacing with our control panel works.

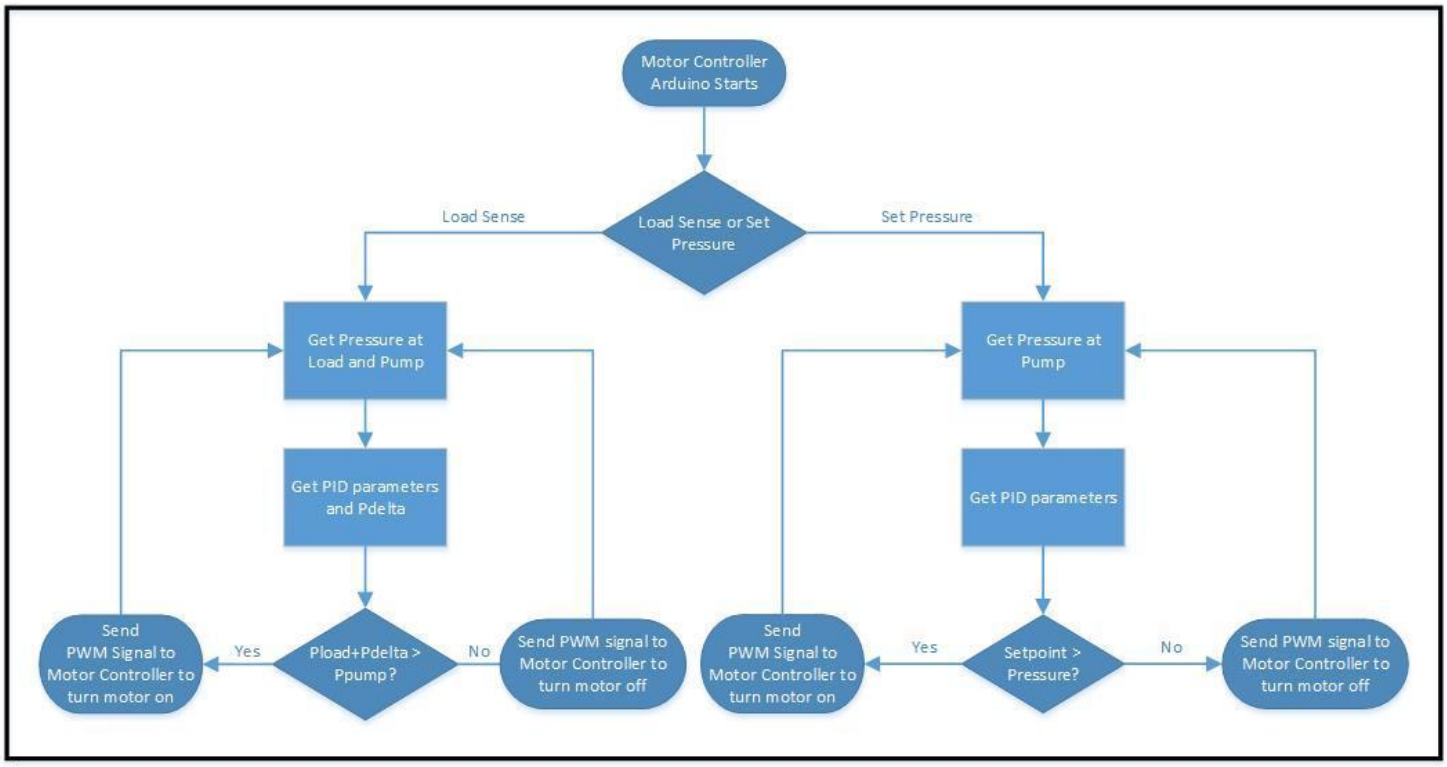


Figure 8: Code Diagram for Load Sense vs. Set Pressure

Control Panel: This mock-up control box panel (Figure 9) was constructed as a 2nd prototype for the final panel that will be eventually located on the machine. As of the date of the picture taken below, many of the primary components had not yet been delivered. Thus, the prototype is missing multiple parts. In the final version, the prototype box will resemble the panel schematic (Figure 6). Also pictured in Figure 8 is the PID tuning box, which allows the users to set and tune the PID controller

parameters to optimize response. This box (or some version of it) will also be a working part of the final machine. The box will also be detachable, allowing tuning parameters to be saved.

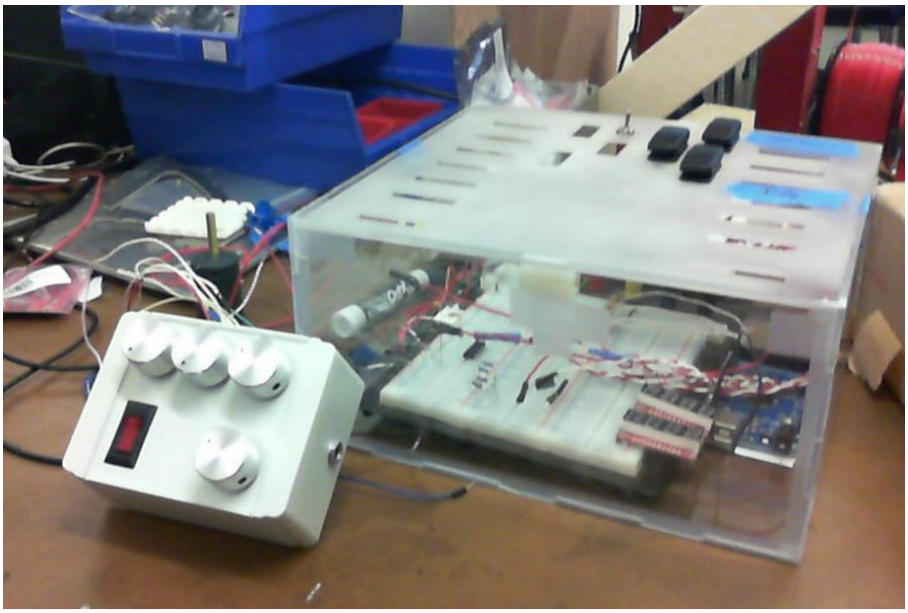


Figure 9: Prototype for Control Panel and PID Tuning Box

Results:

DC motors: From our calculations we found that the EMC-RT 200 is the most efficient when spinning at 3471 RPM and producing a torque of 1.694 Nm. However, the EMC-RT 200's will most likely be operating around 3000 RPM, which means they will be producing a torque of 29 Nm while being 75% efficient. Please refer to Appendix A for the complete spreadsheet containing all data and performance curves for the EMC-RT 200. Figure 10 shows the motors that were chosen for Prosthesis.

