Portland State UNIVERSITY

Brushless DC Motor Electric Scooter

Version 1.00
Dec 6th, 2016

Forrest Kaminski
Thuan Pham
Justin Morgan

Group # 12

Table of Contents:

Needs Statement

With growing populations in urban centers, more and more commuters shy away from the typical combustion car as a primary form of transportation. With average speeds as low as 15-20 miles an hour, bicycles seem to be dominating short ranged transportation. With the recent improvements in lithium ion battery technologies, highly efficient electric bicycle hybrids are a popular, albeit expensive replacement for the traditional bicycle. While bicycles can be used for commutes of several miles, not all commutes are this far.

Many people, especially students at PSU live only a few miles from work or school, use a bicycle as their primary transportation, but find that leaving a bicycle outside leaves it prone to theft, or for secure parking, it requires a paid pass.

An electric scooter fills the niche of lightweight and inexpensive short range transportation, with parts costs ranging only in the realm of a few hundred dollars, compared at the $1000 electric bicycles you can find today.

Requirements

- Attach an electric motor to either an existing or manufactured kick board scooter for use as a personal electric vehicle.
- The vehicle must be capable of moving a full sized adult a constant speed of at least 6 miles per hour
- The motor controller should work efficiently enough so that you can travel more than 2 miles with 133 watt/hours of battery
- The vehicle should be able to transmit relevant driving information to the rider such as speed and battery voltage.
- The final vehicle including the motor, motor controller, scooter base and battery must be less than $300.

Project Proposal

Objectives

Using an electric drive motor, create a solution that will implement a standard scooter with an electric motor which will power this vehicle. The vehicle must function smoothly and safely. While giving the operator complete control over speed, acceleration and direction of motion.

- Develop a working electronic speed controller based on an AVR microcontroller
- Make use of PID control to create a smooth output drive for the electric scooter application
- Attach a sensored BLDC motor with a battery as well as a ESC and have a fully functioning electric scooter
- If time allows, attach a bluetooth HC-06 module and create a smartphone application to readout speed and battery data.

Scope

The end result should be a usable light transportation vehicle that functions at the flip of a switch, but could potentially have additional functionality with a smartphone as an output display.

Timeframe

	Task	Start and End Dates
Phase One	Learn and implement BLDC controller with Arduino	Oct 4th - Oct 17
Phase Two	Transfer Arduino code to atmega328p using atmel AVR - print new circuit board	Oct 17th - Nov 5th
Phase Three	Attach motor, ESC and battery to scooter, verify it will carry the load of a person	Nov 5th - Class end

Project Budget
Project budget is $300 for one finished product, which limits parts spending to about $250 for all the parts for the board as well as the scooter, so that we have some leftover money to purchase the PCB from OSH park.

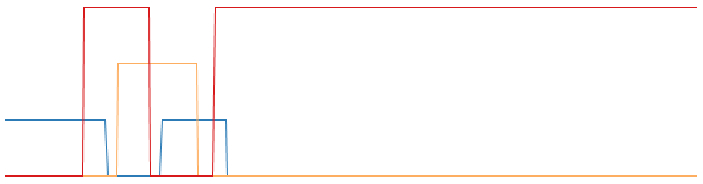
Concept Outline

Each concept is a segment of the overall design process that has been realized as a non-trivial design challenge that must be addressed. Each section outlined describes the physical or electrical requirements, how they can be completed and any issues that may arise in each case.

Functionality
The scooter will be able to transport a user of various weights around by using its brushless dc motor. To control the motor the user will turn down the the throttle hand that is built into the scooter hand. Depending on the position of the hand the scooter will will various amounts of power. The more turned the handle, the more power to the motor which results in a higher speed.

Brushless Motor Control
We are creating an ESC (Electronic Speed Controller) for our scooter which entails the production of a PCB using low voltage MOSFET switches to power the gates of high current MOSFETS which in turn will drive our 3 phase brushless motor. The processor Atmega328p processor to integrate PWM (pulse width modulation) to control the motor drive speed and levels of throttle.

Coil energizing for a sensored brushless motor involves reading the hall effect states, and matching 1 one of the 6 states you are in to a motor phase. For our prototype, we mapped the motor phases to the motor’s hall effect sensors and ensured the the hall effect sensors worked. Our motor comes with three integrated hall effect sensors, which we used as arduino input using the internal pullup resistors.



