CLEVELAND STATE UNIVERSITY

Title: CSU MCE#698 Master's Project
Dissertation: Designing, modeling, and embedded programming of torque limiting brushless motor system.
Professor: Dr.Majid Rashidi
Graduate Student: Daniel Izadnegahdar
Project Start Date: 06/01/16
Project Completion Date: 12/14/16
Last Update: (REV-C) 11/21/16
Cover Image: [Appliance brushless motor] Model courtesy of JRaymond, rendering by DIZAD.

Table of Content:

Report Notes: *This report discusses the step by step process of completing this project. All of the documents (wire diagrams, SolidWorks models, programs, simulations, and excel studies) were created by the graduate student Daniel Izadneghadar unless specified otherwise. Daniel was under the supervision of several technical advisors (software, electronics, and mechanical) for the completion of this project. This project took a total of 200 hours to complete over the course of 6 months.*

1. **Proposal Summary:**
   1.1. This project proposes a system to precisely control the torque of an automatic screwdriver by using a brushless motor and electronic feedback system.

2. **Assess:**
   2.1. **Background:**
      2.1.1. **Applications:** There are several applications for precisely controlling the torque of a motor. Below are three common examples:
         2.1.1.1. Touch Sensing: Robots have become more popular for working side-by-side with operators and assist with some of the routine tasks of manufacturing. Examples of such robots include the *Kuka* 7-DOE robotic arm and the *Rethink Sawyer* operator-assist robot. Many of these robots are designed to be "touch-sensitive", and will shut down their motors after operator contact. This contrasts with conventional robots, where this sensitivity is not used. Conventional robots often require monumental safety fences with complex light curtain systems to avoid operator contact. To achieve "touch-sensitivity", some are equipped with piezoelectric systems and others with torque-sensing mechanisms controlled by current draw.
         2.1.1.2. Overload Protection: A torque limiting mechanism can also be used as a precise overload protection device. Circuit breakers are generally used to prevent current from exceeding the limitations of electrical components, but a torque limiting mechanism can be used to control current more precisely and correlate it to a torque. One application for this is in the aerospace industry, where torque limiters are used to prevent jam damage in landing and air wing mechanisms. Another application is for preventive maintenance of machinery, where precise "normal" ranges can be defined and help detect failures before they occur.
         2.1.1.3. Torque Limiting: Many products in manufacturing require specific torques applied to fasteners. This can depend on the application of the product where material hardness, internal pressure, or vibration can play an effect on the required torque. For example, an excessive

torque applied to a plastic end plate can cause the threads to strip and permanently damage the part. Similarly, an insufficient torque applied on a gasket assembly may prevent the unit to seal properly. Automatically fastening products to a precise torque-limiting set-point is a large industry and this project will focus on this application.

2.1.2. Existing issues: There are several issues with existing torque limiting technologies in the manufacturing industry:

2.1.2.1. Mechanical Reliability: Many existing feedback systems use mechanical methods for limiting torque. This includes friction plates, magnetic clutches, and spring mechanisms. Although many of these methods are inexpensive, they aren't always precise and often require heavy maintenance. Magnetic methods also have slippage from the non-synchronous hysteresis.

2.1.2.2. Price: Electronic feedback systems are available but are expensive and have control systems that are bulky and inefficient. Many of these systems are still in R&D development and are far from optimized.

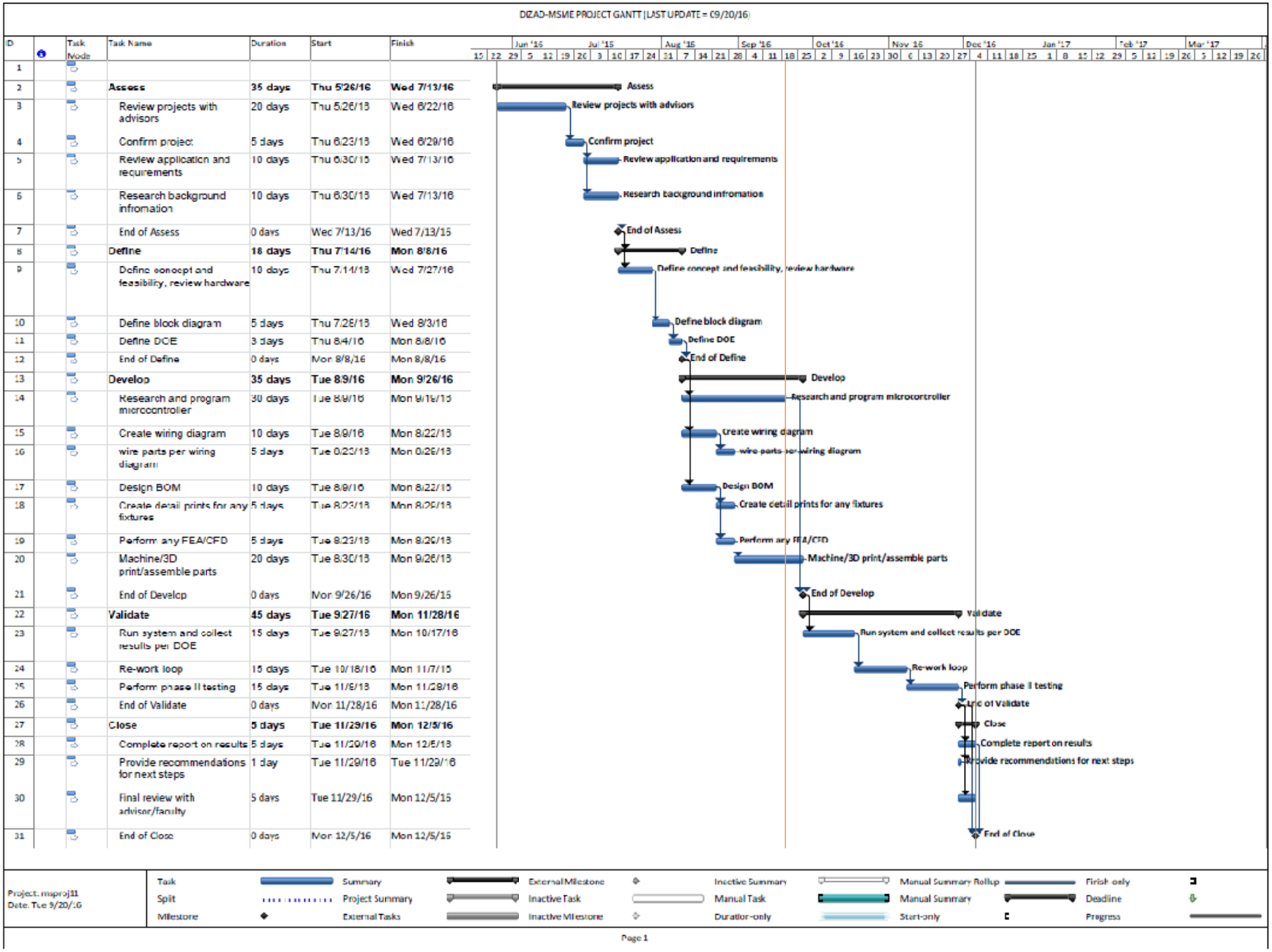
2.1.2.3. Motor Selection: Many existing nut runners use conventional *brushed* motors for actuation. This project investigates actuation using *brushless* motors instead. Brushless motors are more efficient, less noisy, have an increased lifespan, and are often smaller in size.

2.1.2.4. Configurability: Custom configurations can be difficult to find off-the-shelf. Many companies sometimes find it more feasible to design custom per the application. Some of the custom requirements can include screw locations, space restrictions, torque limits, and adjustability.

2.1.3. Conclusion: The objective of this project is to research and develop a working prototype for the groundwork of a torque limiting system that is precise, inexpensive, and easily configurable. This project applies primarily to the electronics manufacturing industry, where the target screw size will be a #4-40 screw given a torque limit of 3-7in-lbs.

2.2. Gantt Deliverable Schedule: An estimated Gantt schedule for the project was created as shown. This was used to review the critical path items, dependencies, slack time, and overall status of the project.

Figure 2.2.1.[1]: Project Gantt Chart:


DIZAD-MSME PROJECT GANTT (LAST UPDATE = 09/20/16)
ID
Task Mode
Task Name
Duration
Start
Finish
Jun '16
Jul '16
Aug '16
Sep '16
Oct '16
Nov '16
Dec '16
Jan '17
Feb '17
Mar '17
1
2 Assess 35 days Thu 5/26/16 Wed 7/13/16
3 Review projects with advisors 20 days Thu 5/26/16 Wed 6/22/16
4 Confirm project 5 days Thu 6/23/16 Wed 6/29/16
5 Review application and requirements 10 days Thu 6/30/16 Wed 7/13/16
6 Research background information 10 days Thu 6/30/16 Wed 7/13/16
7 End of Assess 0 days Wed 7/13/16 Wed 7/13/16
8 Define 18 days Thu 7/14/16 Mon 8/8/16
9 Define concept and feasibility, review hardware 10 days Thu 7/14/16 Wed 7/27/16
10 Define block diagram 5 days Thu 7/28/16 Wed 8/3/16
11 Define DOE 3 days Thu 8/4/16 Mon 8/8/16
12 End of Define 0 days Mon 8/8/16 Mon 8/8/16
13 Develop 35 days Tue 8/9/16 Mon 9/26/16
14 Research and program microcontroller 30 days Tue 8/9/16 Mon 9/19/16
15 Create wiring diagram 10 days Tue 8/9/16 Mon 8/22/16
16 wire parts per wiring diagram 5 days Tue 8/23/16 Mon 8/29/16
17 Design BOM 10 days Tue 8/9/16 Mon 8/22/16
18 Create detail prints for any fixtures 5 days Tue 8/23/16 Mon 8/29/16
19 Perform any FEA/CFD 5 days Tue 8/23/16 Mon 8/29/16
20 Machine/3D print/assemble parts 20 days Tue 8/30/16 Mon 9/26/16
21 End of Develop 0 days Mon 9/26/16 Mon 9/26/16
22 Validate 45 days Tue 9/27/16 Mon 11/28/16
23 Run system and collect results per DOE 15 days Tue 9/27/16 Mon 10/17/16
24 Re-work loop 15 days Tue 10/18/16 Mon 11/7/16
25 Perform phase II testing 15 days Tue 11/8/16 Mon 11/28/16
26 End of Validate 0 days Mon 11/28/16 Mon 11/28/16
27 Close 5 days Tue 11/29/16 Mon 12/5/16
28 Complete report on results 5 days Tue 11/29/16 Mon 12/5/16
29 Provide recommendations for next steps 1 day Tue 11/29/16 Tue 11/29/16
30 Final review with advisor/faculty 5 days Tue 11/29/16 Mon 12/5/16
31 End of Close 0 days Mon 12/5/16 Mon 12/5/16
Assess
Review projects with advisors
Confirm project
Review application and requirements
Research background information
End of Assess
Define
Define concept and feasibility, review hardware
Define block diagram
Define DOE
End of Define
Develop
Research and program microcontroller
Create wiring diagram
wire parts per wiring diagram
Design BOM
Create detail prints for any fixtures
Perform any FEA/CFD
Machine/3D print/assemble parts
End of Develop
Validate
Run system and collect results per DOE
Re-work loop
Perform phase II testing
End of Validate
Close
Complete report on results
Provide recommendations for next steps
End of Close
Project: mproj11
Date: Tue 9/20/16
Task
Split
Milestone
Summary
Project Summary
External Tasks
External Milestone
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
Deadline
Progress
Page 1

3. Define:

3.1. Key Definitions:

3.1.1. Shunt Resistor: A low value resistor that can be used to measure electric current.

3.1.2. Hall Effect: Principal that can measure electric current and deliver a proportional voltage.

3.1.3. SAW: (Surface Acoustic Wave) Interrogation that uses piezoelectric substrates to convert electrical to mechanical energy.

3.1.4. PSM: (Phase Shift Measurement) Technology used to measure angle of twist between 2 positions.

3.1.5. PWM: (Pulse-Width Modulation) Method of encoding a variable signal into a pulsed signal.

3.1.6. DAC: (Digital-to-Analog Converter) Method of converting a digital to analog signal.

3.1.7. Brushless Motor: Electrically commutated motor that does not use any brushes.

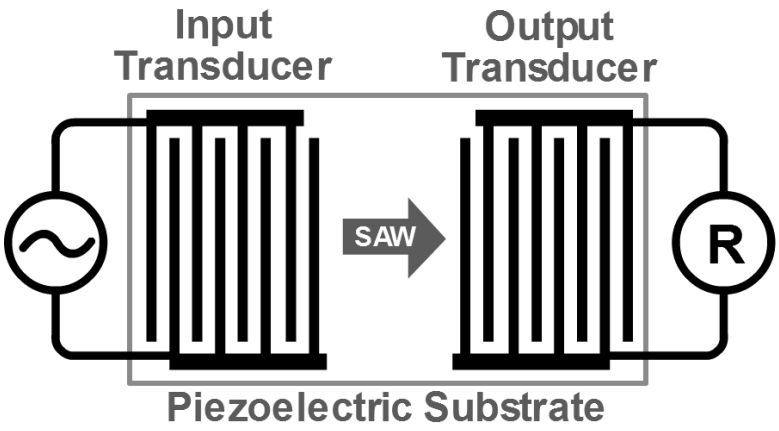
3.2. Torque Measurement:

3.2.1. Summary: Torque measuring is the method of sensing mechanical torque. Torque sensing is used on many rotating systems such as crankshafts, engines, gearboxes, rotors, or transmission boxes. Sensing technologies can vary significantly for static and dynamic torques, and can be much more complex for dynamic torques. Examples of torque sensing technologies include:

3.2.1.1. Strain gages: Using strain gauges in the shape of a *Wheatstone bridge* on the rotating shaft can be used to measure torque. The signal difference between the strain gages correlate to the torque applied on the shaft. One challenge with this technology is that it requires power and signal to be transferred to the anchor housing. To achieve this, contact-free methods can be used like rotary transformers and wireless telemetry. Contact methods are also used such as slip rings (similar to rotary distributors in pneumatics) but are more subject to frictional wear.

3.2.1.2. SAW: Another torque sensing technology is SAW (*Surface Acoustic Wave)* interrogation. It uses a piezoelectric substrate to convert electrical signals to mechanical signals (acoustic waves) and vice versa. As the shaft flexes, the strain of these SAW devices can be acquisitioned remotely and output a proportional calibrated torque value. This technology is a relatively recent development and is used in the automotive industry.

Figure 3.1.1.2.[1]: SAW diagram:



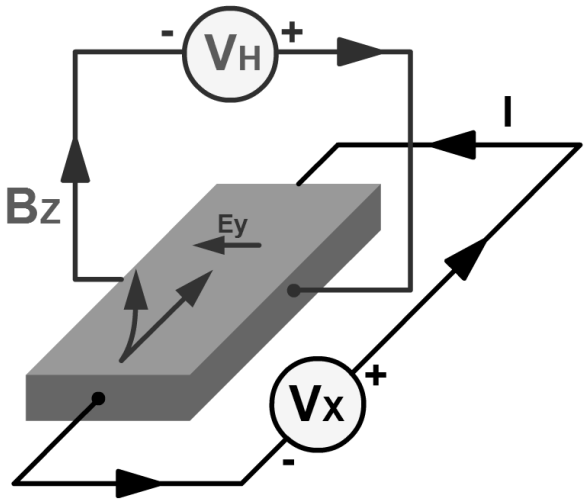
3.2.1.3. PSM: Finally, PSM (*Phase Shift Measurement)* can be used to measure torque. As the shaft undergoes torque stresses, the angle of twist can be measured by using 2 angular position sensors.

3.2.2. Conclusion: This project will experiment with a *Shunt* and *Hall Effect* sensor to determine the best method to measure the torque applied to the motor. The motor will have a weight on a pulley to simulate provided torque values.

3.2.2.1. Mechanical Torque: Provided torque values will be employed on the motor to simulate torque. A pulley with a provided radius will be attached to the motor shaft. The radius and weight will be multiplied to provide torque.

3.2.2.2. Current measurement: To measure the torque of the motor, a *Hall Effect* sensor will first be experimented with by connecting it to the power line of the motor driver. The *Hall Effect* principal uses electrical induction to output a proportional voltage to a variable current input. Electrons on a typical material will traverse in a straight line, but under a magnetic field, the electrons undergo the Lorentz force, where they begin to deviate proportional to its current. When they deviate, they produce a new voltage potential that can be used to correlate to current.

Figure 3.1.2.2[1]: Hall Effect diagram:



3.3. Motor Technology:

3.3.1. Summary: The principle behind DC brushed and brushless motors are essentially the same; a magnetic field is created that repels the force of permanent magnets as the motor windings are energized. There are several advantages and disadvantages when considering either for a design application:

3.3.1.1. Efficiency: Brushless motors are generally more efficient than brushed motors. In a brushless motor, the rotor full circle contacts the magnets and eliminates the performance inefficiencies of commutators, brushes, and transition gaps.

3.3.1.2. Control: Brushless motors have better positioning control accuracy. A brushless motor can be equipped with rotary encoders and *Hall Effect* sensors to control position. They are synchronous motors, meaning their rotors and stators magnetically turn at the same frequency. Speed can also be easily adjusted with brushless motors, where brushed motors are typically designed to run at fixed speed.