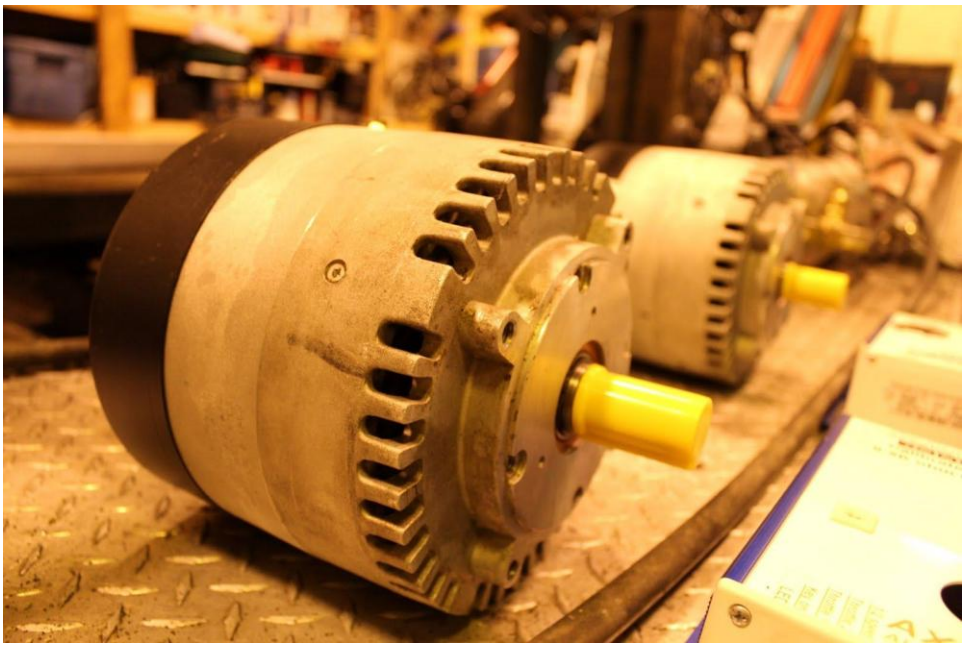


Figure 10: EMC-RT 200 (Prosthesis Motors)

Hydraulic system: The gear pump we decided upon was purchased and used to test the PID control system. This test system was designed to mimic how the final version of Prosthesis would behave. The system worked well for the tests, and allowed us to use one of the Mondo Spider's motors to test the PID up to 2000 psi.

PID: Data was collected to capture and analyze the response of our tunable PID system. For the following graphs, the P, I and D parameters were set constant and the setpoint was varied. Both the step response and the ramp response were analyzed.

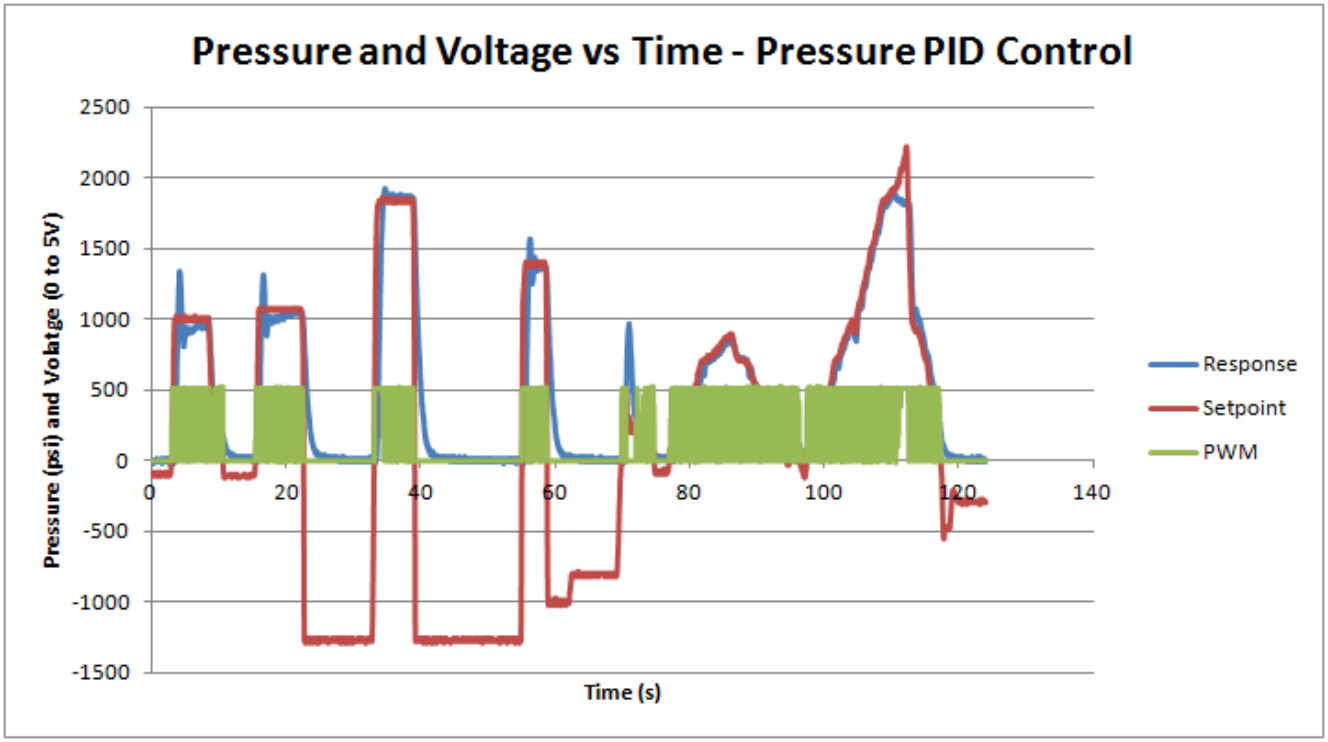


Figure 11: PID test results

In Figure 11 above, we can observe our control system's step response. In the section of time shown, the step response is observed five times. Qualitatively, the response seems to be critically damped. The next graph shows a smaller time range and allows to see this characteristic more clearly. After t=80 s, the ramp response is shown. As can be seen our controller follows the setpoint very closely. Note that the PWM signal is also shown in the graph. It was scaled (multiplied by 100) for clarity.