Example of arduino hall effect outputs as motor spins

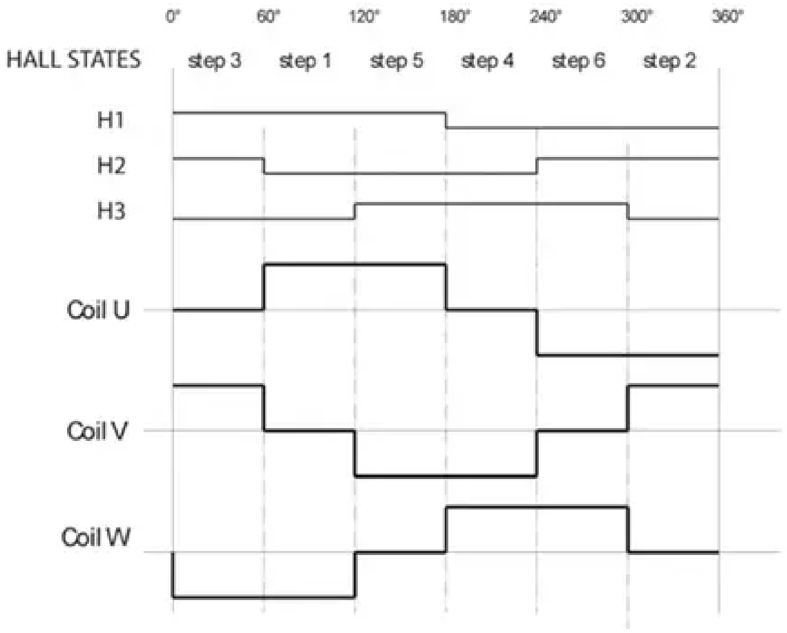
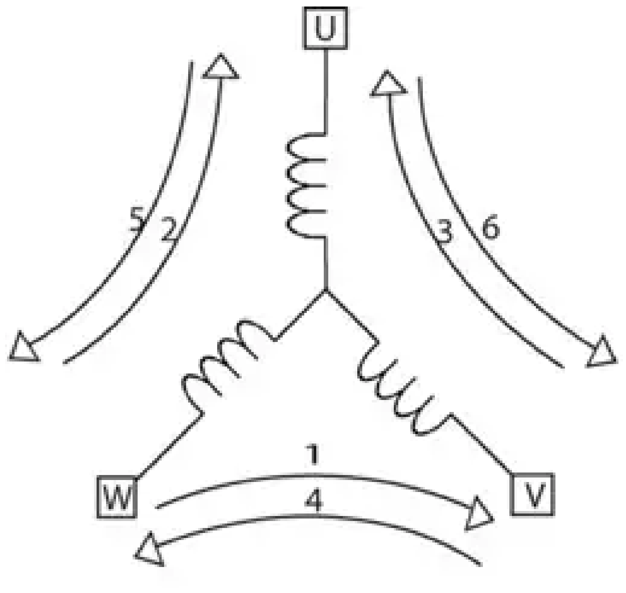


Table: Hall Effect Sensor Motor Position

The hall effect sensors will directly map to the coils energizing for our outputs, for example, if we're in state 3 (step 3) then coil U is untouched, which V is high and W is driven to ground. We will always transition to the right on the diagram above, so if we are in step 4, the next state if we are going forward will always be step 6.



Coil current direction

The coils with push and pull the corresponding magnets depending on how they are energized according to the diagram above.

PWM

We will use the Atmega328p six pulse width modulation pins to control the 6 mosfets. The duty cycle of the modulation will determine the power the motor puts out, and in turn will be our mechanism for altering the motor's speed.

Atmega328p and Arduino

While the final system will be implemented with the Atmega328p processor programmed through Atmel Studio, we will prototype the motor controller using an arduino controlling an array of mosfets, similar or identical to what will be on our final board. To ensure we can spin the motor, we will map the firing of the coils to the hall effect sensors using a control loop similar to a lookup table.

Microprocessor Control Code

- Once the microprocessor and board design is finalized, basic low level control must be established via programming. Programming the microprocessor can be done in many ways, but one of the most common methods is using the ISP interface to program while it is on-board. The Atmega328P microprocessor we have selected will require a 6 pin ISP connector and Atmel Studio.

Bluetooth Connectivity

- If time allows, attach a bluetooth module to our microprocessor to transmit the inputs to the processor to a smartphone app. This is related to the following conceptual topic.

Smartphone Speed/ Battery Sensor App

- If time allows, develop a smartphone enabled application with GUI that allows for battery monitoring and other sensor details like temperature, and even speed (which is reported from hall effect sensors)

Mechanical Implementation

To fully integrate the electric motor and scooter, we will use an existing adult sized scooter and modify it to fit our motor and electronics. This will require extensive modification of the rear wheel assembly. The most compact version, in order to make the most minimal visual adjustments of the scooter, will require a drive train consisting of a belt and pulley system. Mounting the

Motor Electric Power Calculations

- This is the physics of electric propulsion. Will the battery last long enough for the weight? Can the motor deliver enough power to propel a certain weight at a specific speed ? Is the electrical circuit safe ie can we regulate amperage and voltage to protect components and the user? We must take into account the specifications of the products we choose and see if they can feasibly meet the requirements.

Power protection

- To protect the circuit and pcb, the motor controller will feature a battery protection system, cutting off power once the 22.2V battery pack reaches a minimum voltage of 18V. The pcb will be thermally protected via passive cooling as well as an on board TMP36 module thermometer to ensure the board does not overheat, resulting in dangerous shorts. The scooter will also have a battery disconnect for saving on passive power

Implementation

Schematic and Board Design:

The Schematic can be found here.

It incorporates 4 main systems, the processor, the power mosfet bank, the gate drivers, and the voltage regulators. The processor is an implementation of an Atmega328p with several filters in the form of 22pF capacitors distributed among the board. There is also a nearby TMP36 temperature sensor to monitor system temperature in case heat becomes an issue on the board.

The gate driver is a double not gate 15V logic booster implemented with 2n7000's. This was a design decision because the typical push pull NPN and PNP transistor did not appear to be working for me, so I had to improvise with what I had on hand. The first 2n7000 is active low, switching a second active low 2n7000 negating the power mosfet's gate. This way, when there is 5V on the first 2n7000, there is 15V on the power mosfet, unfortunately it has to charge through a 10k or 3.3k resistor, dramatically slowing the mosfet charge time. Unfortunately, this means if the second 2n7000 is destroyed for whatever reason, the power mosfet is locked on, which can create a short circuit condition.

The power mosfet bank is implemented with 6 IRFS7734 high power HEXFETs which feature high power dissipation, 197A of continuous current and up to 75V on the drain. This mosfets are intended for switching power supplies or electric motor drive applications. They also have a built in high power rectifiers.