3.3.1.3. Price: Brushless motors are generally more expensive. Removing brushes makes them simpler for rotation, but there is added complexity on the control and gearbox systems.

3.3.1.4. Maintenance: Brushless motors require less maintenance. Over time, DC motor brushes require cleaning and replacement for continued operation. Brushless motors do not require any of these maintenance tasks and last a lot longer.

3.3.1.5. Heat dissipation: Brushless motors generate less heat, and dissipate heat better. They have less heat generation from rotor limitations, rotor inertia, and electromagnetic interference (EMI) generated from brush arching.

3.3.1.6. Torque Stability: Brushless motors are generally better at maintaining torque stability when running at various speeds. One of the reasons is that there is no power loss from brush transition.

3.3.1.7. Torque-to-size: Brushless motors (primarily from the Faulhaber brand) have a high torque-to-motor size ratio. They are fitted with gearboxes that can output torque values equivalent to brushed motors for a much smaller size.

3.3.1.8. Size: Brushless motors are smaller and more compact.

3.3.1.9. Noise: Brushless motors are less noisy.

3.3.2. Conclusion: A DC brushless motor will be selected for this application. Some of the brushless advantages include efficiency, torque stability, control, size, and maintenance improvements. Although the price of brushless motors are higher than brushed motors, the overall price of the torque system could still be less if proper lean manufacturing and product design principles are taken into account during the developmental phase.

3.4. Motor Driver:

3.4.1. Summary: A motor driver is used to control the performance of the DC brushless motor. For a simple power tool DC motor, the controller can be as simple as a switch. For complex positioning, controllers can be used to sequence multiple outputs. This automatic switching is often done automatically using NAND gates, flip flops, clock switches, and various resistors. Controllers can also have added complexity when they are designed for closed loop feedback positioning.

3.4.2. Controlling: Controllers can be operated manually or automatically via relay or contact. For a servo controller, the input can be manually adjusted using a potentiometer or automatically adjusted using a PWM or DAC signal.

3.5. Microcontroller:

3.5.1. Summary: A microcontroller is a miniature computer designed typically on a series of ICs(Integrated Circuits) of a PCB(Printed Circuit Board). It contains scaled down elements of a typical computer i.e. processing core, memory, I/O peripherals, and USB programmable ports. They are designed for embedded applications, where they can work independently of a central computer. Example applications of embedded microcontrollers can include power tools, alarm clocks, appliances, remote controls, and engine control systems. A PLC is a good midway example between a microcontroller and

desktop PC. Some of the key components of a microcontroller include the following:

3.5.1.1. CPU: Microcontrollers typically have a central IC for processing the embedded software. The software is compiled and assembled in such a way that it is compact and requires minimal memory. Programs can be permanent (firmware), read-only, or editable (flash field-alterable).

3.5.1.2. Digital Input/Outputs: Digital signals are discrete values with 2 binary values, zeroes and ones. Digital signals are typically not affected by system noise, making them ideal for simple discrete devices such as push buttons, LEDs, or relays.

3.5.1.3. Analog Input/Outputs: Analog signals are continuous signals that can deliver a wide range of variable values over time. Analog signals are subject to electronic noise, which degrades the signal-to-noise ratio (SNR). One way to resolve this is by using an Analog-to-Digital converter (ADC) where degradation can be significantly reduced. Analog signals are ideal for variable devices such as transducers, thermocouples, and LVDTs.

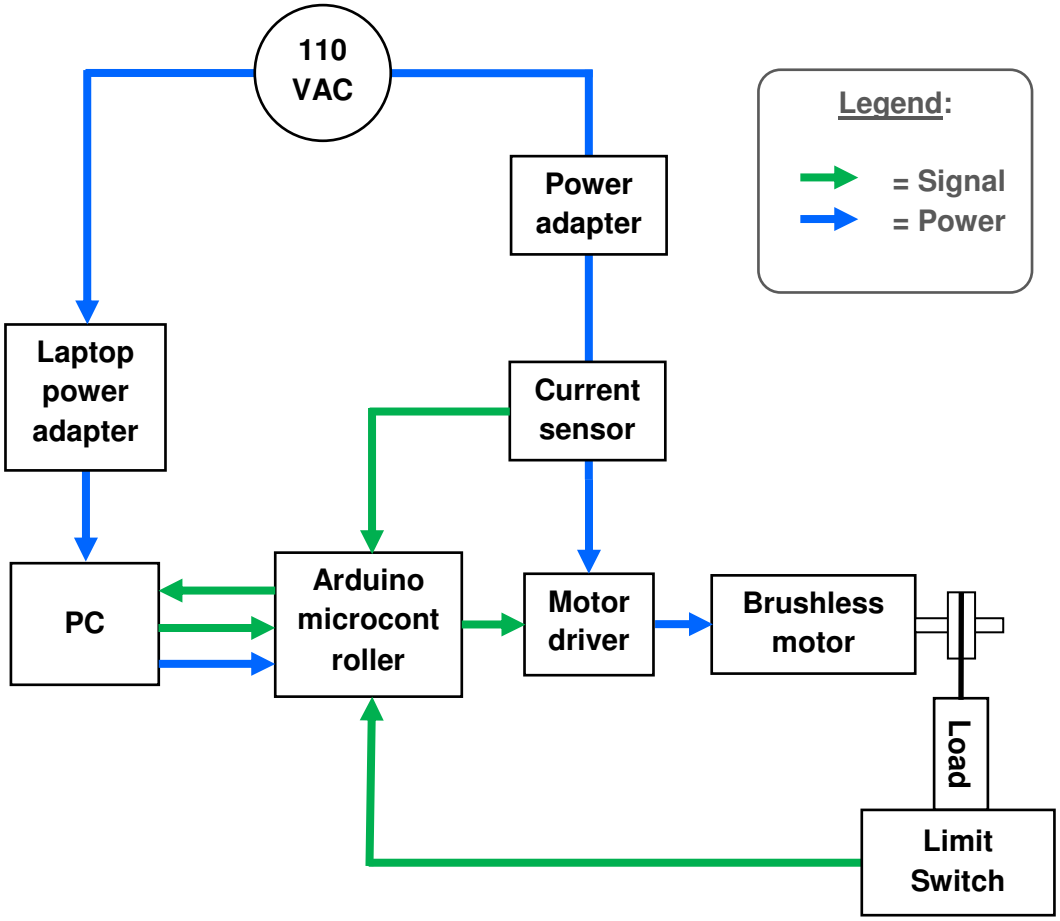
3.5.1.4. PWM(Pulse-Width Modulation) : PWM is used to simulate a variable output using a pulsed rectangular digital signal. It achieves this by varying the delays between pulses. One of the common uses of PWM is in photovoltaic solar battery chargers, servomechanisms, light dimming, and communication. A major advantage of PWM is that the power loss on switching is low.

3.5.2. Conclusion: A MEGA2560 Arduino microcontroller is selected for this project. This microcontroller has digital, analog, and PWM connectors that can be used to connect the system together. This microcontroller is universal and can be programmed with many software packages.

3.6. Block Diagram:

3.6.1. Diagram: Per the components discussed in the previous section, the suggested block diagram is presented below:

Figure 3.6.1[1]: System block diagram:



3.6.2. Sequence Summary: An IDE program(Arduino) is embedded inside the microcontroller for defining the serial I/Os and a C# interface program is used to control these serial I/Os. The interface program begins by sending motor speed and direction requirements to the microcontroller. The microcontroller then converts those values to digital/analog signals and communicates them to the motor driver. The driver splits those signals accordingly on 3 phases to turn the brushless motor. As the current increases, the electric current sensor sends a proportional voltage signal to the microcontroller, which is collected and converted to torque in the interface software. The measured torque is plotted and the selected torque limit is plotted against time. As torque is applied to the motor, the torque

reading will increase and switch to zero as soon as it reaches the torque limit.

4. Develop:

4.1. Summary: Prior to designing models or schematics, each sub-system was tested locally. The key sub-systems of this project included the microcontroller, programs, current sensor, brushless motor, and the torque fixture.

4.1.1. Microcontroller Development: Familiarizing with the microcontroller began by running variable, conditional statements, and loops on 2 digital output LEDs. The LEDs blinked, ran continuously, and ran conditionally per the input conditions of the program. Once this was achieved, the analog inputs were tested by connecting a potentiometer and observing for variable outputs this was achieved by generating a custom PWM.

Figure 4.1.1.1[1]: Initial setup:

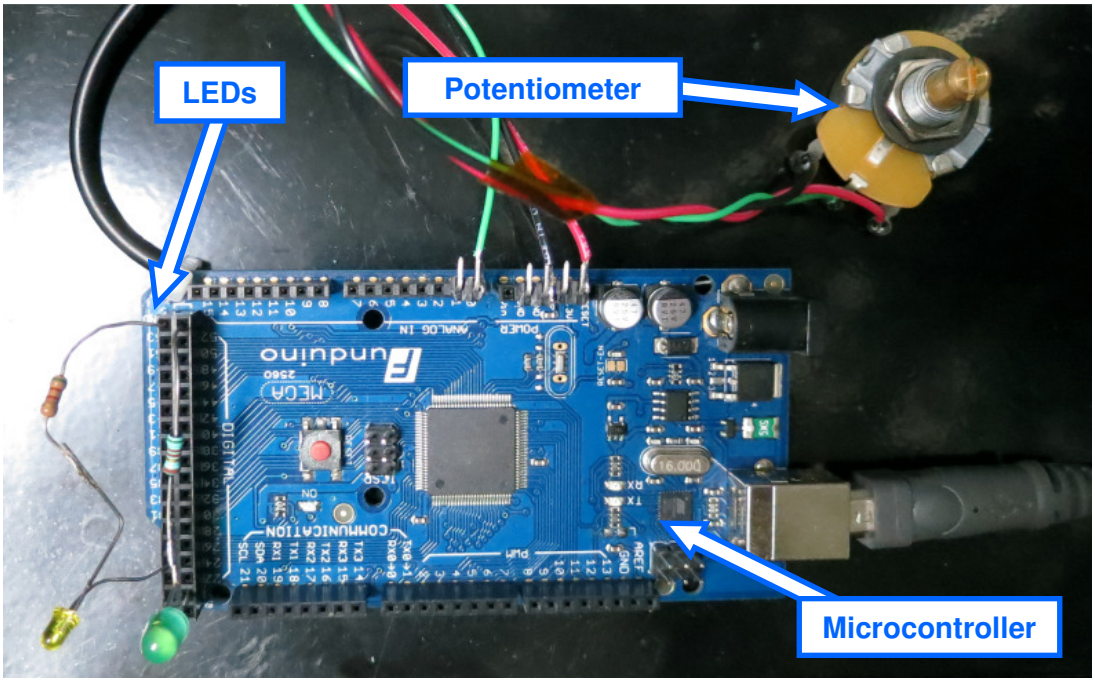


Figure 4.1.1[2]: Selected system microcontroller I/Os:

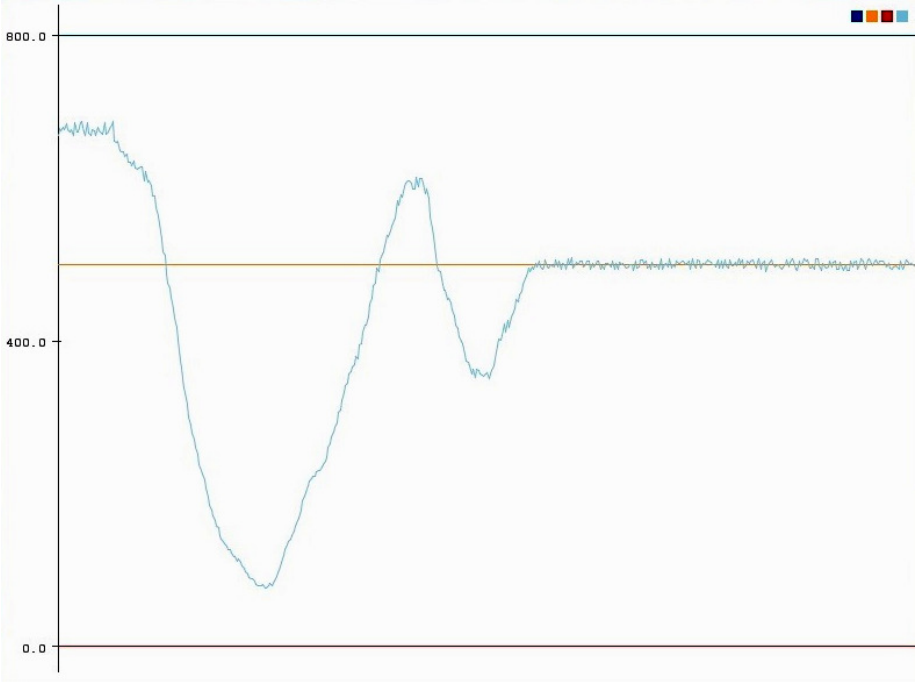
Microcontroller I/O Descriptions			
I/O	**Microcontroller Location**	**Description**	**Useage**
Output	D53	Driver signal	Instrumental
Input	D51	Reverse	Instrumental
Input	D49	Limit switch	Instrumental
Output	D33	LED#01	Testing
Output	D31	LED#02	Testing
Input	A08	Potentiometer Test	Testing
Input	A00	Current Sensor	Instrumental
Output	PW13	Motor Speed	Instrumental

4.1.2. Software Development:

4.1.2.1. Summary: The next step was to develop a method of capturing the analog input of the microcontroller so it can be used to calibrate with the torque. Capturing electric current can easily be performed with an oscilloscope but part of this project was to build a custom interface that can be used for reading live data and acquisitioning. This development would be beneficial for custom interfaces or equipment HMIs similar to the *Allen Bradley* PanelView series. To develop this software interface, several approaches were taken.

4.1.2.1.1. Arduino Plotter: The built-in Arduino plotter was first used for acquisitioning data. The response rate was excellent, but the plot was not highly customizable; labels, line types, and legends were not easy to control on this platform. After further research and testing, this method was not selected.

Figure 4.1.2.1.1[1]: Arduino plot of analog reading:



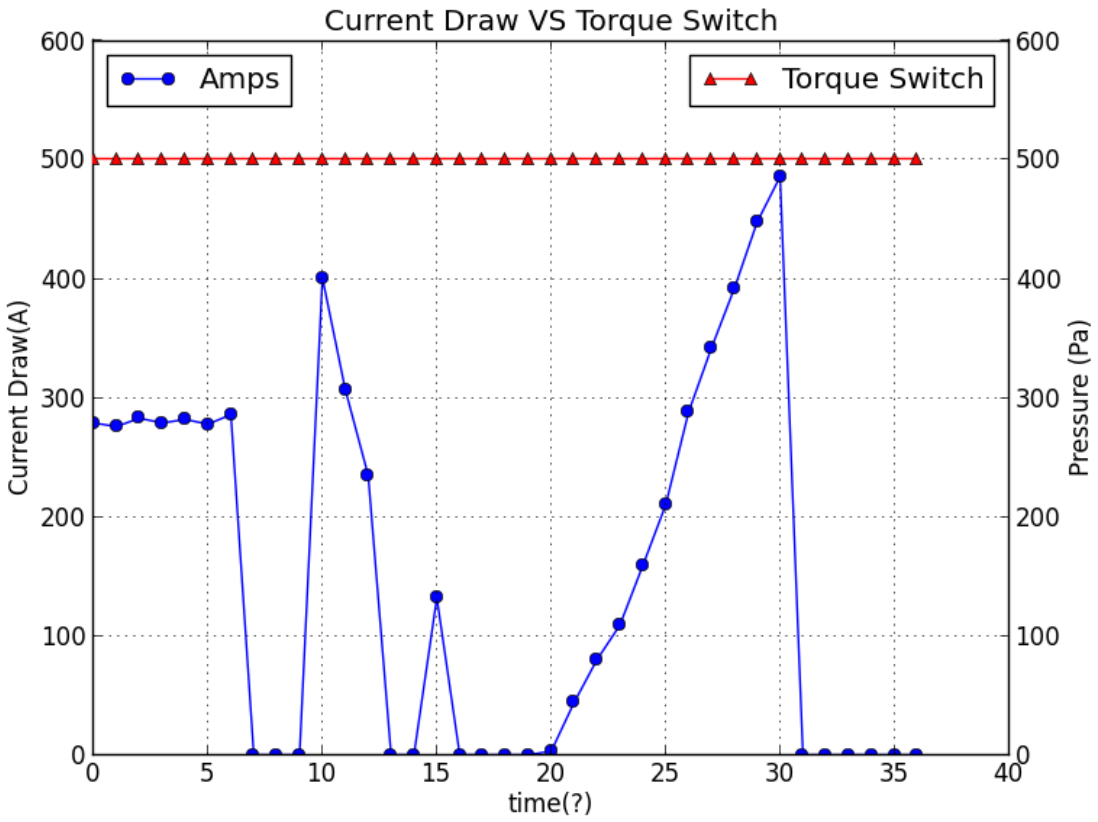
***Function Test Only:* Data not used for analysis**

Figure 4.1.2.1.1[2]: Arduino test code:

```
int sensorValue = 0;

void setup()
{
Serial.begin(9600);
}

void loop()
{
sensorValue = analogRead(A0);
analogWrite(9, sensorValue);

if (sensorValue > 500) // if the threshold is more than 500
  {sensorValue = 0;}  //shut down the motor

Serial.print(800); //set fixed upper limit
Serial.print(" "); //?
Serial.print(500); //set threshold limit
Serial.print(" "); //?
Serial.print(0); //set fixed lower limit
Serial.print(" "); //?
Serial.println(sensorValue);
delay(200);
}
```

4.1.2.1.2. Python MatPlotLive: A second approach was assessed using the Python *MatPlotLive* and *DrawNow* libraries. The language coded similar to Matlab and the plot was easy to modify. However, the response rate was very slow. After further research and testing, this method was not selected.