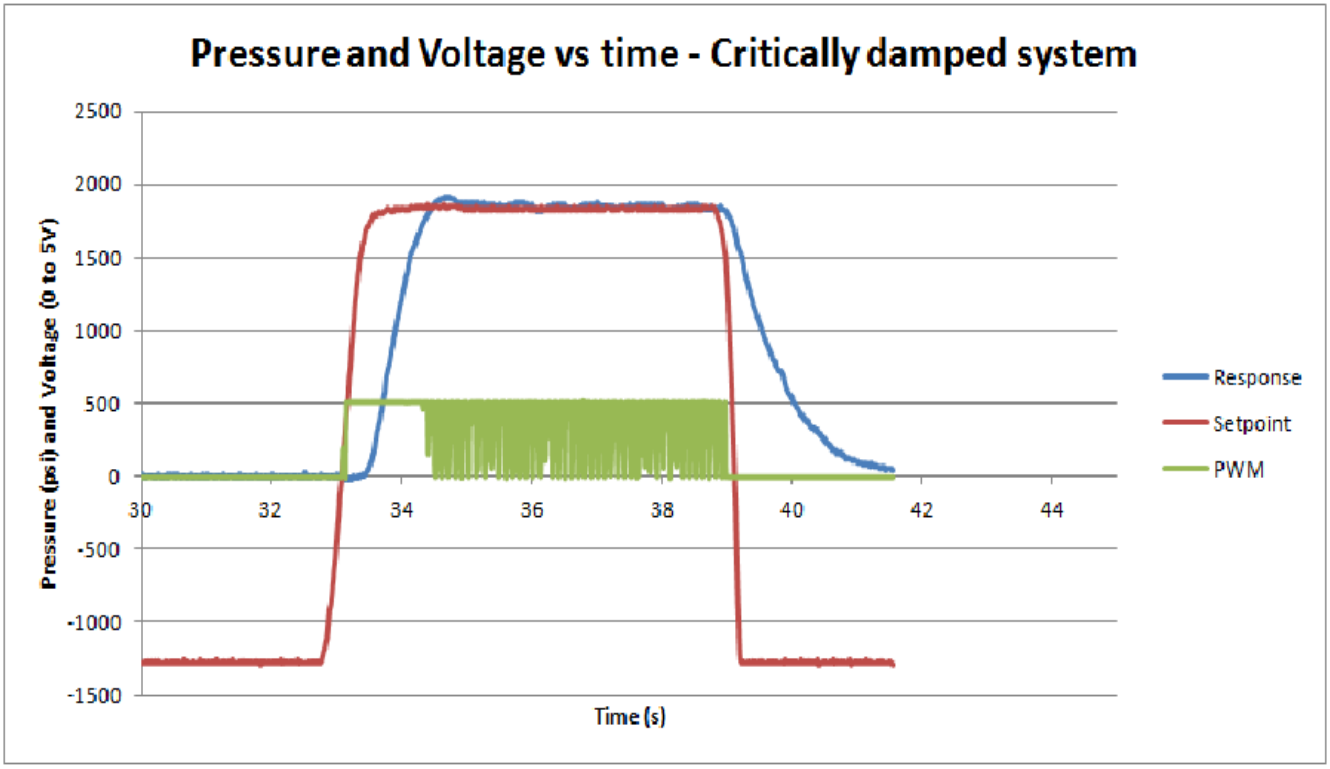


Figure 12: Critically damped response

In Figure 12, a shorter section of time is shown. It is easier to observe the critically damped response to a step function input. From this data, the following parameters were calculated:

2% settling time: 2.82 seconds

Rise time (0-100%): 1.38 seconds

%Overshoot: 3.88%

Discussion of Results:

DC motors: The calculations we made are very valuable to our project sponsor, since they show all possible settings of the EMC-RT 200 assuming the motor is held constant at 72V DC. Given the current provided to the motor, one can use our spreadsheet to look up the torque, speed, and power that the motor produces including how efficient it is at that operating point. The calculations and performance curves for the EMC-RT 200 and Spider motors assume a couple of things. First, we have assumed that the torque and speed relationship is linear from 0 to stall torque. Also, we assume that the motors will be operating at constant voltage and varying current. All the calculations are based on the information provided in the datasheets and we have not done tests on the motors to confirm this information. This can be easily measured once all the sensors are calibrated and attached to the components of the electric power pack.

PID: In a hydraulic system like the one in Prosthesis, parameters like the rise time and %overshoot are the most important. Since the Setpoint pressure will most likely be changing rapidly, the settling time won't be very important. What does matter is how fast we can catch up to the required pressure. It is important to note that we tested our control system by moving hydraulic cylinders on the leg of Prosthesis (the Alpha leg). Due to time constraints, we did not have a chance to test the system with different loads. Testing under these conditions would be beneficial to better understand how the system will respond under varying loads.

Conclusion

DC motors: The EMC-RT 200 is the most efficient (86%) when running at 3470 RPM. This operating point produces almost no torque. Since the axial piston pump to be used in the final power pack has a nominal RPM of 3000 RPM, this will likewise be the speed at which the EMC-RT 200 will spin. In order to achieve this, the motor has to be fed 170A at 72V, producing a torque of 29 Nm at 75% efficiency.

PID: The PID control system maintains the pressure very close to the Setpoint. When the there is a sudden demand for an increase in pressure, the controller takes 1.38 seconds to reach the Setpoint, overshoots but stays within 3.88% of the Setpoint and takes 2.82 seconds to match it exactly.

Project Deliverables

List of Deliverables:

Original Deliverables section from Proposal: “By the end of the project, a Raspberry Pi based control system will be ready to be installed on Prosthesis. The control aspect of this project can be described as a closed loop pressure feedback control system. The only control required by the project sponsor consists of controlling two pumps in order to increase or decrease the pressure in the hydraulic lines based on a desired value input by the user. Apart from this control component, the system can be better described as an input/output device which is used to monitor sensor data and drive outputs in response to specific inputs. Since the machine will not be completely built by the time our project ends, there will be no components on which to mount our control system. The configured R.Pi computer and Arduinos including a touchscreen will be delivered to our project sponsor. In addition, a user manual for the control system will be presented. The R.Pi will be capable of driving a touchscreen interface, communicate with multiple subsystems and enable and disable outputs based on sensor data. The Arduino microcontrollers will be setup to perform PID control of the electric motors based on the feedback provided by the pressure sensors.”