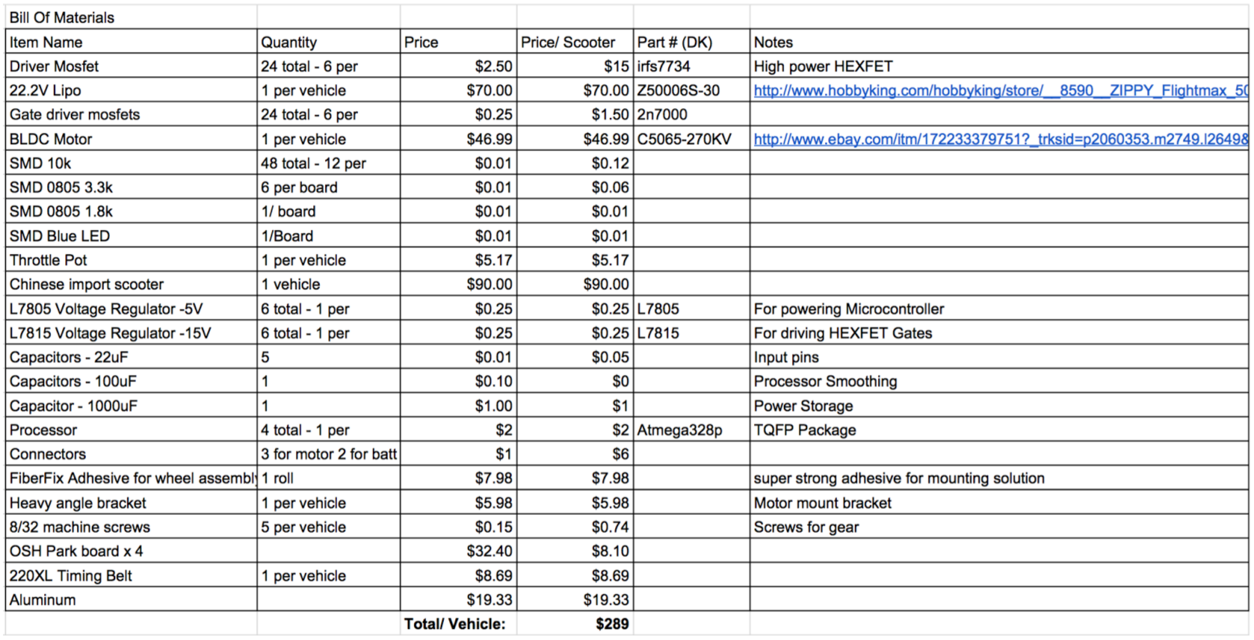
The voltage regulators are simple. The first one steps down the 22.2V lipo battery and converts it to 15V, which is used by the mosfet gates and the 5V regulator, which powers the processor and temperature sensor. There are several capacitive filters in the event of brownouts.

Processor and PWM[2]:

To get around the slow charge times of the power mosfet gate capacitors through the 10k resistor, and still have high frequency PWM which is more efficient, we implemented a second layer of software PWM which is controlled by the throttle. The first layer of hardware PWM is fixed at 1 minus its maximum duty cycle to prevent blast through scenarios. The PWM frequency is fixed at 64k hz, with the software PWM being 300 hz.

In the interest of time, we did not implement any serial outputs or bluetooth functionality. PID will also be added at a later time.

Parts List and Cost



Bill Of Materials					
Item Name	Quantity	Price	Price/ Scooter	Part # (DK)	Notes
Driver Mosfet	24 total - 6 per	$2.50	$15	irfs7734	High power HEXFET
22.2V Lipo	1 per vehicle	$70.00	$70.00	Z50006S-30	http://www.hobbyking.com/hobbyking/store/__8590__ZIPPY_Flightmax_50
Gate driver mosfets	24 total - 6 per	$0.25	$1.50	2n7000	
BLDC Motor	1 per vehicle	$46.99	$46.99	C5065-270KV	http://www.ebay.com/itm/172233797751?_trksid=p2060353.m2749.l2649&
SMD 10k	48 total - 12 per	$0.01	$0.12		
SMD 0805 3.3k	6 per board	$0.01	$0.06		
SMD 0805 1.8k	1/ board	$0.01	$0.01		
SMD Blue LED	1/Board	$0.01	$0.01		
Throttle Pot	1 per vehicle	$5.17	$5.17		
Chinese import scooter	1 vehicle	$90.00	$90.00		
L7805 Voltage Regulator -5V	6 total - 1 per	$0.25	$0.25	L7805	For powering Microcontroller
L7815 Voltage Regulator -15V	6 total - 1 per	$0.25	$0.25	L7815	For driving HEXFET Gates
Capacitors - 22uF	5	$0.01	$0.05		Input pins
Capacitors - 100uF	1	$0.10	$0		Processor Smoothing
Capacitor - 1000uF	1	$1.00	$1		Power Storage
Processor	4 total - 1 per	$2	$2	Atmega328p	TQFP Package
Connectors	3 for motor 2 for batt	$1	$6		
FiberFix Adhesive for wheel assembl	1 roll	$7.98	$7.98		super strong adhesive for mounting solution
Heavy angle bracket	1 per vehicle	$5.98	$5.98		Motor mount bracket
8/32 machine screws	5 per vehicle	$0.15	$0.74		Screws for gear
OSH Park board x 4		$32.40	$8.10		
220XL Timing Belt	1 per vehicle	$8.69	$8.69		
Aluminum		$19.33	$19.33		
		Total/ Vehicle:	**$289**		

Brushless DC Scooter: System Test Plan

1.0 Introduction

2.0 Documentation

2.1 BLDC Scooter System Specification
2.2 BLDC Scooter Block Diagram
2.3 BLDC Scooter Motor Controller Schematic
2.4 BLDC Scooter Control Algorithm
2.5 Applicable Oregon Electric Bicycle Regulations

3.0 BLDC Scooter Overview

3.1 Operational Description
3.2 Motor Speed Control
3.3 Motor Speed Calculation

4.0 Pretest Preparation

1.0 Introduction

This document serves to introduce and guide areas in the BLDC Motor Powered Scooter that require testing. The Scooter is comprised of a brushless motor driver connected to a 22.2V LiPo battery as well as a sensored motor, and these are mounted to a custom modified scooter, assembled from aluminum and connected to the motor via a geared belt drive. Elements that will require testing are the motor controller's speed control and feedback systems, ensuring it responds correctly to throttle input and reports correct speed, battery voltage and temperature information. The physical scooter assembly will require testing to see if it can withstand user weight as well as correctly being connected to the drive system. The entire system will need to be tested, ensuring that it meets range specifications given its battery capacity.