Figure 4.1.2.1.2[1]: Python plot of analog reading:



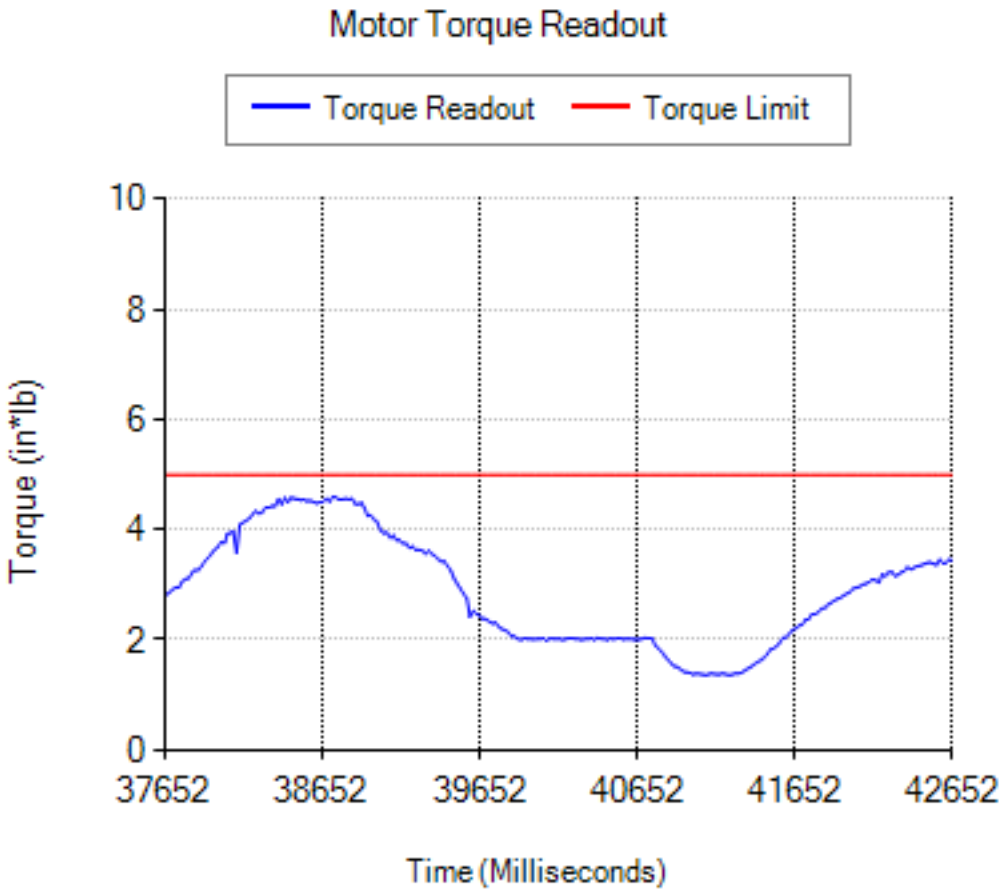
***Function Test Only:* Data not used for analysis**

Figure 4.1.2.1.2[2]: Python code:

```
import serial #import Serial Library
import matplotlib.pyplot as plt #import plotting
import numpy #import arrays
from drawnow import * #import live charts

Asignal = []
Aswitch = []

AData = serial.Serial('com3', 19200) #read from the pcserial, match the baude
plt.ion() #interactive mode for ploting live data
cnt = 0 # a counter used to only show live data

def fig(): #define a function, fig is a placeholder(ph)
    #series of plot commands
    plt.plot(Asignal, 'bo-', label = 'Amps')
    plt.ylim(0,600) #right y boundary
    plt.title('Current Draw VS Torque Switch')
    plt.grid(True)
    plt.ylabel('Current Draw(A)')
    plt.xlabel('time(?)')
    plt.legend(loc = 'upper left')
    plt2 = plt.twinx()
    plt.ylim(0,600) #left y boundary
    plt2.plot(Aswitch,'r^-', label = 'Torque Switch')
    plt2.legend(loc = 'upper right')
    plt2.ticklabel_format(useOffset = False)
    plt2.set_ylabel('Pressure (Pa)')

while True: #true is always true so always execute
    while (AData.inWaiting()==0): #is there data?
        pass #if there's no data, do nothing
    AString = AData.readline() #just a new variable to read the Adata

    #print AString #print the Adata or Astring

    array = AString.split(',') #splits about the "," inside the arduino code
    #upperBound = float(array[0]) #change the array value to a number
    limit = float(array[1])#change the array value to a number
    #lowerBound = float(array[2])#change the array value to a number
    signal = float(array[3])#change the array value to a number


    #print upperBound,",",limit,",",lowerBound,",",signal#change the array value to a number

    Asignal.append(signal)
    Aswitch.append(limit)

    #print Asignal
    print signal

    drawnow(fig, show_once = True)#draw the fig function, that show_once is crucial
    #plt.pause(0.000001)#a pause line that often follows the drawnow line

    cnt = cnt + 1 #counter is incremented
    if(cnt > 50): #if the counter is more than value
        Asignal.pop(0) #remove the initial elements of the pop graph
```

4.1.2.1.3. C# Plotter: Finally, C# via *Microsoft Visual Studio* was experimented with to plot the data. This approach delivered the best results. The response rate was high, the interface was easily customizable, and the software communicated well with the microcontroller.

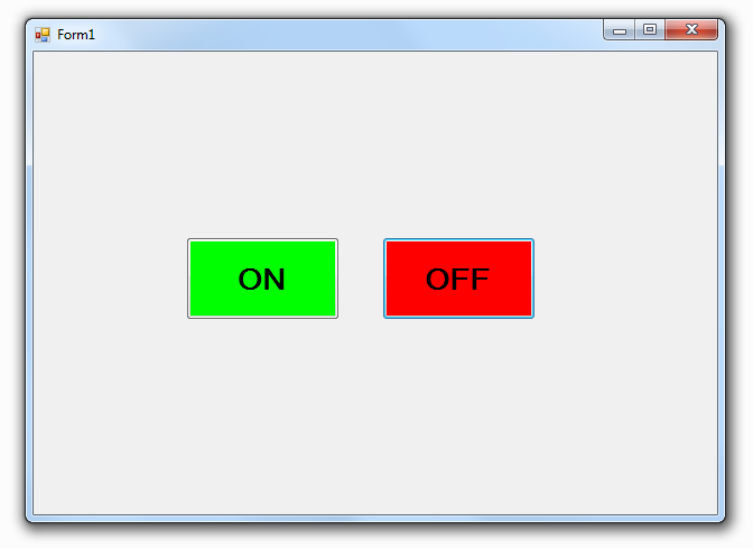
Figure 4.1.2.1.3[1]: C# plot of analog reading:



***Function Test Only:* Data not used for analysis**

4.1.2.1.4. C# Control: Finally, buttons were programmed in a C# form to experiment with microcontroller communication. The system was successfully able to turn ON/OFF LEDs by pressing buttons on the following interface.

Figure 4.1.2.1.4[1]: C# plot of analog reading:



Function Test Only: Data not used for analysis

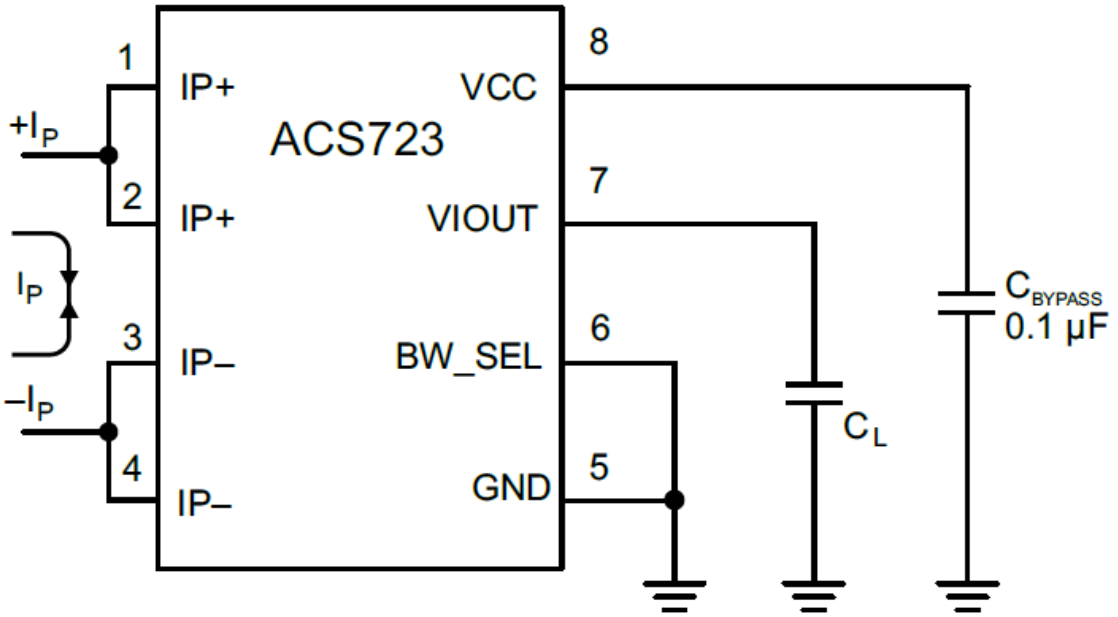
4.1.2.1.5. Other solutions: Matlab and processingJS (linking values to shapes and colors) were also considered but were not used for this project.

4.1.2.1.6. Conclusion: The Arduino program will be used to define the I/Os of the serial and the C# program will communicate with the Arduino program to read and write the appropriate I/Os to the microcontroller.

4.1.3. Torque Measurement Development:

4.1.3.1. Test#01-Hall Effect: A Hall Effect sensor was initially selected for accurately measuring the current draw of the motor. The Hall Effect method uses an electromagnet to output a voltage that is proportional to the current of the measuring line. This voltage was connected to an analog input of the Arduino microcontroller and measured.

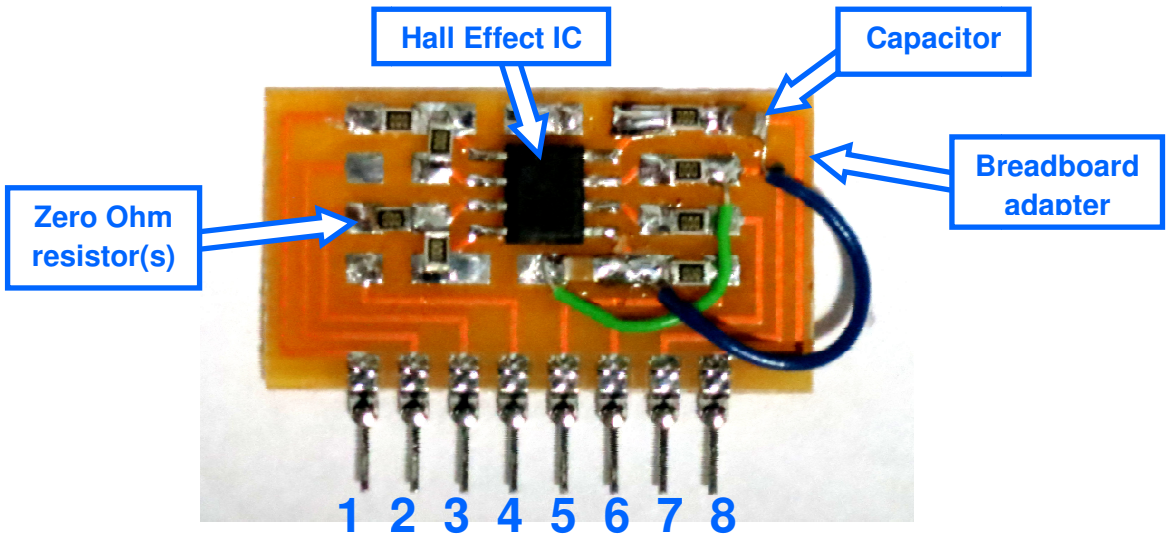
Figure 4.1.3.1[1]: Hall Effect sensor pin diagram (Allegro datasheet):



Typical Application

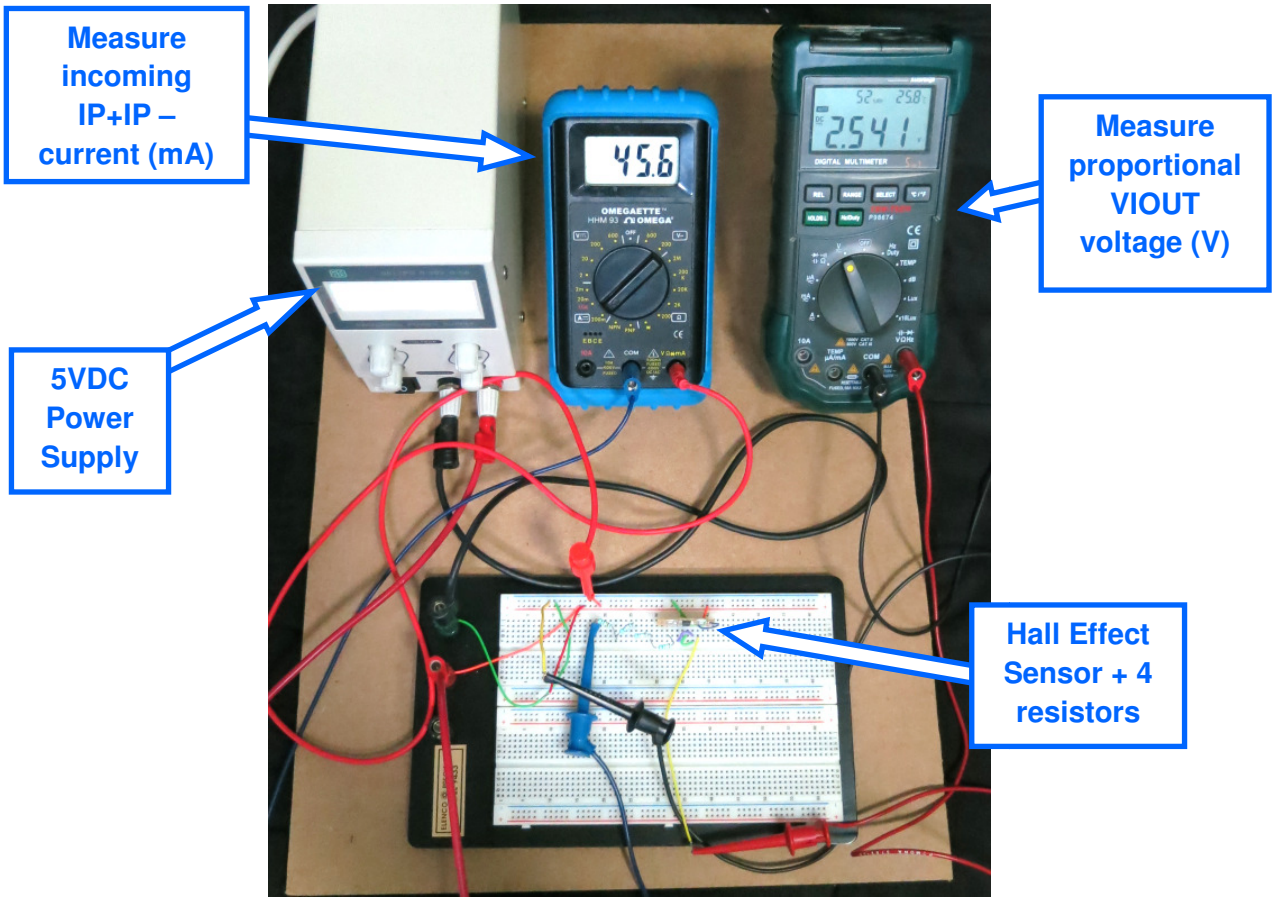
4.1.3.1.1. Adapter: To test this sensor, the ACS7231 Hall Effect/Amp transducer chip was purchased and soldered into a breadboard adapter as shown below in figure 4.1.3.1.1[1]. An adapter was soldered to the IC and zero-Ohm resistors were used to short connections. Per the datasheet, 2 capacitors were soldered to pins #7 & 8, one at 0.1µF and another at 0.15 µF.

Figure 4.1.3.1.1[1]: Actual sensor solder:



4.1.3.1.2. System: The board was connected to the solderless breadboard as shown in figure 4.1.4.1.1[2]. A 5VDC voltage was supplied to pin#1 only(since pin#2 is already shorted to pin#1), the return wire(pin#3) was connected to four 24.9Ω resistors in series, and the load(mA) was measured using a multi-meter. A 2nd multi-meter was used to measure the Voltage Output(VIOUT) or pin#7 i.e. the proportional voltage that we would use to measure the current draw of the brushless motor.