Changes made to deliverables: We were no longer required to setup a touchscreen interface for the project or dealing with the with the raspberry pi and Arduinos communication. Those tasks were delegated to other people working on Prosthesis.

Our project sponsor asked us to concentrate on the operation of the power pack instead. Here is a list of the components we will hand to our project sponsor at the end of the term.

1 Arduino with the PID control code
2 Spreadsheets with DC motors and pump parameters
3 Control Panel prototype with cutouts for the 7 segment displays and switches.
4 All sensors and electrical components contained in the financial summary

Financial summary:

#	Description	Quantity	Vendor	Cost (per unit)	Purchased by:	To be funded by:
1	Arduino Mega	2	Unknown	Unknown	Project Lab	
2	Arduino Mega Shield	4	Unknown	Unknown	Project Lab	
3	4 digit 7 segment display	2	Robotshop	13.6	Project Lab	
4	DC/DC converter	2	RobotShop	25	Project Lab	
5	Arduino CAN Bus Shield	1	RobotShop		Project Lab	
6	Momentary Rocker Switch	9	Digikey	2.94	Project Lab	
7	On/Off Rocker Switch	2	Digikey	2.90	Project Lab	
8	Current Sensors	2	Digikey	29.29	Project Lab	
9	IC temperature sensor	12	Digikey	3.76	Project Lab	

Note: we will leave the "to be funded by" section of the financial summary empty since we don't know which components will be paid by the project lab and which components will be charged to our project sponsor.

Ongoing Commitments by team members:

This will be discussed with our project sponsor at the end of the term. If we stay on board, we anticipate helping out with the following aspects of the project:

- Maintain code for Arduino motor controller
- Transfer pressure PID controller to RPM PID controller
- Construction of final control panel (Including electronics and hardware)

Recommendations

1. Proceed with the 2nd prototype of the control panel for testing. Once everything is working, start making third or final version of the control dashboard module. Use isolating connectors if the box will be made out of metal, or use an isolating material (like the one we used on the prototype). Inside the box, have an efficient way to route the wires, otherwise they become jumbled and a headache to troubleshoot the circuits.
2. When the axial piston pumps arrive, the PID code will need to be altered to control the motor RPM. Right now, the Arduino measures a voltage and calculates a pressure from that value, but for the RPM, the Arduino will already be counting the revolutions in a given time. This means that the mapping will need to be changed. A suitable speed required to get the desired pressure will also need to be determined.
3. Use proper current sensors for the wire gauge that will be used to connect the high current components.
4. Look into EEPROM for Arduino to saving PID values on Arduino.

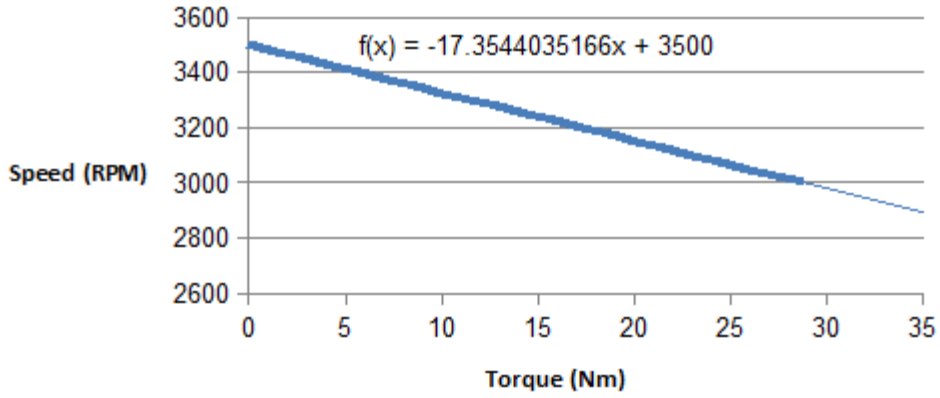
Appendices

Appendix A - EMC-RT 200 Permanent Magnet DC Motor Calculations and Performance Curves

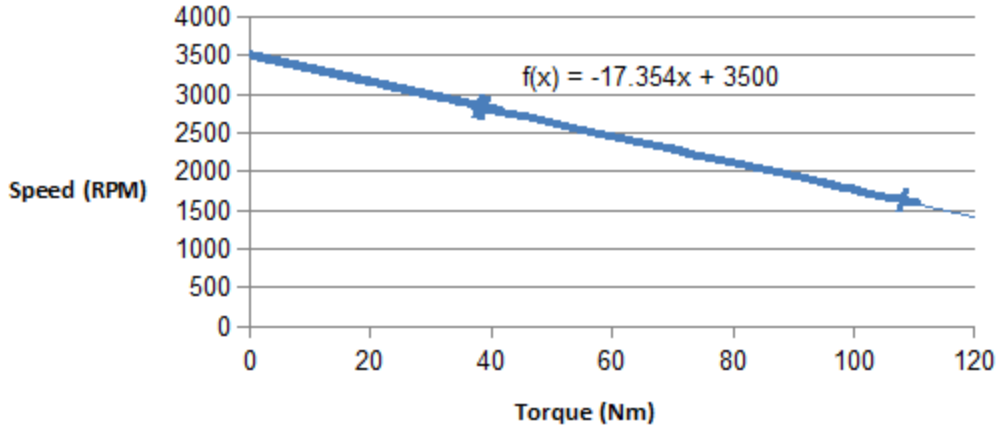
Torque (lb in)	Torque (Nm)	Current (A)	Voltage (V)	Extrapolated Speed (RPM)	Power out (W)	Efficiency
0.00	0.00	0.00	72.00	3500.00	0.00	0.00
15.00	1.69	10.00	72.00	3470.59	615.95	0.86
30.00	3.39	20.00	72.00	3441.18	1221.45	0.85
45.00	5.08	30.00	72.00	3411.77	1816.52	0.84
60.00	6.78	40.00	72.00	3382.36	2401.15	0.83
75.00	8.47	50.00	72.00	3352.94	2975.34	0.83
90.00	10.17	60.00	72.00	3323.53	3539.09	0.82
105.00	11.86	70.00	72.00	3294.12	4092.40	0.81
120.00	13.56	80.00	72.00	3264.71	4635.27	0.80
135.00	15.25	90.00	72.00	3235.30	5167.70	0.80
150.00	16.95	100.00	72.00	3205.89	5689.69	0.79
165.00	18.64	110.00	72.00	3176.48	6201.24	0.78
180.00	20.34	120.00	72.00	3147.07	6702.35	0.78
195.00	22.03	130.00	72.00	3117.66	7193.02	0.77
210.00	23.73	140.00	72.00	3088.24	7673.26	0.76
225.00	25.42	150.00	72.00	3058.83	8143.05	0.75
240.00	27.12	160.00	72.00	3029.42	8602.40	0.75
255.00	28.81	170.00	72.00	3000.01	9051.32	0.74