2.0 Documentation

2.1 BLDC Scooter System Specification

The scooter must adhere to the following requirements:

- The vehicle must be capable of moving a full sized adult a constant speed of at least 6 miles per hour
- The motor controller should work efficiently enough so that you can travel more than 2 miles with 133 watt/hours of battery
- The vehicle should be able to transmit relevant driving information to the rider such as speed and battery voltage.

- The final vehicle including the motor, motor controller, scooter base and battery must be less than $300.

2.2 BLDC Scooter Block Diagram

The motor controller acts as the brain and the muscle for the system, energizing the motor's power coils based on its position given by the hall effect sensors, with a speed determined using PID control and the scooter's throttle. The motor controller using pwm will fire the power mosfets to move the motor. The controller is capable of giving debug information such as speed, battery voltage and board temperature.

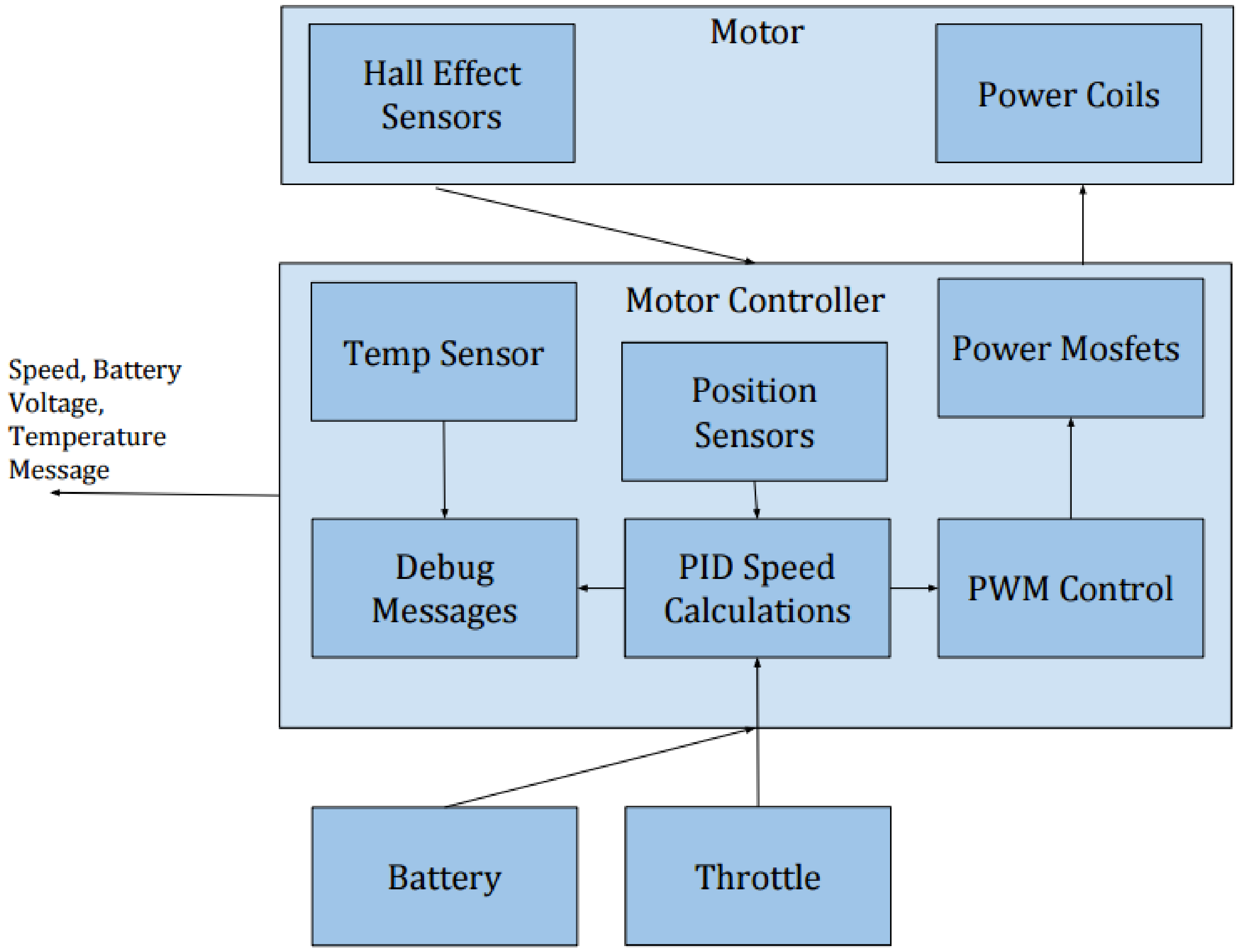


Fig. 1: System Block Diagram

2.3 BLDC Scooter Motor Controller Schematic

- *BLDC Motor Controller Schematic.pdf*

2.4 BLDC Scooter Control Algorithm

- *Brushless DC Motor Scooter.pdf, Concept detail, Microprocessor Control Code*

2.5 Applicable Oregon Electric Bicycle Regulations

In Oregon, there are regulations for any "power assist pedal bicycle", which while technically does not apply to us, has potentially relevant regulations should other forms of power assisted vehicles come to market. We should be aware of the regulations in place pertaining to motor power as well as vehicle speed in particular. Oregon states under ORS 801.258 that the vehicle does not exceed a power output of 1000W, and that the vehicle does not exceed 20 mph on level ground. Both of these can be regulated in software, as the motor controller calculates speed. The speed can be regulated to less than 20 mph if needed, and similarly our power output can be regulated by reducing our average current to the motor using pulse width modulation.