Figure 4.1.3.1.1[2]: Solderless Breadboard Hall Effect test rig:



The load was measured after eliminating one resistor at a time (simulating a gradual increase in load). The results are plotted in figure 4.1.3.2[2] with a highly linear proportional relationship between the current draw and the voltage output. This progression would need to be calibrated on the final rig but showed that the Hall Effect sensor is working.

4.1.3.1.3. Hall Effect Challenges: After the local tests were complete, the HE sensor was tested on the global motor system. The sensor

successfully delivered a proportional voltage to the current, but the change was less than 1%. This issue could have posed a problem when reading the signal on the microcontroller, especially since it can only deliver a resolution of 0-1023 bits. The available Hall Effect sensor seemed to be used for higher current applications so another method was researched to achieve higher sensitivity.

4.1.3.2. Test#02-Shunt Sensor: To achieve higher sensitivity, a shunt resistor was used as an alternative technology. A shunt sensor uses a low resistance path (to avoid disrupting the circuit) to proportionally measure the current flow. The voltage measured across the shunt can be scaled to display the value. Although Hall effect sensors have better circuit isolation, a shunt sensor is a better option for this application since the current measured is low and the sensitivity is high.

Figure 4.1.3.2[1]: Shunt sensor pin diagram (Zetex datasheet):

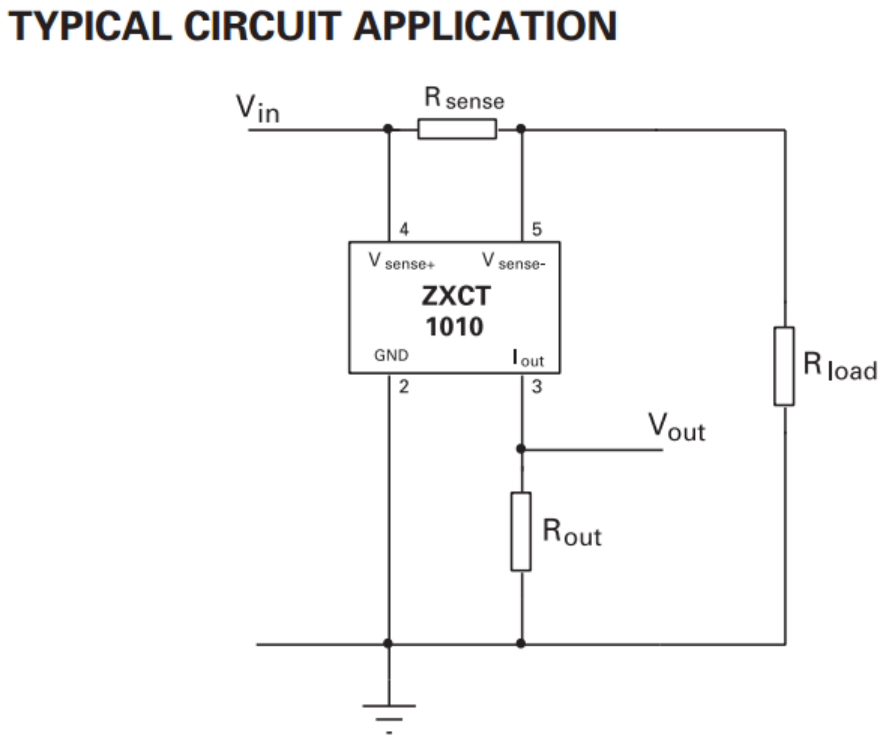
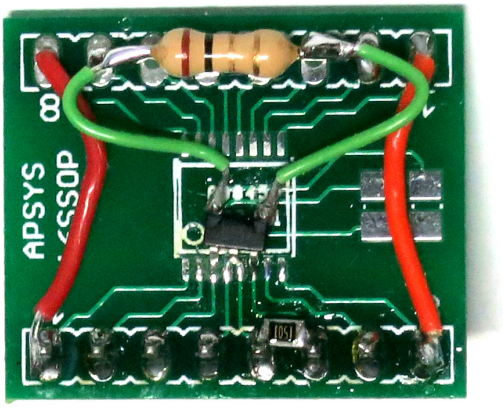


Figure 4.1.3.2[2]: Actual shunt sensor:

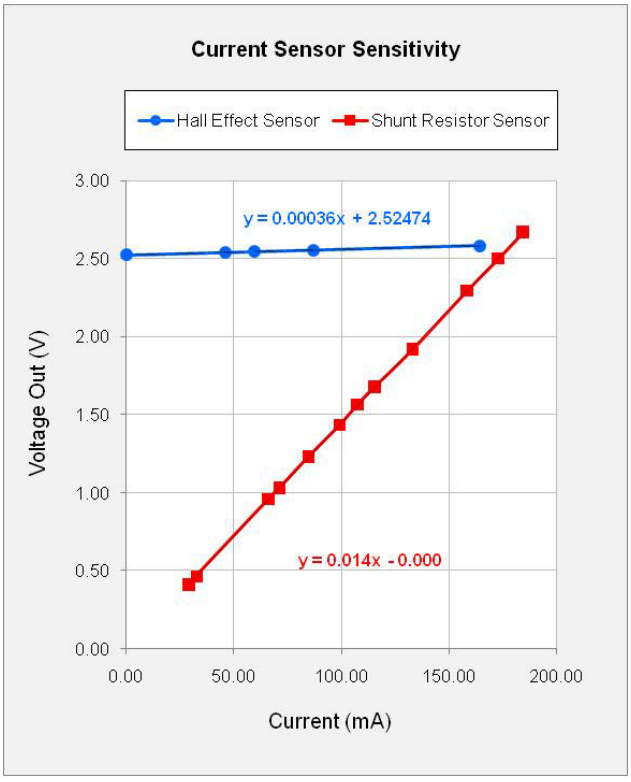


This shunt sensor was tested on the system and provided an electric current sensitivity of about 3%(an improvement to the <1% sensitivity from the HE sensor). This improved sensitivity helped improve setting torque limits on the program. The results were compared with the hall effect sensor and were graphed below in figure 4.1.3.2[2] showing a clear distinction in sensitivity.

Figure 4.1.3.2[2]: Hall Effect and shunt sensor results:

Hall Effect Sensor			
Resistance	Current	Vout	Vin
(Ω)	(mA)	(V)	(V)
∞	0.00	2.53	5.00
99.6	46.00	2.54	5.00
74.7	59.50	2.55	5.00
49.8	87.10	2.56	5.00
24.9	164.70	2.58	5.00

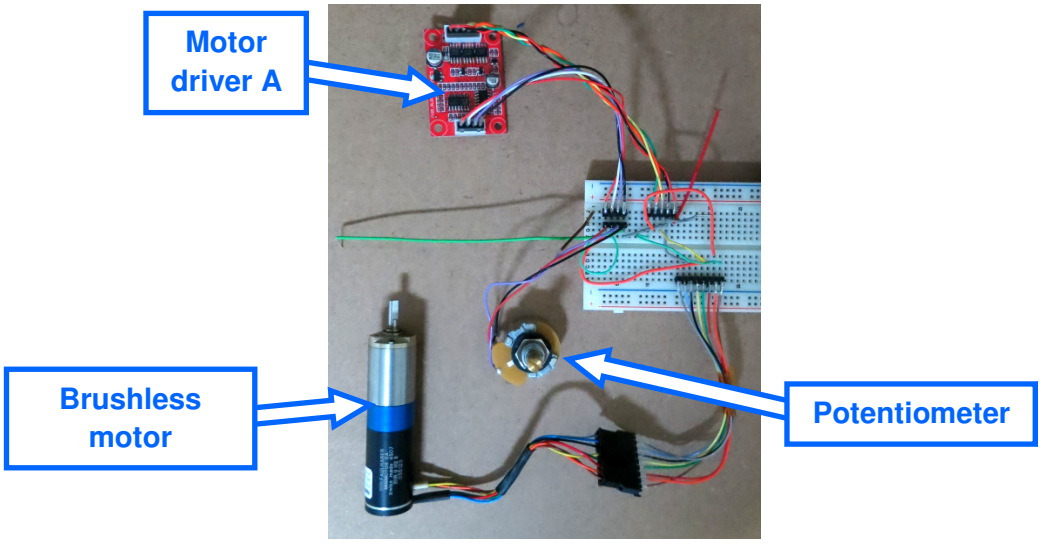
Shunt Resistor Sensor			
Resistance	Current	Vout	Vin
(Ω)	(mA)	(V)	(V)
N/A	32.60	0.47	5.00
N/A	28.80	0.41	5.00
N/A	66.00	0.96	5.00
N/A	71.20	1.03	5.00
N/A	84.60	1.23	5.00
N/A	99.20	1.43	5.00
N/A	107.40	1.57	5.00
N/A	115.50	1.68	5.00
N/A	133.20	1.92	5.00
N/A	158.40	2.29	5.00
N/A	173.00	2.50	5.00
N/A	184.60	2.67	5.00



4.1.4. Brushless Motor Development:

4.1.4.1. Test#01-Driver A: The first driver (Driver A) was tested in a local system. The brushless motor rotated successfully given an analog input but had trouble accepting a PWM signal. With a PWM, the motor would stall and could only rotate at minimal speeds.

Figure 4.1.4.1[1]: Driver A test setup:



4.1.4.2. Test#02-Driver B: Ideally, a DAC (Digital-to-Analog Converter) signal was preferred over a PWM. A DAC signal can vary the actual voltage from 0V to 5V as opposed to a PWM (unfiltered) signal that only varies the delay oscillation from a binary 0V or 5V signal. In order for this driver to work, it needed either a standard analog input or a DAC signal that varies from 0V-5V. An alternate driver (PRT#JYQD_V6.3) was used as shown below. This driver has a built-in DAC that allows it to take a PWM signal from the microcontroller. Driver B was tested and rotated the motor smoothly and at higher speed ranges to driver A.

Figure 4.1.4.2[1]: Driver comparison (old-red driver A, new-blue driver B):

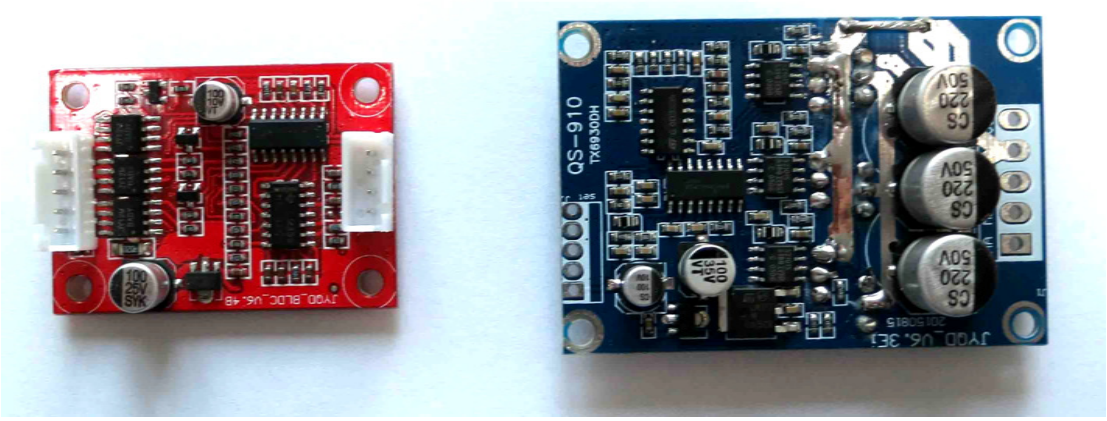
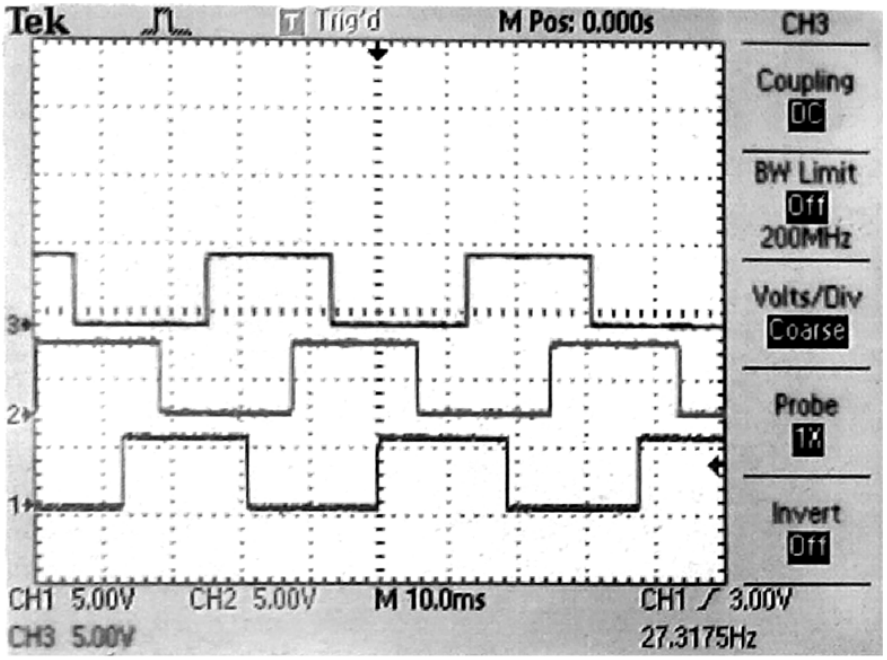
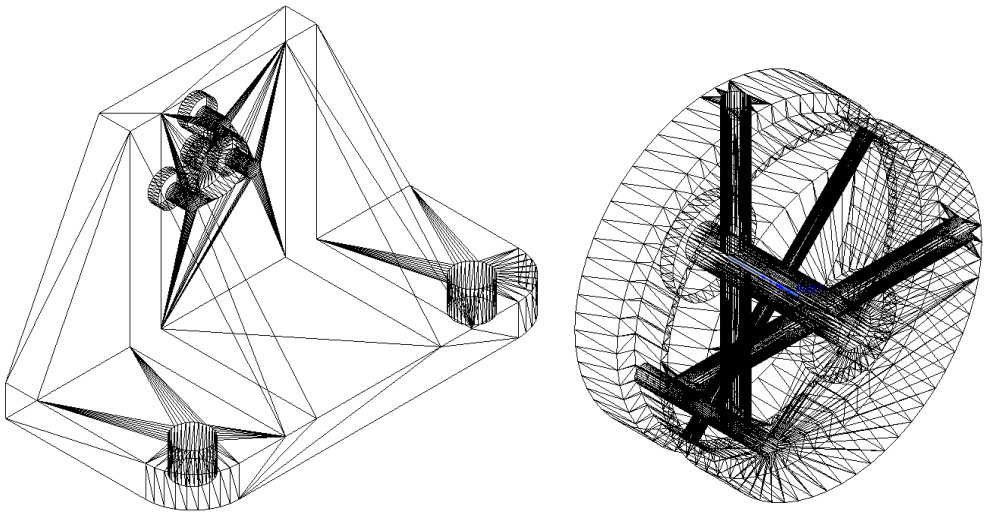


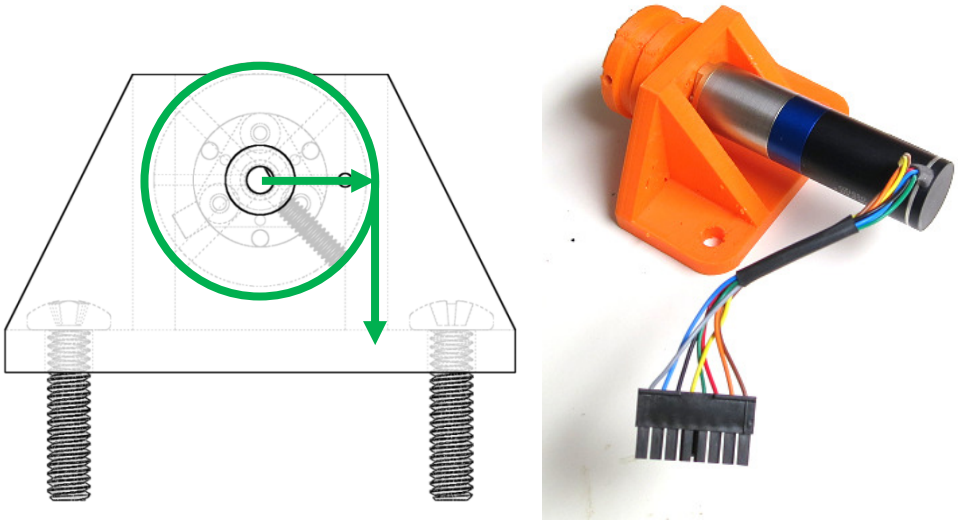
Figure 4.1.4.2[2]:: Oscilloscope readouts of driver phases A, B, and C out of motor leads, given a PWM input:



4.1.5. Torque Fixturing Development: Torque was generated using a weight and pulley mechanism. Specific torques were delivered provided a given radius and weight. The pulley and brackets to maintain this geometry was 3D printed and assembled.

Figure 4.1.5[1]: 3D printed torque fixture:





For future phases of this project were dynamic torque will be analyzed, the fixture will be coupled with a DC motor to simulate torque. The resistance on the DC motor will be varied to provide variable torque values.