270.00	30.51	180.00	72.00	2970.60	9489.79	0.73
285.00	32.20	190.00	72.00	2941.19	9917.83	0.72
300.00	33.90	200.00	72.00	2911.78	10335.42	0.72
315.00	35.59	210.00	72.00	2882.37	10742.58	0.71
330.00	37.28	220.00	72.00	2852.96	11139.30	0.70
345.00	38.98	230.00	72.00	2823.55	11525.57	0.70
360.00	40.67	240.00	72.00	2794.13	11901.41	0.69
375.00	42.37	250.00	72.00	2764.72	12266.81	0.68
885.07	100.00	590.05	72.00	1764.60	18478.85	0.43
955.88	108.00	637.25	72.00	1625.77	18387.00	0.40

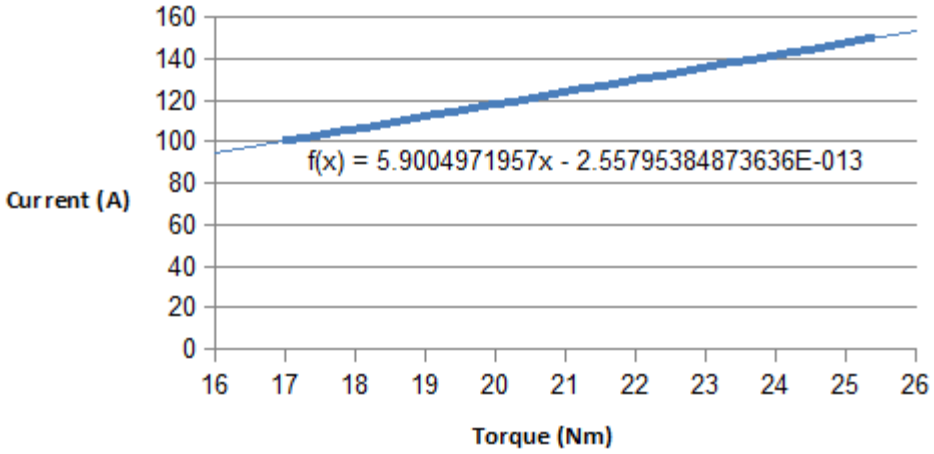
EMC-RT200 - Speed vs Torque from datasheet



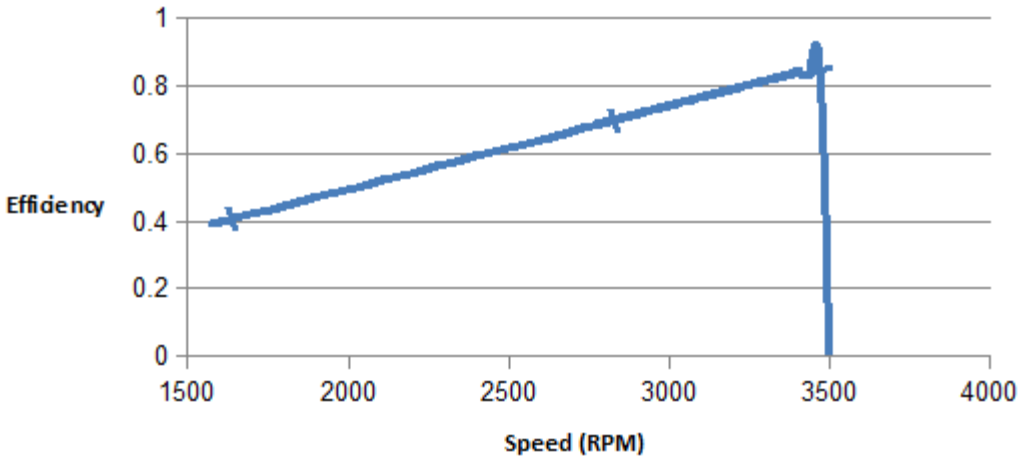
EMC-RT200 - Speed vs Torque extrapolated