3.0 BLDC Scooter Overview

3.1 Operational Description

Attach an electric motor to either an existing or manufactured kick board scooter for use as a personal electric vehicle. The vehicle must be capable of moving a full sized adult a constant speed of at least 6 miles per hour. The motor controller should work efficiently enough so that you can travel more than 2 miles with 133 watt/hours of battery
The vehicle should be able to transmit relevant driving information to the rider such as speed and battery voltage. The final vehicle including the motor, motor controller, scooter base and battery must be less than $300.

3.2 Motor Speed Control

The electric BLDC motor is attached to a small PCB equipped with a microprocessor, which functions as the speed controller. There are two segments on this board. The right half of the board contains the BLDC circuit arrangements of mosfets and resistors, and the left half contains an Atmega328P 8 bit microprocessor which functions as a controller. It takes input from the throttle and converts that into a value which corresponds to correct sequence of MOSFETS that must be fired in correct order to move the motor. The power for this circuit is delivered by a 22.2V LiPo battery pack mounted nearby, in the motor assembly.

3.3 Motor Speed Calculation

The motor speed is classified in our case as rotations per minute (RPM), and is calculated by monitoring the motor's changes in state. The motor will change its phase a total of 45 times in a full rotation, so the changes in phase are counted to 45 and then the rpm is taken by measuring the time taken to make the full rotation. For conversion to miles per hour, the rpm will be multiplied by the gearing ratio (15: 50) and then multiplied by the circumference of our wheel.

4.0 Pretest Preparation

4.1 Test Equipment

- Tachometer
- Measured 2 mile long distance
- Computer with a type A USB port and some form of tty terminal.
- 6 cell LiPo capable battery charger.
- Thermometer capable of measuring PCB temperature.
- Oscilloscope Tektronix
- True RMS Multimeter
- Atmel Studio IDE
- AVR Dragon Programmer

4.2 Test Setup and Calibration

For measurements involving distance the scooter will need to have level terrain and a pre measured 2 mile distance. The distance does not have to follow a straight path if need be.

For measurements involving verifying board readouts, such as temperature or speed, the on board sensors will require a calibrated thermometer and tachometer to check against the board tachometer and temperature sensor.

5.0 Unit Tests

5.1 Temperature Report

Monitor and record the operating temperature of controller circuit board. Using the included on board temperature sensor, confirm that the temperature is accurate to the RPM readings. The purpose of this report is to confirm the accuracy of the temperature sensor, and verify it functions properly.

5.2 RPM Report

Monitor the RPM status message output from the motor controller and verify that it is accurate to +/- 5%. The speed will be provided in the form of RPM, and should be compared with a calibrated tachometer to verify that the speed report is correct and functions properly.

5.3 Correct Logic Level

Test correct logic level outputs from microprocessor and verify all possible logical sequences are producible and are valid. This logic level is critical to our system because they will control the mosfets which will send power to the motor through various cycles and speed. Without correct logic levels our controller will not work. Logical levels will be 5 volts +/- .3V.

5.4 Software Control

Test that the uploaded program code is working correctly. The easiest way to do this is to connect to the output port and verify you are receiving accurate debug rpm, temperature and battery voltage measurements. This can be done by checking the battery voltage readout with the battery voltage using a multimeter. To check that the software is correctly working at controlling the mosfets, check the voltage of the test pins provided on the board to ensure that the processor is outputting the correct pulse width values on the correct pins by applying throttle while the motor is unplugged.

5.5 Logic level switching MOSFET Check

Test correct voltage output on mosfets. This test is to make sure each mosfet outputs the correct voltage value for each of the six possible combinations inputs. This will make sure our mosfets are working correctly and corresponds to the logic inputs. All measurements will be measured via multimeter and looked up on our table of inputs/outputs.

5.6 Power MOSFET Check

Test correct voltage output on mosfets. This test is to make sure each mosfet outputs the correct voltage value for each of the six possible combinations inputs. This will make sure our mosfets are working correctly and corresponds to the logic inputs. All measurements will be measured via multimeter and looked up on our table of inputs/outputs.

5.7 Voltage Regulator

This test will check if the voltage regulators will output the correct stepped down voltages. Main power will be from a 24 volt battery and the voltage regulator will step the voltage down to 15 volts and the second voltage regulator will step the 15 volts future down to 5 volts. This will give us two rails different voltages to use. The 15 volts to drive the power MOSFET's and the 5 volts to power our processor and logic level mosfets. This test will look at the voltage outputted from each regulator.

5.8 Reset Button

Test the reset function of the processor. This test is to make sure the rest button is function correctly. To test this we will press the button and confirm that the program that is currently running on the microprocessor will be reseted.

5.9 Hall Effect Sensors

Test the hall effect sensors on the motor. This test will connect the hall effect sensors from the motor to our board, from there we will check if all possible 6 positions are detected and read properly to our processor. This test is key to our system, without the current position of the motor there would be no way to move the motor correctly.

5.10 Potentiometer for Throttle

Test the throttle handle on our scooter. This test makes sure the throttle outputs a mapped value of 0-255. This is key to controlling how fast the motor will spin. The 0-255 values will be used to control the PWM of the circuit thus changing the power. Make sure in rest position of throttle it outputs a 0 and as we turn it to maximum it will linearly increase to 255 where the handle will max out.