4.1.6. **Conversions & Calibrations** :

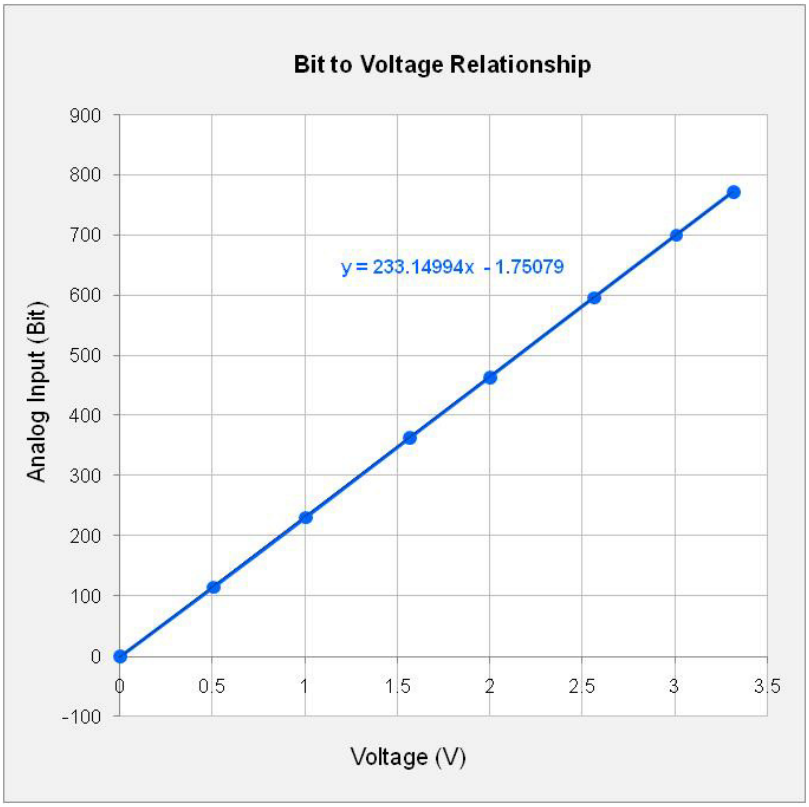
For properly displaying results on the interface, proper conversions and calibrations were required. This included bit conversions, torque conversions, and RPM conversions.

4.1.6.1. **Voltage-Bit Conversion**:

The bit conversion was for converting the Arduino analog 0 – 1023 bit signal to a voltage. This was essential for properly converting signals to electrical units. The Arduino datasheet suggested that 1 bit = 0.0049V but testing was done to confirm this. Voltages were adjusted from 0.0 – 3.3V and the bit was collected through the software. The linear regression is shown below in figure 4.1.7[1] and show that 1 bit is equivalent to 0.0075V. The software will use this regression line for converting voltage to bit.

Figure 4.1.6.1[1]: Voltage to bit relationship study:

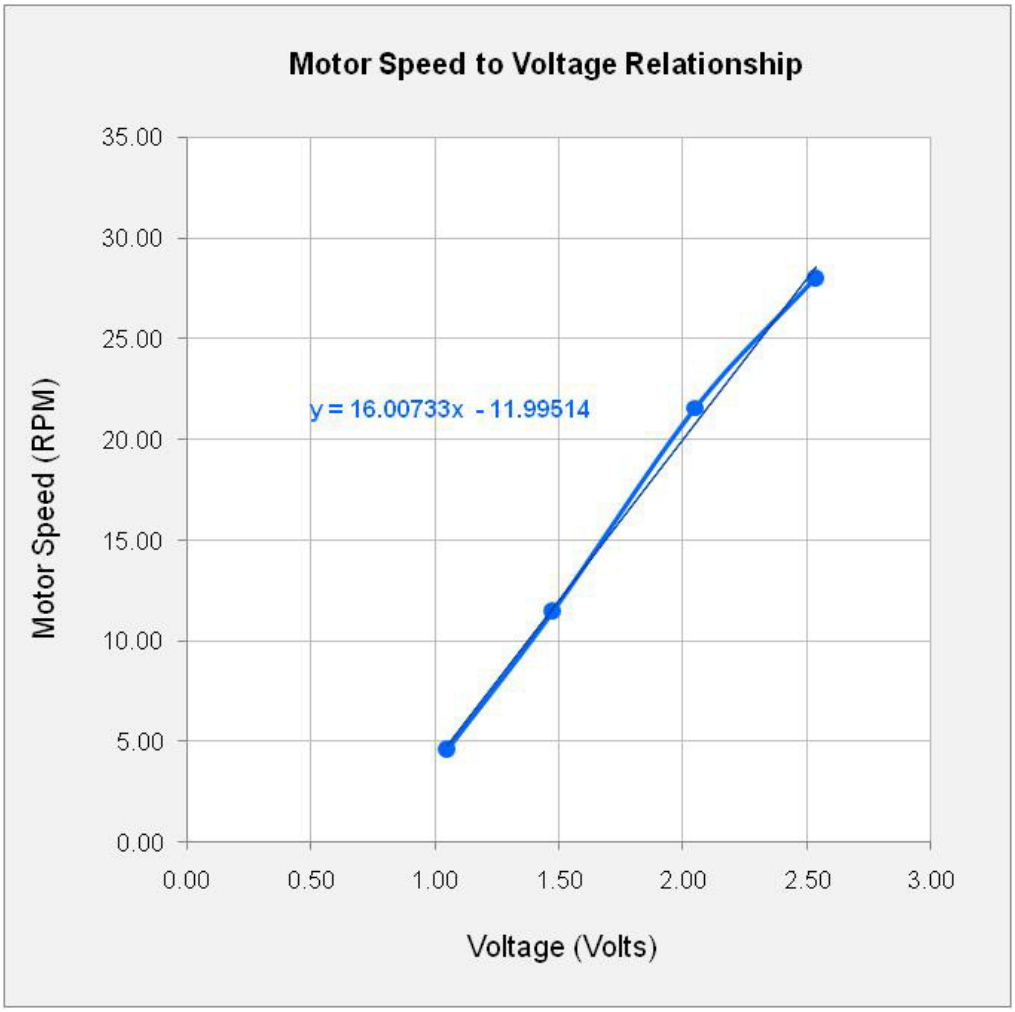
Voltage	Analog Input
(Volts)	(Bit)
3.318	772
3.008	700
2.562	596
1.999	464
1.566	363
1.004	231
0.504	115
0	0



4.1.6.2. **RPM-Bit Conversion**: The RPM conversion study was for understanding the voltage-to-RPM relationship of the brushless motor. Sets of 10 pulley rotations were timed and converted to RPM units while recording the input motor voltage.

Figure 4.1.6.2[1]: RPM-to-voltage study results:

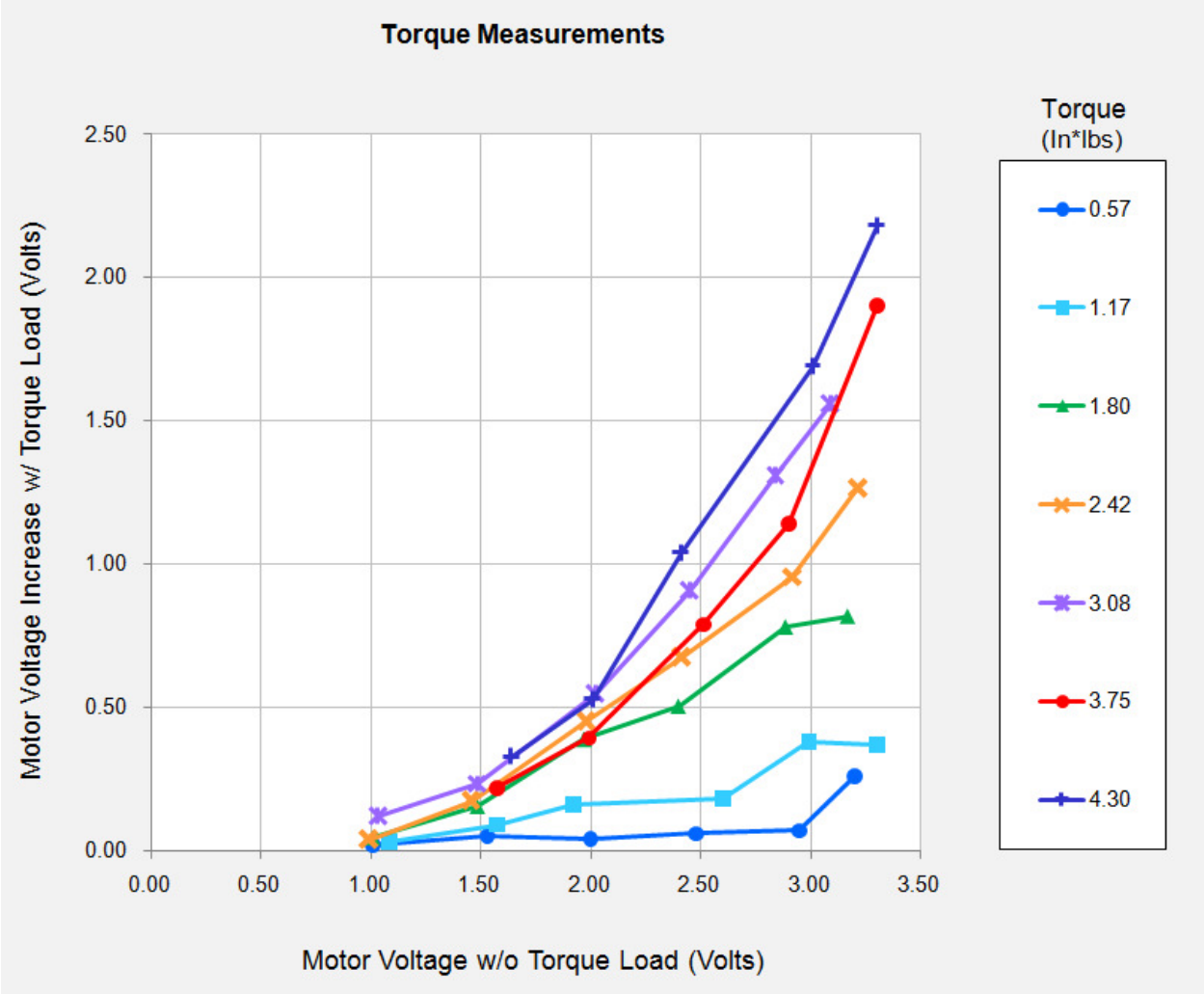
Motor Voltage	Duration for 10 cycles	Motor Speed	Motor Speed
(Volts)	(Seconds)	RPS (Revs/Sec)	RPM (Revs/Min)
1.05	130.00	0.08	4.62
1.48	52.17	0.19	11.50
2.05	27.79	0.36	21.59
2.54	21.37	0.47	28.08



4.1.6.3. **Torque-Bit Conversion**: Finally, a Torque-to-Bit conversion was required for displaying an accurate plot to the user. The process began by calculating the maximum allowable torque of the motor given the supplier datasheets. Next, the before(no torque) and after(with torque) voltages of the motor were measured provided a fixed input RPM of the motor. This experiment was repeated for 7 fixed torque values and the results are shown below.

Figure 4.1.6.3[1]: Torque-to-bit study results:

Measured	Calculated	Calculated	Calculated	Calculated	Measured	Measured
Torque Applied	**Motor Speed**	**Analog Signal (Without Load)**	**Analog Signal (With Load)**	**Analog Signal Difference**	**Motor Voltage (Without load)**	**Motor Voltage (With load)**
IN*LBS	**RPM**	**BIT**	**BIT**	**BIT**	**V**	**V**
0.57	4.17	233.73	238.39	4.66	**1.01**	**1.03**
	12.50	354.97	366.63	11.66	**1.53**	**1.58**
	20.02	464.55	473.88	9.33	**2.00**	**2.04**
	27.70	576.46	590.45	13.99	**2.48**	**2.54**
	35.23	686.04	702.36	16.32	**2.95**	**3.02**
	39.23	744.33	804.95	60.62	**3.2**	**3.46**
1.17	5.29	250.05	257.05	6.99	**1.08**	**1.11**
	13.14	364.29	385.28	20.98	**1.57**	**1.66**
	18.74	445.90	483.20	37.30	**1.92**	**2.08**
	29.62	604.44	646.41	41.97	**2.60**	**2.78**
	35.87	695.37	783.96	88.60	**2.99**	**3.37**
	40.83	767.64	853.91	86.27	**3.3**	**3.67**
1.80	4.17	233.73	243.99	10.26	**1.01**	**1.05**
	11.79	344.71	380.62	35.91	**1.49**	**1.64**
	19.56	457.79	548.48	90.70	**1.97**	**2.36**
	26.42	557.81	675.55	117.74	**2.40**	**2.91**
	34.22	671.35	853.91	182.56	**2.89**	**3.67**
	38.75	737.33	928.52	191.18	**3.17**	**3.99**
2.42	3.85	229.07	237.93	8.86	**0.990**	**1.03**
	11.33	337.95	378.75	40.80	**1.457**	**1.63**
	19.70	459.89	565.27	105.38	**1.980**	**2.43**
	26.58	560.14	717.28	157.14	**2.410**	**3.08**
	34.67	677.88	900.54	222.66	**2.915**	**3.87**
	39.45	747.59	1042.76	295.17	**3.214**	**4.48**
3.08	4.48	238.16	266.37	28.21	**1.03**	**1.15**
	11.73	343.78	398.33	54.56	**1.48**	**1.72**
	20.34	469.21	597.44	128.23	**2.02**	**2.57**
	27.22	569.47	781.63	212.17	**2.45**	**3.36**
	33.47	660.40	965.82	305.43	**2.84**	**4.15**
	37.47	718.68	1082.40	363.71	**3.09**	**4.65**
3.75	13.14	364.29	415.59	51.29	**1.57**	**1.79**
	19.86	462.22	554.31	92.09	**1.99**	**2.39**
	28.18	583.46	767.64	184.19	**2.51**	**3.30**
	34.43	674.38	940.17	265.79	**2.90**	**4.04**
	40.83	767.64	1210.63	442.98	**3.30**	**5.20**
4.30	14.26	380.62	457.55	76.94	**1.64**	**1.97**
	20.18	466.88	590.45	123.57	**2.01**	**2.54**
	26.58	560.14	802.62	242.48	**2.41**	**3.45**
	36.19	700.03	1094.05	394.02	**3.01**	**4.70**
	40.83	767.64	1275.91	508.27	**3.30**	**5.48**



4.1.7. Final Conversion map:

*RPM = 0.06866*BIT - 11.875*
*VOLT = 0.00429*BIT + 0.00751*
TORQUE =

Torque	Slope Factor	Offset Factor
IN*LB	N/A	N/A
1	0.000	0.000
	0.088	-25.402
	0.069	-22.154
	0.050	-18.965
2	0.000	0.000
	0.080	-24.011
	0.064	-20.185
	0.048	-16.558

Torque	Slope Factor	Offset Factor
IN*LB	N/A	N/A
3	0.000	0.000
	0.072	-22.612
	0.059	-18.205
	0.045	-14.137
4	0.000	0.000
	0.064	-21.352
	0.054	-16.474
	0.043	-11.720

Torque	Slope Factor	Offset Factor
IN*LB	N/A	N/A
5	0.000	0.000
	0.060	-19.960
	0.051	-14.490
	0.042	-9.305

Figure 4.1.7[1]: System Conversion Map

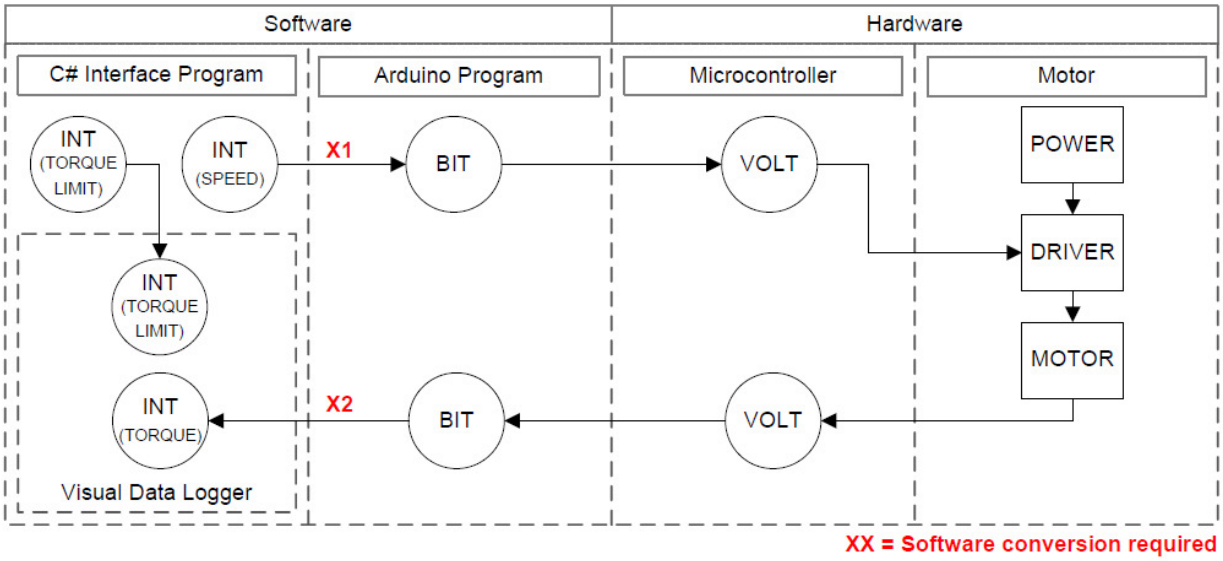


Figure 4.1.7[2]: Datasheet Equations

	Electrical	Mechanical
Torque	Km * I	= M + Cx
Voltage	Un - I * R	= n * (Ke/1000)
Combined Equation	M= (Km / R) *(U-[n * Ke/1000]) - Cx	