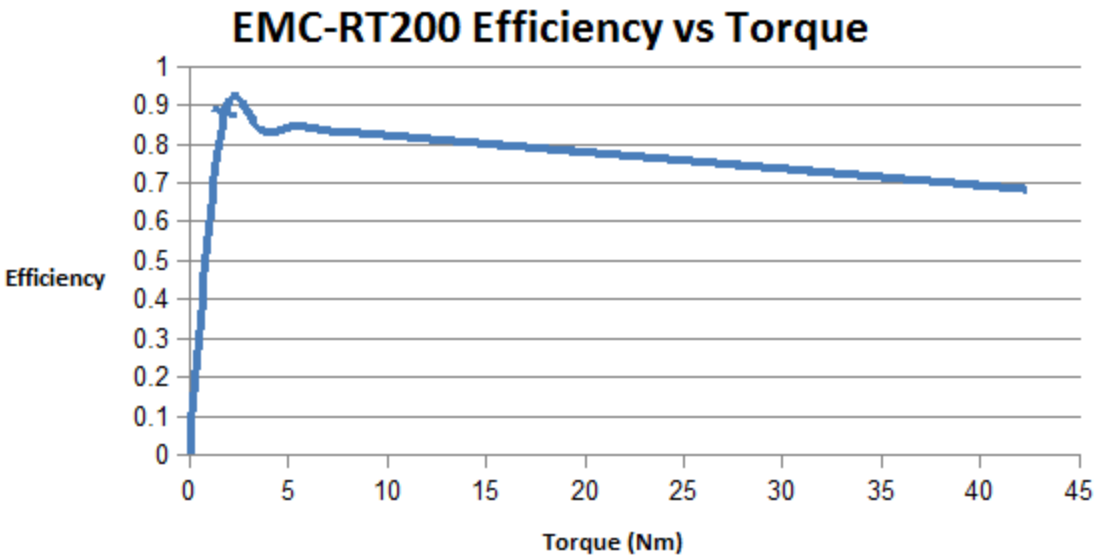


EMC-RT200 Current vs Torque



EMC-RT200 Efficiency vs Speed



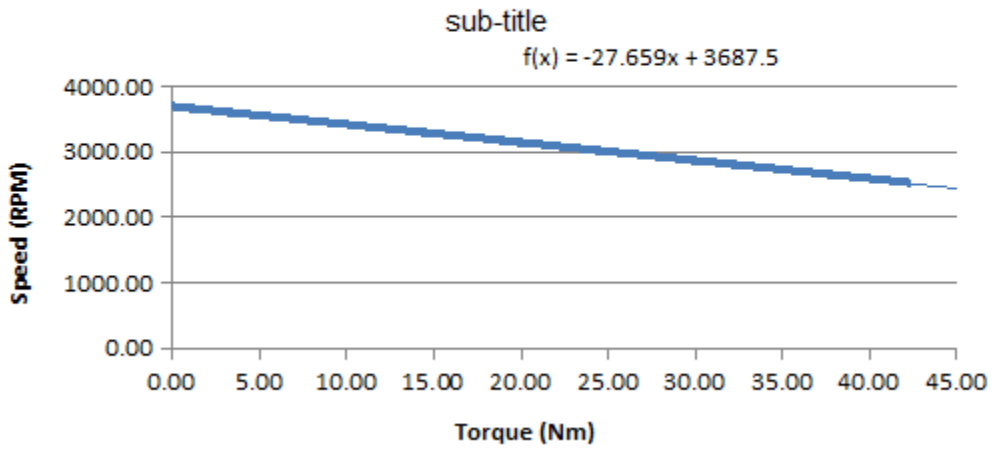

EMC-RT200 Efficiency vs Torque
Efficiency
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30
35
40
45
Torque (Nm)

Appendix B - Mondo Spider DC motor Calculations and Graphs

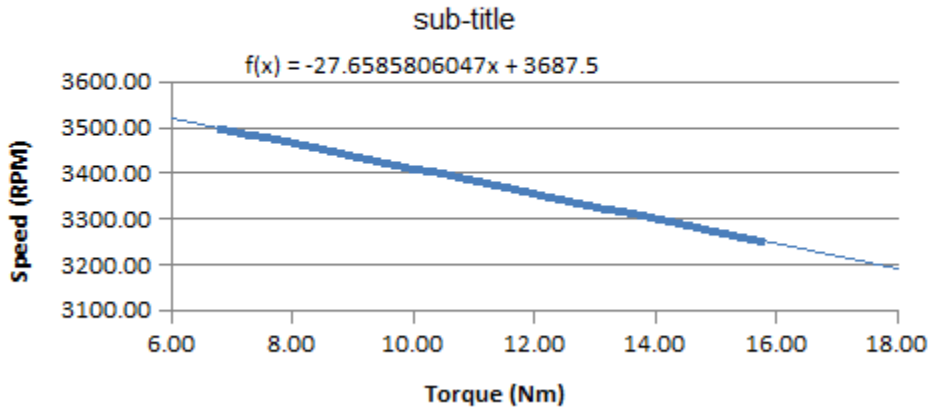
Torque (lb in)	Torque (Nm)	Current (A)	Voltage (V)	Extrapolated Speed (RPM)	Power out (W)	Efficiency
0.00	0.00	7.50	48.00	3687.50	0.00	0.00
15.00	1.69	20.62	48.00	3640.62	646.12	0.65
30.00	3.39	33.75	48.00	3593.75	1275.61	0.79
45.00	5.08	46.87	48.00	3546.87	1888.46	0.84
60.00	6.78	60.00	48.00	3500.00	2484.66	0.86
75.00	8.47	73.12	48.00	3453.12	3064.23	0.87
90.00	10.17	86.25	48.00	3406.25	3627.16	0.88
105.00	11.86	99.37	48.00	3359.37	4173.46	0.87
120.00	13.56	112.50	48.00	3312.49	4703.11	0.87
135.00	15.25	125.62	48.00	3265.62	5216.12	0.87
140.00	15.82	130.00	48.00	3249.99	5383.43	
150.00	16.95	138.75	48.00	3218.74	5712.50	0.86
165.00	18.64	151.87	48.00	3171.87	6192.24	0.85
180.00	20.34	165.00	48.00	3124.99	6655.34	0.84
195.00	22.03	178.12	48.00	3078.12	7101.80	0.83
210.00	23.73	191.25	48.00	3031.24	7531.62	0.82
225.00	25.42	204.37	48.00	2984.36	7944.80	0.81
240.00	27.12	217.50	48.00	2937.49	8341.35	0.80
255.00	28.81	230.62	48.00	2890.61	8721.25	0.79
270.00	30.51	243.75	48.00	2843.74	9084.52	0.78

285.00	32.20	256.87	48.00	2796.86	9431.15	0.76
300.00	33.90	270.00	48.00	2749.99	9761.14	0.75
315.00	35.59	283.12	48.00	2703.11	10074.49	0.74
330.00	37.28	296.25	48.00	2656.23	10371.20	0.73
345.00	38.98	309.37	48.00	2609.36	10651.28	0.72
360.00	40.67	322.50	48.00	2562.48	10914.71	0.71
375.00	42.37	335.62	48.00	2515.61	11161.51	0.69