5.11 PWM Outputs on Logic Levels

This test will check the PWM logic outputs onto our logic level mosfets. Use oscilloscope to confirm the duty cycle and frequencies of each pulse width output. This test will make sure our power control on the motor will work correctly because the system uses PWM logic levels to control the amount of power to the motor.

5.12 ISP Programmer Port

This test will confirm if the ISP programming port is working properly. The ISP programming port is very important. If the port does not work the microprocessor cannot be programmed. Will be using AVR Dragon Programmer to test this port. If chip is programmable and detected, unit test success.

5.13 USB Port Pins

Test will confirm if USB port pins are working. Will be tested using serial cable, if serial data is sendable and receivable, this verifies serial port works. This is significant to our program debugging software.

5.14 External Clock

Use Oscilloscope probe to verify correct 16 MHz frequency. This will be important for serial communication.

5.15 Brushless Motor

The Brushless DC motor used is a 2 HP motor around two inches in diameter and 3 inches in height. The three hall effect sensors attached provide info on the positioning of the motor while undergoing a full revolution. There are six stages in a revolution. Test each stage and map them by using arduino programmer. Mapping each stage by comparing to BLDC control graph will also allow mapping of the correct MOSFET output.

5.16 Li-Po Battery

Test Voltage and current on LiPo battery via multimeter. Make sure battery is in good health and chargeable. Will be using Li-Po charger to check battery status.

5.17 Power Status LED

For programming and also to verify system state, This LED should receive power through 5V input. When this LED is on, the system is on.

5.18 Atmega 328p

This tests will confirm if the microprocessor chip is working or dead. Will connect microprocessor chip to AVR Dragon Programmer, if chip is detected then it will pass the test. Will also check each solder joint on the chip to confirm no bridging of any solder joints.

5.19 Gears of Drive Train

Test efficiency of drivetrain. Motor Assembly requires two gears, one driven by motor and another directly attached to rear wheel. Belt should be tight with no slack over the gears. Belt must withstand operational conditions of friction, heat, longevity. Belt must not fracture / break at full motor operational speed.

5.20 Motor Mount

Test placement and strength of motor mount within motor assembly.   Motor mount should have proper spacing for mounting, and the thickness of metal shall be sufficient to withstand any vibrations/operational conditions that may occur.

6.0 Integration Test

6.1 On board controller to BLDC circuit

After verification of BLDC circuit functioning with Arduino code, and after programming on board controller component, connect output pins of board controller to the input pins of BLDC. The controller should function identically to the arduino run code, powering the BLDC motor efficiently but without the complex arduino connections.

6.2 Battery to motor controller

After successful test of motor controller with Bench power supply, test successful operation of Motor controller and the corresponding BLDC motor while connected and powered by 11.2 V and 22.2V battery. The differences in current require that the circuit not short or overload, damaging components.

6.3 Throttle to motor controller

Having mapped our hall effect states, with arduino code the motor can only be run at one set speed with the firmware. This test will take the potentiometer / throttle and connect the power, ground, and input to the arduino. Map the inputs in firmware to test speed control. Motor should speed up and slow down proportional to the position of the throttle.

6.4 Motor Assembly On Scooter

After verifying motor assembly functions, and construction of scooter chassis capable of receiving motor assembly, confirm fit and finish and that the assembly functions identically on scooter compared to outside of scooter.

7.0 System Test

7.1 Motor Speed Control

Confirm that the scooter's motor speed responds proportionally to the throttle on the scooter. This should be done by attempting to speed the scooter quickly along the motor's power curve, and similarly it should be done slowly, by following the motor's efficiency curve.

7.2 Scooter Assembly test

Confirms that the scooter assembly is structurally capable of supporting the weight of a human plus some cargo such as a backpack. For this test, the tester should place about 220 pounds of force on the board spread over the area of a typical adult human foot. The custom wheel assembly must remain tight with no issue.

The motor should be coupled with a belt to the scooter. When the motor wheel spins, the belt should drive the motor with no slipping and visa versa.

8.0 Operational Acceptance Test

8.1 Motor Speed Test

Test Case Name: Verify Motor Speed			
Description: Checks that the on board motor speed calculator is accurate, giving accurate results that can be used to calculate vehicle speed. This is not a vehicle speed test.			
Setup: Motor assembly detached from scooter, no belt required. Test should be run on a workbench with tachometer.			
Step:	**Actions:**	**Expected Result:**	**Comments**:
1:	Connect to the processor over the provided USB connection and open the corresponding ttyUSB or COM port.	The reported rpm by the processor should be 0 rpm until the motor is spun.	
2:	Spin the motor using the throttle through a range of speeds , beginning at stopped (0 rpm) to it's maximum speed (650 rpm) and measure the rpm of the motor using a tachometer, comparing against the rpm readout over USB.	The rpm readout over usb should now be equivalent or within 5% of the tachometer reading.	
3:	Bring the motor to a stop, get a final rpm reading and disconnect the usb port and unplug the battery.	The motor should return to 0 rpm and the board should now be powered down.	
Overall test result:			

8.2 Motor Startup Torque Test

Test Case Name: Motor Startup Torque Test			
Description: Measure that the torque is sufficient to move the weight of a full sized adult. plus the entire weight of the vehicle. Purpose is to determine whether torque is suitable, not finding discrete torque values in lb-ft. The scooter is allowed to fail this test, as it was not a part of the original design requirements.			
Setup: Place a full sized adult with known weight close to 180 lbs on scooter, start scooter at rest.			
Step:	**Actions:**	**Expected Result:**	**Comments**:
1:	Place test load on vehicle, 180 pounds on vehicle, To simulate human passenger.	The motor should move full weight from starting velocity of 0 m/s	
Overall test result:			