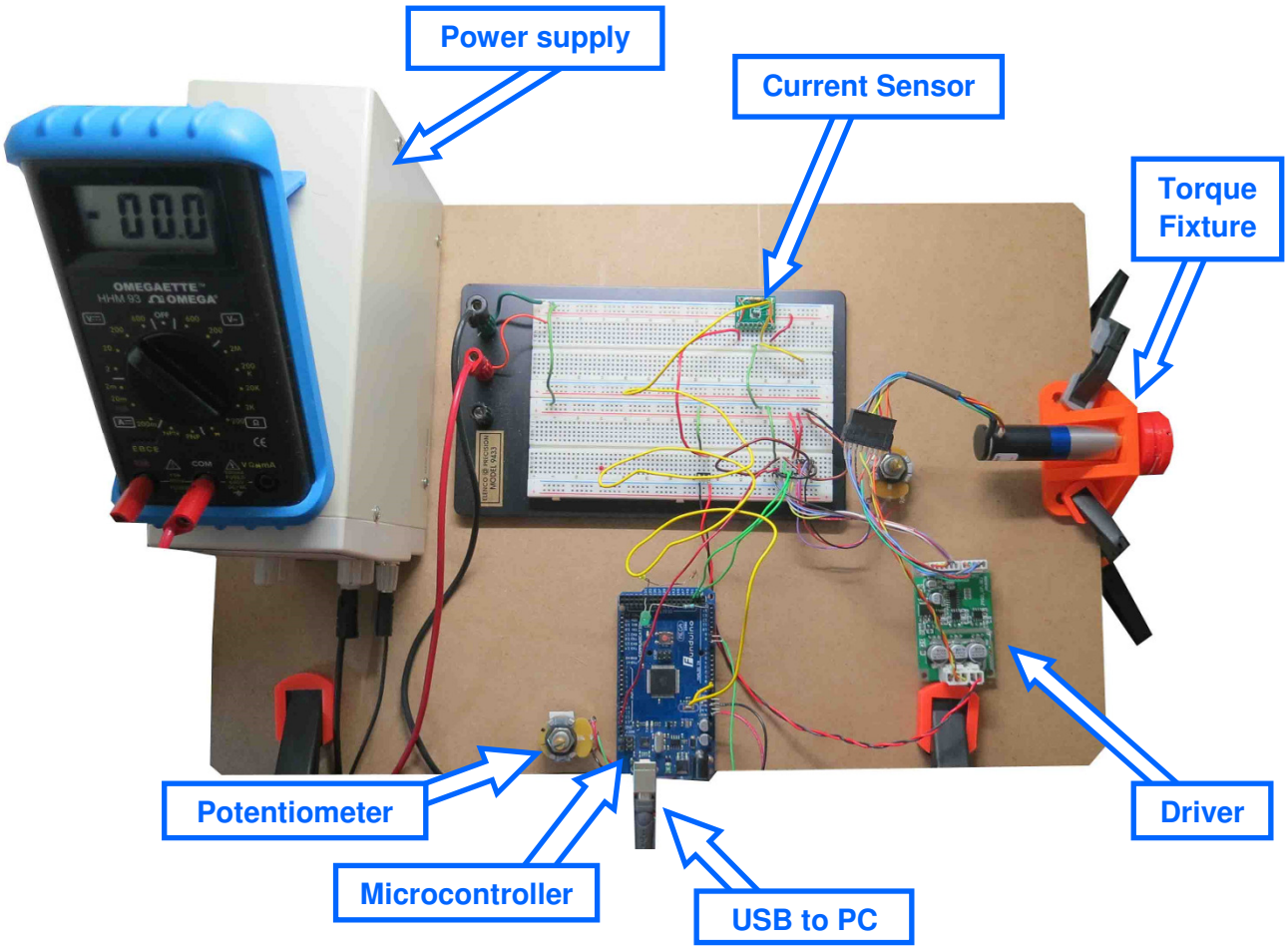
Symbol	Description
M	Torque
Km	Torque constant
R	Resistance
U	Voltage
n	speed
Ke	back EMF
Cx	Friction torque

4.2. Final Setup:

4.2.1. Final System:

The final system is pictured below.

Figure 4.2.1[1]: Final test rig



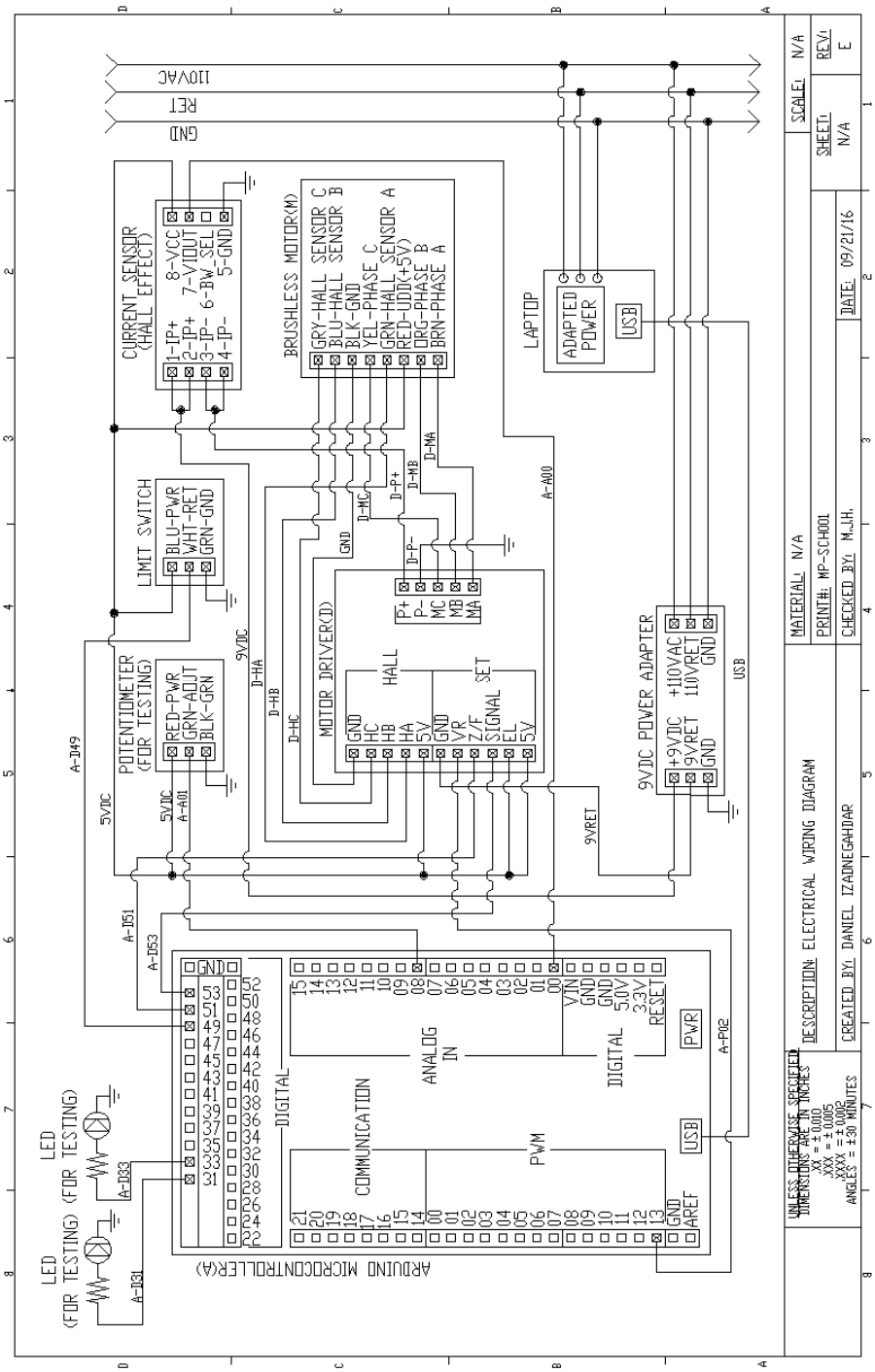
4.2.2. Final Bill of Materials(BOM):

PRT#	Description	Supplier	Mfg.	Qty
MEGA2560	Arduino board	Arduino	Arduino	1
2036V0009	Motor(Brushless)	N/A	Faulhaber	1
042726	Motor gearbox	N/A	Faulhaber	1
JYQD_V7.3D1	Motor driver	Amazon	N/A	1
46SS0P	Electric current sensor	DigiKey	Allegro	1
N/A	3D Bracket	RP+M	RP+M	2

4.2.3. Final Wiring Diagram:

4.2.3.1. A wiring diagram was finalized after the block diagram, datasheets, and pin diagrams were reviewed and tested. This diagram underwent 5 revisions and the latest revision is shown (REV-E).

Figure 4.2.3.1[1]: Wiring diagram:



4.2.4. Final Programs:

4.2.4.1. Arduino Embedded Program:

```
//Variables
  int reverse_OUT = 53; //Reverse motor output
  int potentiometer_IN = A8; //Test input
  int motor_OUT = 13; //Motor speed output to PWM13
  int sensor_IN = A0; //Electric current input

//Setup conditions
  void setup()
    {
       Serial.begin(9600); //Set the baude to value
       pinMode(reverse_OUT,OUTPUT); //Define the output pin
    }

//Loop conditions
  void loop()
    { //Read
       //Read the analog input of the potentiometer(for testing purposes only)
            Serial.println(analogRead(potentiometer_IN)); //Print the value of the potentiometer
            delay(20); //Send analog value to serial port every X/1000 second

      /* //Read the analog input of the sensor i.e. torque of motor
            Serial.println(analogRead(sensor_IN)); //Print the value of the sensor
            delay(20); //Send analog value to serial port every X/1000 second */

      //Write
        //Write microcontroller conditions per the C# command cases
          char data = Serial.read();
          switch(data)
              {
                //Direction write
                  case '0' : digitalWrite(reverse_OUT,LOW);break; //Change to the forward direction of the motor
                  case '1' : digitalWrite(reverse_OUT,HIGH);break; //Change to the reverse direction of the motor

                //Motor speed write
                  case '2' : analogWrite(motor_OUT,map(0, 0, 1023, 0, 255));break; // Change to SPEED 0, map to 0-255
                  case '3' : analogWrite(motor_OUT,map(319, 0, 1023, 0, 255));break; // Change to SPEED 1, map to 0-255
                  case '4' : analogWrite(motor_OUT,map(464, 0, 1023, 0, 255));break; // Change to SPEED 2, map to 0-255
                  case '5' : analogWrite(motor_OUT,map(610, 0, 1023, 0, 255));break; // Change to SPEED 3, map to 0-255
            }
    }
```

4.2.4.2. C# Interface Program:

```
1  //System namespaces
2      using System;
3      using System.Collections.Generic;
4      using System.ComponentModel;
5      using System.Data;
6      using System.Drawing;
7      using System.Linq;
8      using System.Text;
9      using System.Threading.Tasks;
10     using System.Windows.Forms;
11     using System.IO.Ports; // Added for reading arduino I/Os
12
13 //MAIN program
14     namespace MOTOR INTERFACE //Interface program
15     {//MAIN start bracket
16     //FORM program
17     public partial class Motor_Interface_Form : Form
18     {//FORM start bracket
19     //Initiation
20         private SerialPort myPort; //Added for serial port review
21         private DateTime datetime; //Added for live analog read
22         private string data_IN; //Added for specifying the incoming analog data
23         private DateTime startTime; //Added for realtime chart
24         public Motor Interface Form()
25
26         {
27             InitializeComponent();
28             GetAvailablePorts(); //Function to review the ports
29             ///Enable/Disable buttons
30                 But_Connect.Enabled = true;
31                 But_Close.Enabled = false;
32                 But_Forward.Enabled = false;
33                 But_Reverse.Enabled = false;
34                 But_Start.Enabled = false;
35                 But_Stop.Enabled = false;
36                 But_CalTorque.Enabled = false;
37                 But_Start_Data.Enabled = false;
38                 But_Stop_Data.Enabled = false;
39                 But_Save_Data.Enabled = false;
40             I_Motor.Enabled = false; //Stop the motor GIF
41             startTime = DateTime.Now; //Added for realtime chart
42         }
43
44     //Review port availability
45         void GetAvailablePorts()
46         {
47             String[] ports = SerialPort.GetPortNames();
48             SBox_Ports.Items.AddRange(ports);
49         }
50
51     //Receiving data
52     private void Myport_DataReceived(object sender, SerialDataReceivedEventArgs e)
53             {
54                 try
55                 {
56                     data_IN = myPort.ReadLine(); //Read analog signal from port and write to data_IN
57                     this.Invoke(new EventHandler(displaydata_event));
58                 }
59                 catch (Exception) { MessageBox.Show("Error"); }
60             }
61
62     //General and graphical interface commands
63         private void displaydata_event(object sender, EventArgs e)
64         {
```

```
65              //General items
66                  if (TBar_RPM.Value == 0) { I_Motor.Enabled = false; } //Turn off the motor GIF
67
68              //Display data on the single readout boxes
69                  double torque_value = (Convert.ToDouble(EBox_CalTorque_S.Text)) * Convert.ToDouble(data_IN)
70                  + (Convert.ToDouble(EBox_CalTorque_O.Text)); //Convert bit to torque
71                  RBox_Bit.Text = data_IN;    //Display the bits on the readout box
72                  RBox_Torque.Text = Convert.ToString(torque_value);
73
74              //Display data on the central readout box
75                  datetime = DateTime.Now; //set the datetime variable to the current time
76                  string time = datetime.Hour + ":" + datetime.Minute + ":" + datetime.Second; //set time
77                  RBox_Data.AppendText(time + "\t\t" + data_IN + "\n"); //Display data and time on readout box
78
79              //Display the data on the meter
80                  int integer_data_IN = Convert.ToInt32(data_IN); //Convert the data_IN to an integer
81                  if (integer_data_IN > PBar_Torque.Maximum)
82                      { PBar_Torque.Value = PBar_Torque.Maximum; } //Set max readout limit on progress bar
83                  else if (integer_data_IN < PBar_Torque.Minimum)
84                      { PBar_Torque.Value = PBar_Torque.Minimum; } //Set min readout limit on progress bar
85                  else { PBar_Torque.Value = integer_data_IN; } //Connect torque progress bar to torque readout
86
87              //Chart code
88                  TimeSpan sinceStart = DateTime.Now - startTime; //Define var between current and start time
89
90              //Chart graphics
91                  this.Chart_Torque.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.Silver; //Grid line color
92                  this.Chart_Torque.ChartAreas[0].AxisX.MajorGrid.LineDashStyle =
                      System.Windows.Forms.DataVisualization.Charting.ChartDashStyle.Dot; //Grid line style
93                  this.Chart_Torque.ChartAreas[0].AxisY.MajorGrid.LineDashStyle =
                      System.Windows.Forms.DataVisualization.Charting.ChartDashStyle.Dot; //Grid line style
94                  double Chart_XValue = sinceStart.TotalMilliseconds; // datetime.Millisecond
95
96              //Chart XY visual limits
97                  this.Chart_Torque.ChartAreas[0].AxisY.Minimum = 0; //set the Y axis min values
98                  this.Chart_Torque.ChartAreas[0].AxisY.Maximum = 6; //set the Y axis max values
99                  this.Chart_Torque.ChartAreas[0].AxisX.Minimum = Math.Round(sinceStart.TotalMilliseconds
                      5000); //set the X axis min values
100                 this.Chart_Torque.ChartAreas[0].AxisX.Maximum = Math.Round(sinceStart.TotalMilliseconds); //
                      set the X axis max values
101                 string test = Convert.ToString(torque_value); //double works fine
102                 this.Chart_Torque.Series["Torque Readout"].Points.AddXY(Chart_XValue, test); //Set plot
103                 this.Chart_Torque.Series["Torque Readout"].Color = Color.Green; //Change the color of the line
104
105             //Torque Limit Series
106                 double C_Limit = Convert.ToDouble(EBox_Torque_Limit.Text); //Redundant callout
107                 this.Chart_Torque.Series["Torque Limit"].Points.AddXY(Chart_XValue, C_Limit); //Set plot
108                 this.Chart_Torque.Series["Torque Limit"].Color = Color.Orange; //change the color of the line
109
110             //Shut down motor if the torque limit is reached*****
111                 if (TBar_Torque_Limit.Value < Convert.ToInt32(torque_value))
112                     { But_Stop.PerformClick();}//Turn off motor when torque limit is reached
113         }
114
115     //Setting default Calibration values
116         private void SetDefaultCalibration()
117         {
118         //change the cal values:
119             //0 RPM
120                 if (TBar_RPM.Value == 0 && TBar_Torque_Limit.Value == 1)
121                     { EBox_CalTorque_S.Text = "0.000"; EBox_CalTorque_O.Text = "0.000";}
122                 if (TBar_RPM.Value == 0 && TBar_Torque_Limit.Value == 2)
123                     { EBox_CalTorque_S.Text = "0.000"; EBox_CalTorque_O.Text = "0.000";}
124                 if (TBar_RPM.Value == 0 && TBar_Torque_Limit.Value == 3)
```

```
125                     { EBox_CalTorque_S.Text = "0.000"; EBox_CalTorque_O.Text = "0.000";}
126                 if (TBar_RPM.Value == 0 && TBar_Torque_Limit.Value == 4)
127                     { EBox_CalTorque_S.Text = "0.000"; EBox_CalTorque_O.Text = "0.000";}
128                 if (TBar_RPM.Value == 0 && TBar_Torque_Limit.Value == 5)
129                     { EBox_CalTorque_S.Text = "0.000"; EBox_CalTorque_O.Text = "0.000";}
130             //10 RPM
131                 if (TBar_RPM.Value == 1 && TBar_Torque_Limit.Value == 1)
132                     { EBox_CalTorque_S.Text = "0.088"; EBox_CalTorque_O.Text = "-25.402";}
133                 if (TBar_RPM.Value == 1 && TBar_Torque_Limit.Value == 2)
134                     { EBox_CalTorque_S.Text = "0.080"; EBox_CalTorque_O.Text = "-24.011";}
135                 if (TBar_RPM.Value == 1 && TBar_Torque_Limit.Value == 3)
136                     { EBox_CalTorque_S.Text = "0.072"; EBox_CalTorque_O.Text = "-22.612";}
137                 if (TBar_RPM.Value == 1 && TBar_Torque_Limit.Value == 4)
138                     { EBox_CalTorque_S.Text = "0.064"; EBox_CalTorque_O.Text = "-21.352";}
139                 if (TBar_RPM.Value == 1 && TBar_Torque_Limit.Value == 5)
140                     { EBox_CalTorque_S.Text = "0.060"; EBox_CalTorque_O.Text = "-19.960";}
141             //20 RPM
142                 if (TBar_RPM.Value == 2 && TBar_Torque_Limit.Value == 1)
143                     { EBox_CalTorque_S.Text = "0.069"; EBox_CalTorque_O.Text = "-22.154";}
144                 if (TBar_RPM.Value == 2 && TBar_Torque_Limit.Value == 2)
145                     { EBox_CalTorque_S.Text = "0.064"; EBox_CalTorque_O.Text = "-20.185";}
146                 if (TBar_RPM.Value == 2 && TBar_Torque_Limit.Value == 3)
147                     { EBox_CalTorque_S.Text = "0.059"; EBox_CalTorque_O.Text = "-18.205";}
148                 if (TBar_RPM.Value == 2 && TBar_Torque_Limit.Value == 4)
149                     { EBox_CalTorque_S.Text = "0.054"; EBox_CalTorque_O.Text = "-16.474";}
150                 if (TBar_RPM.Value == 2 && TBar_Torque_Limit.Value == 5)
151                     { EBox_CalTorque_S.Text = "0.051"; EBox_CalTorque_O.Text = "-14.490";}
152             //30 RPM
153                 if (TBar_RPM.Value == 3 && TBar_Torque_Limit.Value == 1)
154                     { EBox_CalTorque_S.Text = "0.050"; EBox_CalTorque_O.Text = "-18.965";}
155                 if (TBar_RPM.Value == 3 && TBar_Torque_Limit.Value == 2)
156                     { EBox_CalTorque_S.Text = "0.048"; EBox_CalTorque_O.Text = "-16.558";}
157                 if (TBar_RPM.Value == 3 && TBar_Torque_Limit.Value == 3)
158                     { EBox_CalTorque_S.Text = "0.045"; EBox_CalTorque_O.Text = "-14.137";}
159                 if (TBar_RPM.Value == 3 && TBar_Torque_Limit.Value == 4)
160                     { EBox_CalTorque_S.Text = "0.043"; EBox_CalTorque_O.Text = "-11.720";}
161                 if (TBar_RPM.Value == 3 && TBar_Torque_Limit.Value == 5)
162                     { EBox_CalTorque_S.Text = "0.042"; EBox_CalTorque_O.Text = " 09.305";}
163         }
164
165     //Buttons
166         //Data buttons
167             //Start Data button
168                 private void But_Start_Data_Click(object sender, EventArgs e)
169                     {
170                         myPort.Parity = Parity.None; // Added for analog read
171                         myPort.DataBits = 8; // Added for analog read
172                         myPort.StopBits = StopBits.One; // Added for analog read
173                         myPort.DataReceived += Myport_DataReceived; // Added for analog read
174                         //Enable/Disable buttons
175                             But_Start_Data.Enabled = false;
176                             But_Stop_Data.Enabled = true;
177                             But_Save_Data.Enabled = false;
178                             But_Start.Enabled = true;
179                     try
180                         {
181                             RBox_Data.Text = "";
182                             myPort.Open();
183                         }
184                     catch (Exception) { }
185                     }
186
187             //Stop Data button
188                 private void But_Stop_Data_Click(object sender, EventArgs e)
```