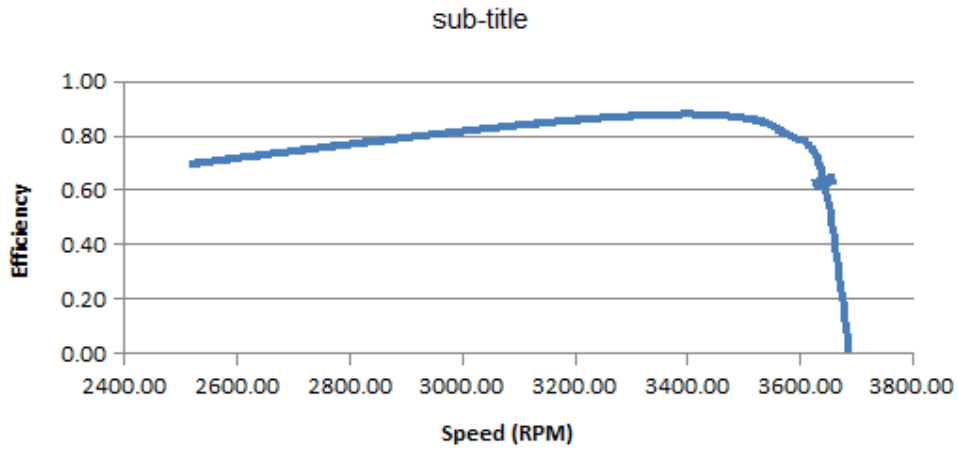
Spider Motor - Speed vs Torque extrapolated



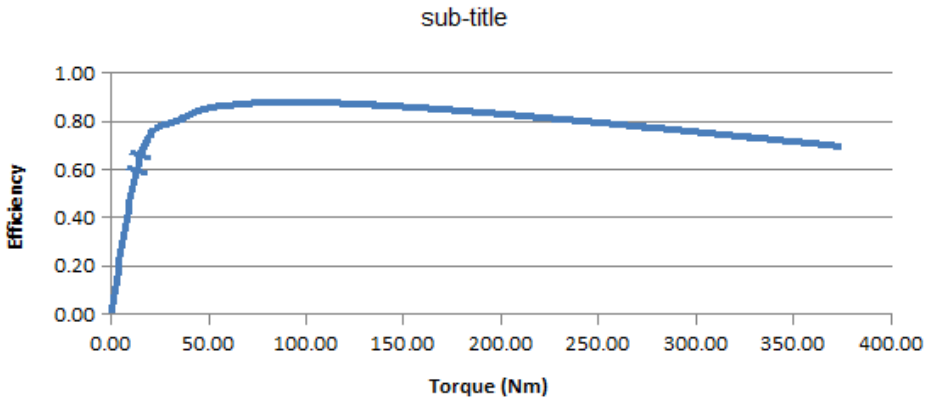
Spider motor - Speed vs Torque from datasheet



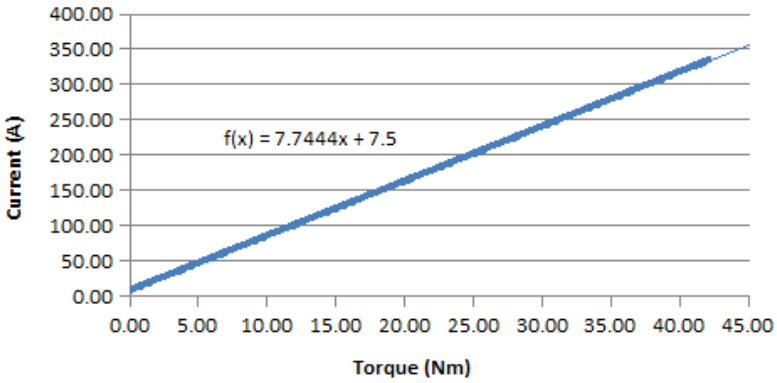
Spider Motor - Efficiency vs Speed



Spider Motor - Efficiency vs Torque



Spider Motor - Current vs Torque



Appendix C - 8.5 GPM Hydraulic Gear Pump Calculations

Pressure (psi)	Pump Torque[with Eff.] (Nm)	Motor RPM (from other worksheet)	Flow rate (Gpm)	Flow rate (Lpm)	Pump Power (W)
3200.00	16.92	3219.39	3.48	13.19	4849.90
3150.00	16.66	3226.70	3.49	13.22	4784.96
3100.00	16.40	3234.02	3.50	13.25	4719.68
3050.00	16.13	3241.33	3.51	13.28	4654.06
3000.00	15.87	3248.65	3.52	13.31	4588.10
2950.00	15.60	3255.96	3.52	13.34	4521.79
2900.00	15.34	3263.27	3.53	13.37	4455.13
2850.00	15.07	3270.59	3.54	13.40	4388.13
2800.00	14.81	3277.90	3.55	13.43	4320.79
2750.00	14.54	3285.22	3.56	13.46	4253.10
2700.00	14.28	3292.53	3.56	13.49	4185.07
2650.00	14.02	3299.85	3.57	13.52	4116.69
2600.00	13.75	3307.16	3.58	13.55	4047.97
2550.00	13.49	3314.47	3.59	13.58	3978.91
2500.00	13.22	3321.79	3.60	13.61	3909.50
2450.00	12.96	3329.10	3.60	13.64	3839.74
2400.00	12.69	3336.42	3.61	13.67	3769.65
2350.00	12.43	3343.73	3.62	13.70	3699.20
2300.00	12.16	3351.05	3.63	13.73	3628.42

2250.00	11.90	3358.36	3.63	13.76	3557.28
2200.00	11.64	3365.67	3.64	13.79	3485.81
2150.00	11.37	3372.99	3.65	13.82	3413.99
2100.00	11.11	3380.30	3.66	13.85	3341.83
2050.00	10.84	3387.62	3.67	13.88	3269.32
2000.00	10.58	3394.93	3.67	13.91	3196.46
1950.00	10.31	3402.25	3.68	13.94	3123.27
1900.00	10.05	3409.56	3.69	13.97	3049.73
1850.00	9.78	3416.87	3.70	14.00	2975.84
1800.00	9.52	3424.19	3.71	14.03	2901.61
1750.00	9.26	3431.50	3.71	14.06	2827.04
1700.00	8.99	3438.82	3.72	14.09	2752.12
1650.00	8.73	3446.13	3.73	14.12	2676.85
1600.00	8.46	3453.44	3.74	14.15	2601.25
1550.00	8.20	3460.76	3.75	14.18	2525.29
1500.00	7.93	3468.07	3.75	14.21	2449.00
1450.00	7.67	3475.39	3.76	14.24	2372.36
1400.00	7.40	3482.70	3.77	14.27	2295.37
1350.00	7.14	3490.02	3.78	14.30	2218.04
1300.00	6.88	3497.33	3.78	14.33	2140.37
1250.00	6.61	3504.64	3.79	14.36	2062.35
1200.00	6.35	3511.96	3.80	14.39	1983.99
1150.00	6.08	3519.27	3.81	14.42	1905.28
1100.00	5.82	3526.59	3.82	14.45	1826.23