9.0 Parametric Test

9.1 Overall Scooter Range / Battery Charge

Test Case Name: Battery Range Test			
Description: Measure the range and overall power consumption of the included 22.2V battery. The aim is to set several fixed distances and determine whether the battery can power the scooter for the entire distance.			
Setup: Charge battery fully, for a 22.2V 133 watt/hour LiPo battery, fully charged will be 25V. Scooter should be ridden with a full sized adult for duration of test.			
Step:	**Actions:**	**Expected Result:**	**Comments**:
1:	Startup and mount vehicles,	Scooter should be able to	

	start accelerating to 20 mph and maintain speed.	accelerate passenger to top speed of 20 mph.	
2:	Maintain speed of 20 mph for 2 miles with less than 50 feet of elevation gain or decline while allowing the motor controller to handle cut off voltage (if applicable).	The scooter should maintain the top speed of 20mph for the entire duration of the 2 mile ride. The scooter will cut off power once the battery reaches 18V, our depleted charge voltage.	
Overall test result:			

10.0 Stress Testing

10.1 Motor Stress Test

Test Case Name: Motor Stress Test			
Description: Measure the durability of the motor. In comparison to range test, this case is testing the durability and performance of the BLDC motor in running continuously. The circuit components should maintain normal operating temperature and the motor should be able to run.			
Setup: Charge battery fully, fully charged will be 25V. Use device with rolling resistance, either a treadmill or proper dynamometer. Position scooter so it remains upright without human assistance.			
Step:	**Actions:**	**Expected Result:**	**Comments**:
1:	Place load on Scooter, on device that provides rolling resistance, like treadmill or Dynamometer. Set and hold throttle to max speed, run this scooter for 20 minutes.	Motor should run continuously for the duration without failure or overheating of any component.	
2:	Suspend scooter so that wheels do not touch ground, ie no resistance, no load on scooter. Set and hold throttle to max speed, run for 30 minutes	Motor Should run continuously for the duration without failure or overheating of any component.	

Overall test result:	

11.0 Alpha Testing

11.1 Alpha test for scooter

Test Case Name: Scooter Alpha Test			
Description: Have each team member to test scooter by riding around campus			
Setup: Completely build scooter, get rider to test			
Step:	**Actions:**	**Expected Result:**	**Comments**:
1:	Get on scooter and ride around at various speeds.	Scooter moving at 5 - 10mph. Change speed via throttle on handle	
Overall test result:			