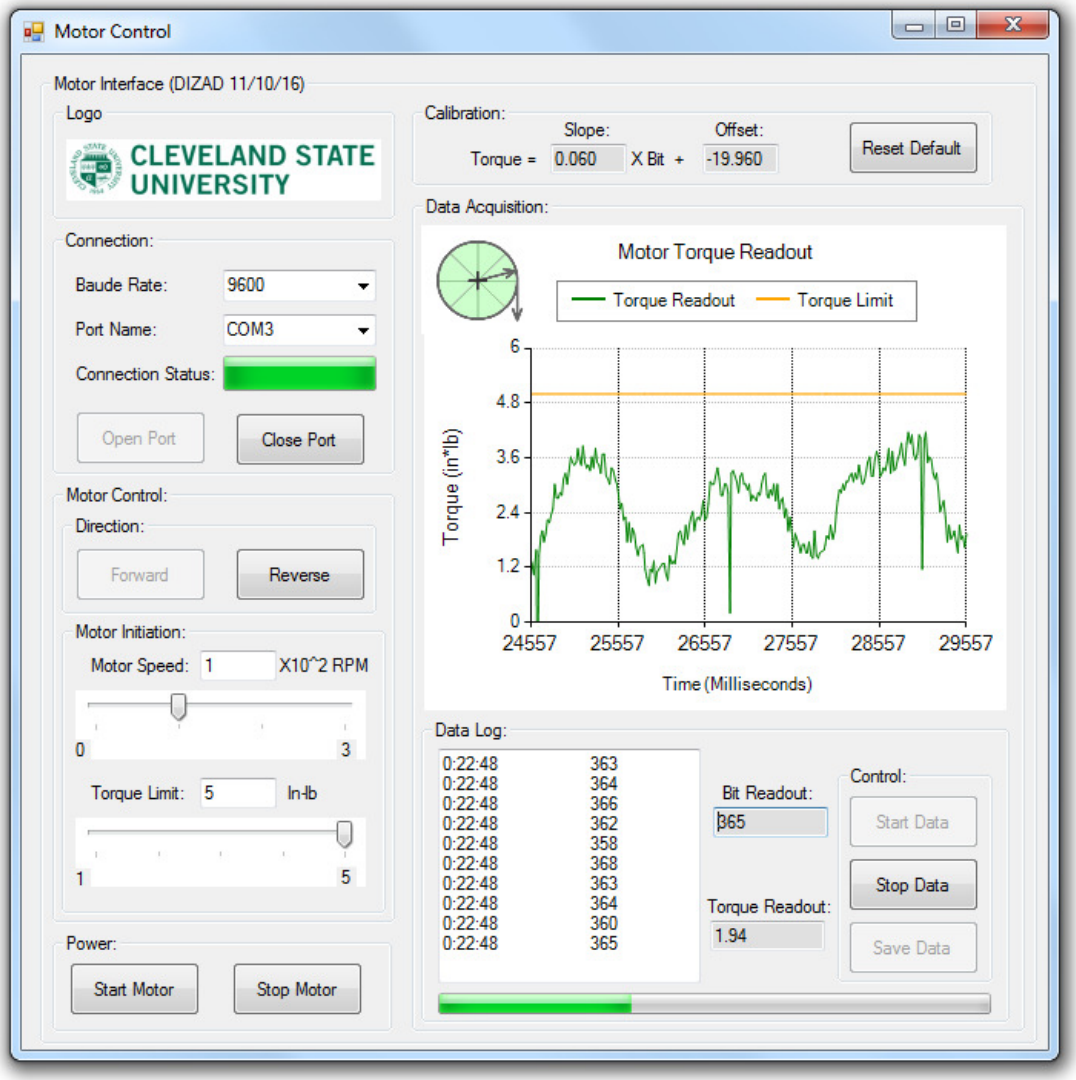
```
189                     {
190                         try
191                         {
192                             myPort.Close();
193                             But_Connect.Enabled = false;
194                             But_Close.Enabled = false;
195                             But_Forward.Enabled = false;
196                             But_Reverse.Enabled = false;
197                             But_CalTorque.Enabled = false;
198                             But_Start_Data.Enabled = true;
199                             But_Stop_Data.Enabled = false;
200                             But_Save_Data.Enabled = true;
201                         }
202                         catch (Exception){MessageBox.Show("Error");}
203                     }
204 
205             //Save Data button
206                 private void But_Save_Data_Click(object sender, EventArgs e)
207                     {
208                         try
209                         {
210                             string pathfile = @"C:\Users\DIZAD\Desktop\CHAPTER 2 08-12-16\MSME\PROJECT-
                            MECHATRONICS\PROGRAMS\C#\SAVED DATA\";
211                             string filename = "TData.txt";
212                             System.IO.File.WriteAllText(pathfile + filename, RBox_Data.Text);
213                             MessageBox.Show("Data has been saved to: " + pathfile, "Save File");
214                         }
215                         catch (Exception) { MessageBox.Show("Error"); }
216                     }
217 
218         //Calibration buttons
219             //Torque reset default button
220             private void But_CalTorque_Click(object sender, EventArgs e)
221                             {
222                                 try
223                                 {
224                                     SetDefaultCalibration();
225                                 }
226                                 catch (Exception) { MessageBox.Show("Error"); }
227                             }
228 
229         //Connection buttons
230             //Connect button
231                 private void But_Connect_Click(object sender, EventArgs e)
232                     {
233                         try
234                         {
235                             if (SBox_Baude.Text == "" || SBox_Ports.Text == "") //check for baude/port#
236                                 {
237                                     MessageBox.Show("Fill Baude Rate and Port#.");
238                                 }
239                             else
240                                 {
241                                     myPort = new SerialPort();
242                                     myPort.BaudRate = Convert.ToInt32(SBox_Baude.Text);
243                                     myPort.PortName = SBox_Ports.Text;
244                                     myPort.Open(); //Open the port so the arduino analog signal can be read
245                                     PBar_Connection.Value = 100; //Animate the status bar to complete
246                                     //Enable/Disable buttons
247                                         But_Connect.Enabled = false;
248                                         But_Close.Enabled = true;
249                                         But_Forward.Enabled = false;
250                                         But_Reverse.Enabled = true;
251                                         But_Start.Enabled = true;
```

```
252                                     But_Stop.Enabled = false;
253                                     But_CalTorque.Enabled = true;
254                                     But_Start_Data.Enabled = false;
255                                     But_Stop_Data.Enabled = false;
256                                     But_Save_Data.Enabled = false;
257                                 }
258                         }
259                         catch (Exception){MessageBox.Show("Error");}
260                     }
261
262             //Close button
263                 private void But_Close_Click(object sender, EventArgs e)
264                     {
265                         myPort.Close();
266                         FBar_Connection.Value = 0;
267                         But_Stop.PerformClick();
268                         //Enable/Disable buttons
269                             But_Connect.Enabled = true;
270                             But_Close.Enabled = false;
271                             But_Forward.Enabled = false;
272                             But_Reverse.Enabled = false;
273                             But_Start.Enabled = false;
274                             But_Stop.Enabled = false;
275                             But_CalTorque.Enabled = false;
276                             But_Start_Data.Enabled = false;
277                             But_Stop_Data.Enabled = false;
278                             But_Save_Data.Enabled = false;
279                     }
280
281         //Motor control buttons
282             //Reverse button
283                 private void BReverse_Click(object sender, EventArgs e)
284                     {
285                         myPort.WriteLine("1");//Energize case 1 of arduino
286                         I_Motor.Image = Image.FromFile("C:/Users/DIZAD/Desktop/CHAPTER 2 08-12-16/MSME/
                            PROJECT-MECHATRONICS/PROGRAMS/C#/MOTOR INTERFACE/MOTOR INTERFACE/ELEMENTS/
                            ANIMATIONS/MReverse.gif");
287                         //Disable/Enable buttons
288                             But_Reverse.Enabled = false;
289                             But_Forward.Enabled = true;
290                     }
291             //Forward button
292                 private void BForward_Click(object sender, EventArgs e)
293                     {
294                         myPort.WriteLine("0"); //Energize case 0 of arduino
295                         I_Motor.Image = Image.FromFile("C:/Users/DIZAD/Desktop/CHAPTER 2 08-12-16/MSME/
                            PROJECT-MECHATRONICS/PROGRAMS/C#/MOTOR INTERFACE/MOTOR INTERFACE/ELEMENTS/
                            ANIMATIONS/MForward.gif");
296                         //Disable/Enable buttons
297                             But_Reverse.Enabled = true;
298                             But_Forward.Enabled = false;
299                     }
300             //Motor Stop button
301                 private void But_Stop_Click(object sender, EventArgs e)
302                     {
303                         FRox_RPM.Text = "0"; //shut down the motor signal
304                         I_Motor.Enabled = false; //stop the motor GIF
305                         //Disable/Enable buttons
306                             But_Stop.Enabled = false;
307                             But_Start.Enabled = true;
308                     }
309             //Motor Start button
310                 private void But_Start_Click(object sender, EventArgs e)
311                     {
```

```
312                     //Send the speed requires to the Arduino
313                         if (TBar_RPM.Value == 0) {myPort.WriteLine("2");} //speed 0 -> 0 RPM
314                         if (TBar_RPM.Value == 1) {myPort.WriteLine("3");} //speed 1 -> 10 RPM
315                         if (TBar_RPM.Value == 2) {myPort.WriteLine("4");} //speed 2 -> 20 RPM
316                         if (TBar_RPM.Value == 3) {myPort.WriteLine("5");} //speed 3 -> 30 RPM
317                     //Enable buttons as needed
318                         But_Stop.Enabled = true;
319                         But_Start.Enabled = false;
320                         But_Start_Data.Enabled = true;
321                         But_Stop_Data.Enabled = false;
322                     //Other
323                         if (TBar_RPM.Value > 0){ I_Motor.Enabled = true;} //if motor speed> 0, play motor GIF
324                     }
325
326     //Entry boxes and meters
327         //RPM entry box
328             private void EBox_RPM_TextChanged(object sender, EventArgs e)
329             {
330                 try
331                 {
332                     //Prevent from entering a value higher than the label max value
333                     if (Convert.ToInt32(EBox_RPM.Text) > Convert.ToInt32(L_RPM_MAX.Text))
334                     {
335                         EBox_RPM.Text = L_RPM_MAX.Text;
336                         MessageBox.Show("Entry is limited to label max value.");
337                     }
338                     //Prevent from entering a value lower than the label min value
339                     if (Convert.ToInt32(EBox_RPM.Text) < Convert.ToInt32(L_RPM_MIN.Text))
340                     {
341                         EBox_RPM.Text = L_RPM_MIN.Text;
342                         MessageBox.Show("Entry is limited to label min value.");
343                     }
344                     //Value Conversion
345                         int C_RPM = Convert.ToInt32(EBox_RPM.Text);
346                         TBar_RPM.Value = C_RPM;
347                         SetDefaultCalibration();
348                 }
349                 catch (Exception)
350                 {
351                     MessageBox.Show("Invalid Entry: Enter positive integers only");
352                 }
353             }
354         //RPM meter
355             private void TBar_RPM_Scroll(object sender, EventArgs e)
356             {
357                 EBox_RPM.Text = Convert.ToString(TBar_RPM.Value);
358             }
359         //Torque limit entry box
360             private void EBox_Torque_Limit_TextChanged(object sender, EventArgs e)
361             {
362                 try
363                 {
364                     //Prevent someone from entering a value higher than the label max value
365                     if (Convert.ToInt32(EBox_Torque_Limit.Text) > Convert.ToInt32(L_Limit_MAX.Text))
366                     {
367                         MessageBox.Show("Entry is limited to label max value.");
368                         EBox_Torque_Limit.Text = L_Limit_MAX.Text;
369                     }
370                     //Prevent someone from entering a value lower than the label min value
371                     if (Convert.ToInt32(EBox_Torque_Limit.Text) < Convert.ToInt32(L_Limit_MIN.Text))
372                     {
373                         EBox_Torque_Limit.Text = L_Limit_MIN.Text;
374                         MessageBox.Show("Entry is limited to label min value.");
375                     }
```

```
376                     //convert value to an integer
377                         int C_Limit = Convert.ToInt32(EBox_Torque_Limit.Text);
378                         TBar_Torque_Limit.Value = C_Limit;
379                         SetDefaultCalibration(); //reset the calibration values
380                 }
381                 catch (Exception)
382                 {
383                     MessageBox.Show("Invalid Entry: Enter positive integers only");
384                 }
385             }
386         //Torque limit meter
387             private void TBar_Torque_Limit_Scroll(object sender, EventArgs e)
388             {
389                 EBox_Torque_Limit.Text = Convert.ToString(TBar_Torque_Limit.Value);
390             }
391             }//FORM end bracket
392     }//MAIN end bracket
393
```

4.2.4.3. C# Interface Visual:



4.2.4.4. Interface Description:

4.2.4.4.1. Connection Module:

The user begins by selecting the *Baude Rate* and *Port Name* of the Connection module to connect the interface software with the microcontroller.