1050.00	5.55	3533.90	3.82	14.48	1746.84
1000.00	5.29	3541.22	3.83	14.51	1667.10
950.00	5.02	3548.53	3.84	14.54	1587.01
900.00	4.76	3555.84	3.85	14.57	1506.59
850.00	4.50	3563.16	3.86	14.60	1425.81
800.00	4.23	3570.47	3.86	14.63	1344.70
750.00	3.97	3577.79	3.87	14.66	1263.24
700.00	3.70	3585.10	3.88	14.69	1181.43
650.00	3.44	3592.42	3.89	14.72	1099.28
600.00	3.17	3599.73	3.90	14.75	1016.79
550.00	2.91	3607.04	3.90	14.78	933.95
500.00	2.64	3614.36	3.91	14.81	850.77
450.00	2.38	3621.67	3.92	14.84	767.24
400.00	2.12	3628.99	3.93	14.87	683.37
350.00	1.85	3636.30	3.94	14.90	599.15
300.00	1.59	3643.61	3.94	14.93	514.59
250.00	1.32	3650.93	3.95	14.96	429.69
200.00	1.06	3658.24	3.96	14.99	344.44
150.00	0.79	3665.56	3.97	15.02	258.85
100.00	0.53	3672.87	3.97	15.05	172.91
50.00	0.26	3680.19	3.98	15.08	86.63
0.00	0.00	3687.50	3.99	15.11	0.00

Appendix D - Motor Controller code CPP file

```
#include "Motor_Controller.h"

MotorController::MotorController(int ModePin, int MotorPin, int InputPressurePin, int
LoadSensePressurePin, int TuningBoxPin, int KpPin, int KiPin, int KdPin, double Kp,
double Ki, double Kd, double dt, double LoadSenseOffset, double NumericalSetpoint,
double Input, double Output, double Setpoint)
: mPIDControl(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT),
mModePin(ModePin),
mMotorPin(MotorPin),
mInputPressurePin(InputPressurePin),
mLoadSensePressurePin(LoadSensePressurePin),
mTuningBoxPin(TuningBoxPin),
mKpPin(KpPin),
mKiPin(KiPin),
mKdPin(KdPin),
mKp(Kp),
mKi(Ki),
mKd(Kd),
mdt(dt),
mLoadSenseOffset(LoadSenseOffset),
mNumericalSetpoint(NumericalSetpoint),
mInput(Input),
mSetpoint(Setpoint),
mOutput(Output){}

const double MotorController::PID_Pot_Sensitivity = .02;
const double MotorController::Setpoint_Pot_Sensitivity = 2.44375;
const double MotorController::Pressure_Sample_Count = 20.0;
const double MotorController::Pressure_Sensitivity = 2500.0;
const double MotorController::Pressure_Intercept = 1230.0;
const double MotorController::Analog_to_Voltage = 0.004887586;

void MotorController::Initialize()
{
       mPIDControl.SetMode(AUTOMATIC);
       mPIDControl.SetSampleTime(mdt);
       mPIDControl.SetTunings(mKp, mKi, mKd);
}

void MotorController::Iterate()
{
  if(digitalRead(mTuningBoxPin) == HIGH)
  {
       mKp = analogRead(mKpPin)*PID_Pot_Sensitivity;
       mKi = analogRead(mKiPin)*PID_Pot_Sensitivity;
       mKd = analogRead(mKdPin)*PID_Pot_Sensitivity;
       mPIDControl.SetTunings(mKp, mKi, mKd);
  }
```

```
  if(digitalRead(mModePin) == HIGH)
  {
       mSetpoint = GetPressure(mLoadSensePressurePin) + mLoadSenseOffset;
  }
  if(digitalRead(mModePin) == LOW)
  {
       mSetpoint = mNumericalSetpoint;
  }

  mInput = GetPressure(mInputPressurePin);
  mPIDControl.Compute();
  analogWrite(mMotorPin, mOutput);
}

double MotorController::GetPressure(int Pin)
{
  int volt_sum = 0;
  for(int i=0; i<int(Pressure_Sample_Count); i++){
       volt_sum = volt_sum + analogRead(Pin);
  }

  return (volt_sum/Pressure_Sample_Count)*Pressure_Sensitivity*Analog_to_Voltage-
Pressure_Intercept;
}

void MotorController::SetLoadSenseOffset(double NewOffset)
{
  mLoadSenseOffset = NewOffset;
}

void MotorController::SetNumericalSetpoint(double NewSetpoint)
{
  mNumericalSetpoint = NewSetpoint;
}
```

Appendix E - Motor Controller code header file

```
#ifndef __MOTOR_CONTROLLER_H_
#define __MOTOR_CONTROLLER_H_

#include <Arduino.h>
#include <PID_v1.h>

class MotorController
{

  public:

      MotorController(int ModePin, int MotorPin, int InputPressurePin, int
LoadSensePressurePin, int TuningBoxPin, int KpPin, int KiPin, int KdPin, double Kp,
double Ki, double Kd, double dt, double LoadSenseOffset, double NumericalSetpoint,
double Input, double Output, double Setpoint);

      void Initialize();

      void Iterate();

      double GetPressure(int Pin);

      void SetLoadSenseOffset(double NewOffset);

      void SetNumericalSetpoint(double NewSetpoint);

  private:
      int mModePin;
      int mMotorPin;
      int mInputPressurePin;
      int mLoadSensePressurePin;
      int mTuningBoxPin;
      int mKpPin;
      int mKiPin;
      int mKdPin;
      double mKp;
      double mKi;
      double mKd;
      double mPressure;
      double mdt;

      double mLoadSenseOffset;
      double mNumericalSetpoint;

      double mInput;
      double mOutput;
      double mSetpoint;

      PID mPIDControl;
```

```
	//conversion constants
	static const double PID_Pot_Sensitivity;
	static const double Setpoint_Pot_Sensitivity;
	static const double Pressure_Sample_Count; //double instead of int to avoid
integer division
	static const double Pressure_Sensitivity;
	static const double Pressure_Intercept;
	static const double Analog_to_Voltage;

};
#endif __MOTOR_CONTROLLER_H_
```