4.2.4.4.2. Motor Control Module:

4.2.4.4.2.1. Direction:

Next, the user selects the direction of the motor (either forward or backwards). The reverse button sends a write signal to the microncontroller to energize 5VDC to pin51. Per the electrical schematic, pin51 connects to Z/F line of the motor driver, the reverse indicator.

4.2.4.4.2.2. Motor Initiation:

Next, the user selects the *Motor Speed* and *Torque Limit*. The Motor speed should stay consistent throughout the test, since adjusting it can affect downstream parameters. The Torque Limit defines the limit of the motor torque before it will be shut off. A conditional statement is defined in the C# software that will trigger the *Stop Motor* button when the torque limit is reached.

4.2.4.4.2.3. Power:

Once the direction, speed, and Torque Limits are defined, the motor can be energized via the *Start Motor* button. Per the electrical schematic, this button energizes PWM13 on the microcontroller, which provides pin#VR on the motor driver the power required to run the motor.

4.2.4.5. Calibration Module:

Calibration factors can also be adjusted as needed to convert the 0-1023bit signal to a useable torque value. The default is left to values that were determined through the studies discussed in the *Conversions & Calibrations* section.

4.2.4.6. Data Acquisition Module:

Finally, the *Start Data* button can be activated to begin collecting data. The chart will show live continuous data and will populate the numbers inside the readout box. The data can be stopped, and saved in a .txt format for review.

5. Validate:

5.1. Design of Experiment (DOE):

The following experiment was designed to test the performance of the system. This test used a fixed 10RPM throughout.

Figure 5.1[1]: DOE Table:

Weight	Arm	Torque Applied	Motor Speed	Torque Limit
lb	in	in*lb	RPM	in*lb
3.256	0.645	2.1	10	2
4.806	0.645	3.1	10	3
6.357	0.645	4.1	10	4

5.2. Results**:** The system turned off as expected with the following response curves.

Figure 5.2[1]: 2in*lb test:

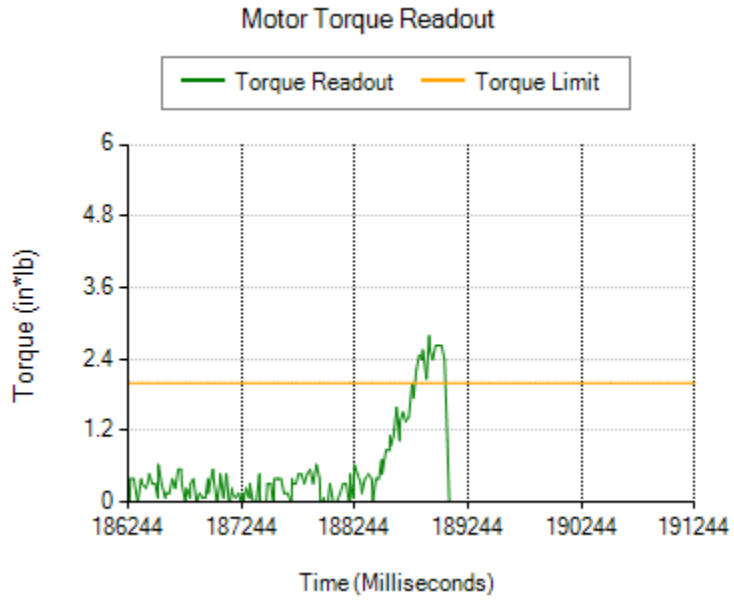


Figure 5.2[2]: 3in*lb test:

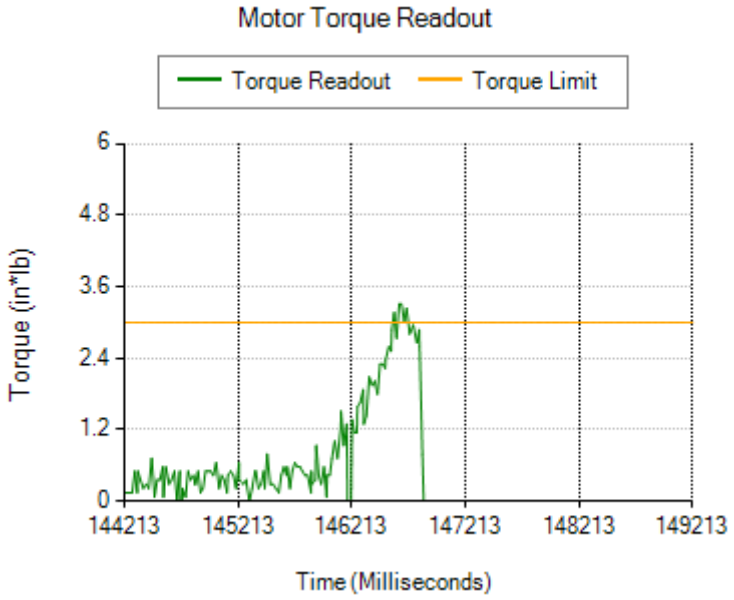
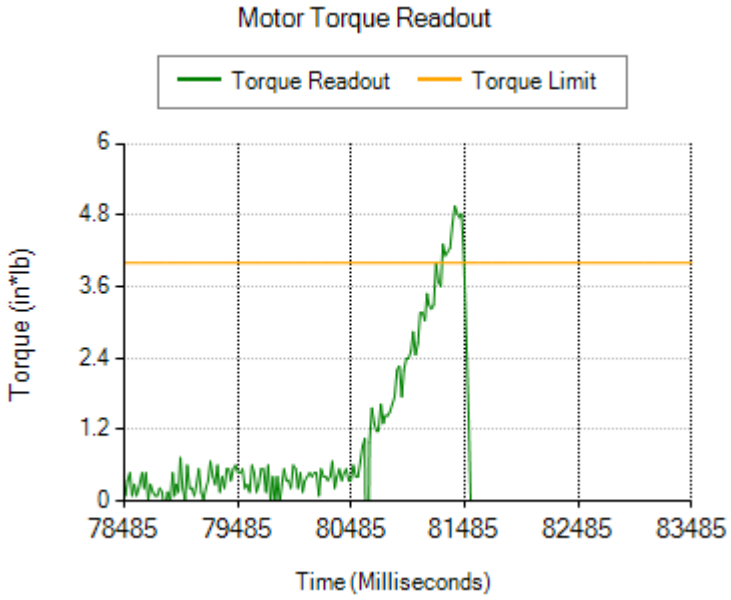


Figure 5.2[3]: 4in*lb test:



6. Close:

6.1. Conclusion:

6.1.1. The results show a proof-of-concept that a brushless-motor can effectively, affordably, and precisely be turned off provided a torque value. This working prototype can be used as a design guide for a variety of custom applications in the industry. This study provided the building blocks for a compact product re-design of a current sensor, arduino microcontroller, and motor driver all built into a single marketable product.

7. Appendix:

7.1. Logged Hours:

DATE	START	END	BREAKS (mins)	Net Hours	Module	Comments
5/26/2016	13:00	13:30	0.00	0.5	Assessment	RASHIDI
8/6/2015	N/A	N/A	0.00	1	Assessment	MOUNIR IBRAHIM
8/13/2015	N/A	N/A	0.00	0.5	Assessment	GOBERT
N/A	N/A	N/A	0.00	0.5	Assessment	GARY MATSON
N/A	N/A	N/A	0.00	0.25	Assessment	GALLAGHER
N/A	N/A	N/A	0.00	0.5	Assessment	FURLICH
N/A	N/A	N/A	0.00	0.5	Assessment	FOTTA
N/A	N/A	N/A	0.00	1.5	Assessment	MARSHALL
N/A	N/A	N/A	0.00	1	Assessment	RUBINSKI
7/25/2016	16:30	17:30	0.00	1:00	Assessment	Write report
7/25/2016	18:30	20:50	0.00	2:20	Assessment	Simplify models
7/27/2016	22:09	22:55	0.00	0:46	Assessment	Brainstorm scenarios
8/1/2016	14:00	17:00	0.00	3:00	Assessment	Consult with Marshall
8/1/2016	17:30	18:30	0.00	1:00	Assessment	Complete boundary conditions
8/5/2016	16:00	17:00	0.00	1:00	Assessment	Review with Dr.Ibrahim
8/11/2016	20:30	21:30	0.00	1:00	Assessment	Mechatronics project review
8/19/2016	22:12	22:53	0.00	0:41	Assessment	Writing report
8/22/2016	11:36	16:35	17.00	4:42	Assessment	Writing report, SolidWorks assembly
8/25/2016	5:00:00 PM	5:30:00 PM	0.00	0:30	Assessment	Review with advisor

8/27/2016	3:54:00 PM	4:26:00 PM	0.00	0:32	Define	Report update
8/28/2016	7:30:00 PM	10:00:00 PM	35.00	1:55	Define	Review scope
8/30/2016	11:18:00 PM	12:55:00 AM	10.00	1:27	Define	Review block diagram
8/31/2016	9:33:00 PM	10:26:00 PM	0.00	0:53	Define	Discussed block diagram and advised to phase the aparatus with an initial Torque to Amp setup.
9/2/2016	7:36:00 PM	8:10:00 PM	0.00	0:34	Define	Reviewed BOM and began collecting part numbers, parts, and reviewed datasheets.
9/3/2016	7:26:00 PM	9:30:00 PM	0.00	2:04	Define	Read 100 page Arduino step by step guide and installed required software to run.
9/5/2016	9:36:00 PM	11:30:00 PM	0.00	1:54	Define	Send update, first Arduino program to blink LEDs, was successful to program the unit to blink 2 LEDs.
9/13/2016	11:06:00 PM	2:02:00 AM	35.00	2:21	Define	Created a gradual signal in arduino. Tried to figure out other commands.
9/14/2016	10:24:00 PM	12:59:00 AM	26.00	2:09	Define	Began experimenting with the analog input of the ardunio. Sucessfully programmed microcontroller to take in analog signal from potentiometer and transmit it to the delay function of the output LED. The program divided the input 1023 bit signal(10ohm) to increase the speed of the LED flicker, making it dim and lighten as the potentiometer is switched.
9/18/2016	10:25:00 PM	2:29:00 AM	60.00	3:04	Define	Researched and successfully programmed arduino to take a threshold value. Researched a way to read analog input live using python .
9/19/2016	1:09:00 PM	1:30:00 AM	275.00	7:46	Define	Continue to research plotting analog signal. Successfully plotted on arduino interface but still problems with python interface. Prepared more hardware. Began wiring diagram, updated report.
9/20/2016	9:50:00 PM	2:43:00 AM	0.00	4:53	Develop	Wiring diagram + reviewing datasheets, review hall effect sensor.
9/21/2016	9:07:00 PM	2:52:00 AM	75.00	4:30	Develop	Python programming for interface.
9/22/2016	1:26:00 PM	4:44:00 PM	20.00	2:58	Develop	Python programming for interface, discussion update with advisor.
9/23/2016	9:10:00 PM	2:40:00 AM	0.00	5:30	Develop	Updated wiring schematic(rev-B), researched PWM and prompt programming. Successfully programmed a C# interface to turn on/off LEDs.
9/25/2016	9:58:00 PM	1:25:00 AM	0.00	3:27	Develop	Wire hall effect sensor and test functionality. Successful test. Also, successfully tested the PWM output.
9/27/2016	10:00:00 PM	11:41:00 PM	0.00	1:41	Develop	Researching how to turn brushless motor. Successfully turned but driver has problems picking up PWM signal, analog in only works.
9/29/2016	11:47:00 PM	1:10:00 AM	10.00	1:13	Develop	New driver ordered, research on arduino DAC.
9/30/2016	9:30:00 PM	11:40:00 PM	0.00	2:10	Develop	Began putting the parts together and brainstormed the designs required for the pulley, couplers, and fixtures.
10/1/2016	4:00:00 PM	11:45:00 PM	80.00	6:25	Develop	Wiring, designing fixtures, and reviewing datasheets.
10/2/2016	3:10:00 PM	8:30:00 PM	0.00	5:20	Develop	Started brushless motor.
10/14/2016	8:51:00 PM	12:39:00 AM	30.00	3:18	Develop	Technology research.
10/15/2016	3:13:00 PM	11:00:00 PM	80.00	6:27	Develop	Technology research + driver datasheet review+interface development.
10/16/2016	3:34:00 PM	8:00:00 PM	0.00	4:26	Develop	Interfce design, oscilloscope signal analysis.
10/22/2016	8:00:00 PM	9:00:00 PM	0.00	1:00	Develop	Meeting on torque measurements.
10/23/2016	5:31:00 PM	10:48:00 PM	0.00	5:17	Develop	Setting up aparatus, update wire schematic.
10/24/2016	1:54:00 AM	3:55:00 AM	0.00	2:01	Develop	perform wire connections, figures.
10/25/2016	10:42:00 PM	1:00:00 AM	0.00	2:18	Develop	Power, test, and current sensor inspection.
10/26/2016	4:15:00 PM	6:00:00 PM	0.00	1:45	Develop	Fixture design.
10/27/2016	10:00:00 PM	3:52:00 AM	30.00	5:22	Develop	Programming R&D-digital switches.
10/28/2016	10:33:00 PM	5:00:00 AM	40.00	5:47	Develop	Programming R&D-connectivity controls.
10/29/2016	3:48:00 PM	1:46:00 AM	60.00	8:58	Develop	Programming R&D-data acquisition.
10/30/2016	11:00:00 PM	4:07:00 AM	20.00	4:47	Develop	Programming R&D-data acquisition + others.
10/31/2016	8:52:00 PM	1:13:00 AM	20.00	4:01	Develop	Fixture development + programming.
11/1/2016	1:45:00 AM	4:53:00 AM	30.00	2:38	Develop	Programming

11/2/2016	1:13:00 AM	2:36:00 AM	0.00	1:23	Develop	Programming
11/3/2016	4:00:00 PM	4:40:00 AM	100.00	11:00	Develop	Programming, report, calibration studies
11/5/2016	8:38:00 PM	1:45:00 AM	0.00	5:07	Develop	Programming, conversions
11/6/2016	8:30:00 PM	11:00:00 PM	0.00	2:30	Develop	Conversion map
11/7/2016	9:45:00 PM	2:26:00 AM	20.00	4:21	Develop	Torque calulations and DOE testing on different torque values.
11/9/2016	2:14:00 PM	10:52:00 PM	60.00	7:38	Develop	Torque conversions + updated program + testing, clean program.
11/11/2016	7:30:00 PM	12:30:00 AM	20.00	4:40	Validate	Report, DOE testing
11/12/2016	3:56:00 PM	7:30:00 PM	20.00	3:14	Validate	DOE testing
11/13/2016	4:00:00 PM	12:00:00 AM	0.00	8:00	Close	Report, desktop cam
11/14/2016	8:35:00 PM	1:30:00 AM	0.00	4:55	Close	Report
			TOTAL =	197		
			REQUIRED HOURS =	80		
			% COMPLETE =	246%		

7.2. References:

7.2.1. Websites:

7.2.1.1. Supplier sites

7.2.1.1.1. www.digikey.com
7.2.1.1.2. www.allegromicro.com
7.2.1.1.3. www.arduino.cc
7.2.1.1.4. www.faulhaber.com
7.2.1.1.5. www.mcmaster-carr.com

7.2.1.2. Technical Sites

7.2.1.2.1. www.rethinkrobotics.com
7.2.1.2.2. www.kuka-robotics.com
7.2.1.2.3. www. stackoverflow.com
7.2.1.2.4. www.wikipedia.com
7.2.1.2.5. www.assemblymag.com
7.2.1.2.6. www.atlascopco.com
7.2.1.2.7. www.translatorscafe.com
7.2.1.2.8. www.electronicdesign.com

7.2.2. Books:

7.2.2.1. The Art of Electronics by: *Paul Horowitz*
7.2.2.2. Shigley's Mechanical Engineering Design by: *Richard G. Budynas*
7.2.2.3. Machinery's handbook by: *Erik Oberg*
7.2.2.4. Arduino Step-by-Step Guide by: *Peter Dalmaris*
7.2.2.5. Robotics-Theory and Industrial Applications by: *Larry T.Ross*

7.2.3. Journals:

7.2.3.1. IEEE/ASME Transactions on Mechatronics

7.2.4. Conferences:

7.2.5. IRIS (The Electronic Technology Expo/Symposium)