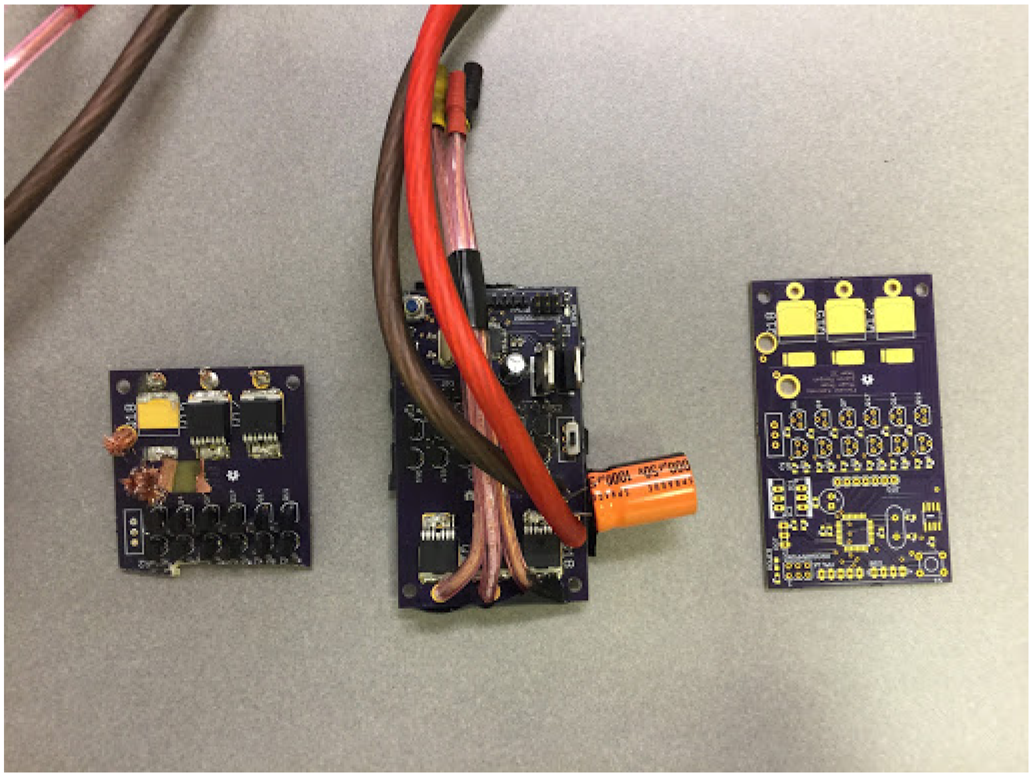
12.0 Environmental Testing

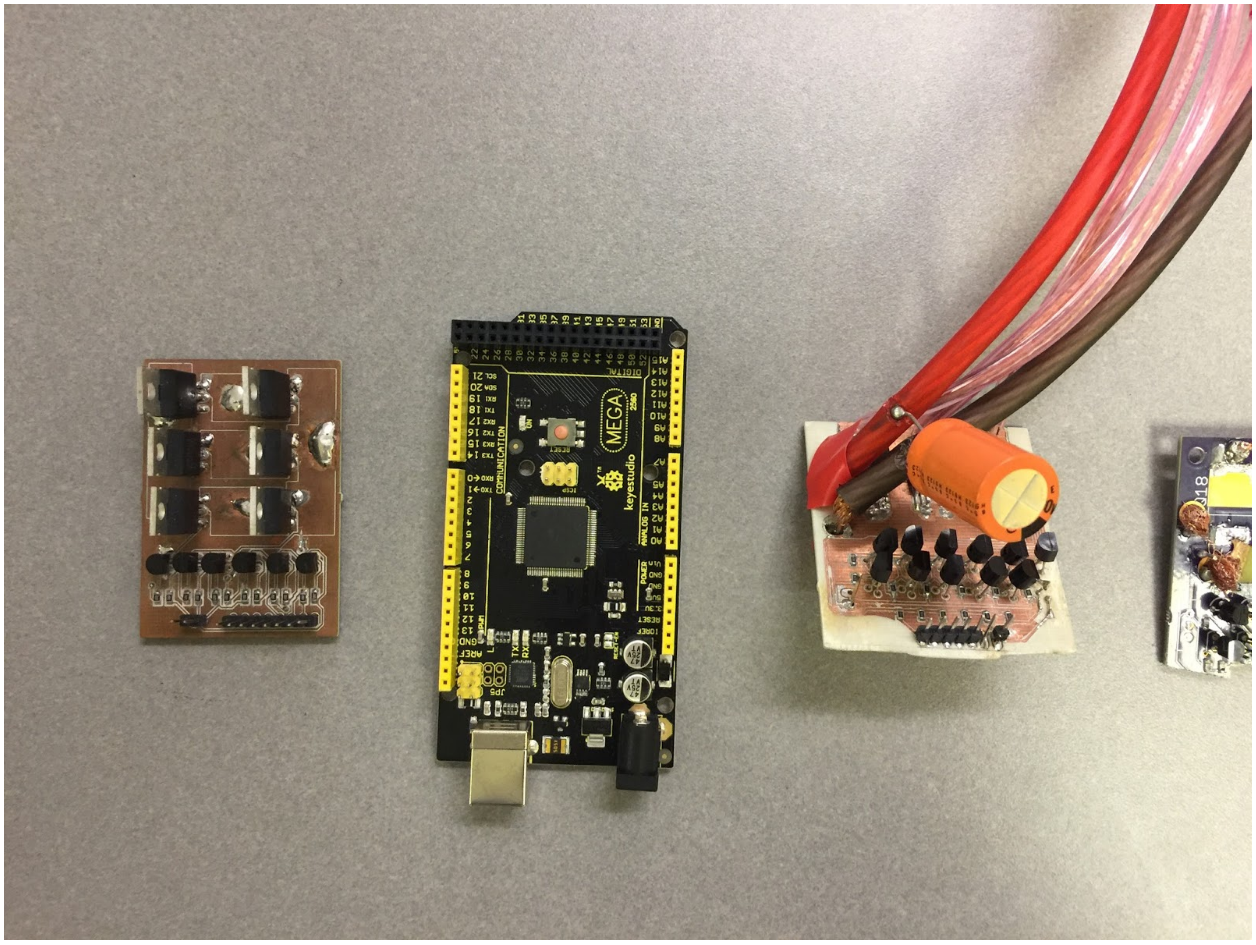
12.1 Scooter environmental conditions

Test Case Name: Scooter Environment Test			
Description: Run scooter is hot, cold and wet environments			
Setup: Build final version of scooter thats ready to be ridden			
Step:	**Actions:**	**Expected Result:**	**Comments**:
1:	Use scooter in the Rain	Runs like normal, should not stall or fall apart. Should not short circuit.	
2:	Use scooter in freezing temperatures	Runs like normal, should not freeze up.	
3:	Use scooter in hot	Runes like normal, should not	

	temperatures	overheat and shut off.	
Overall test result:			

	S1	S2	S3	S4	S5	S6
H1	0	0	0	1	1	1
H2	0	1	1	1	0	0
H3	1	1	0	0	0	1





Software Implementation

We used a Atmega328p microprocessor to control our board. The Atmega328p features 6 hardware PWM channels, which makes controlling our motor circuit much more robust, rather than relying on the software.

In this project we used the ISP (In System Programmer) to program the chip. We first started using the Arduino to test our circuit. The code is below.

```
#define OFF 0
#define SEN A2
#include "TimerOne.h"



volatile int Speed = 170;
int isReady = 0;
int states = 7;
int count = 0;
volatile int noSpeedCount = 0;
double Time = 0;
unsigned long startTime = 0;
unsigned long endTime = 0;
int prevState = 7;
volatile int rpm = 0.0;


void setup() {
  Serial.begin(115200);
  pinMode(22, INPUT_PULLUP); //Yellow
  pinMode(24, INPUT_PULLUP); //Green
  pinMode(26, INPUT_PULLUP); //White

  pinMode(7, OUTPUT); //MOSI  Rp
  pinMode(6, OUTPUT); //5  Bp
  pinMode(5, OUTPUT); //4  Yn
  pinMode(4, OUTPUT); //3  Rn
  pinMode(3, OUTPUT); //2  Bn
  pinMode(2, OUTPUT); //1  Yp

  pinMode(A2, INPUT); //Throttle

  Timer1.initialize(50000);
  Timer1.attachInterrupt(defSpeed);
  startTime = millis();
  turnOff();
}

void defSpeed(){
  Speed = map(analogRead(SEN), 175, 869, 0, 150);
  //Speed = 75;
  if(Speed < 0)
    Speed = 0;

  ++noSpeedCount;
  if(noSpeedCount >= 5){
    rpm = 0;
    noSpeedCount = 0;
  }

}

void loop() {
  states = ((digitalRead(22) << 2) + (digitalRead(24) << 1) + digitalRead(26));

Serial.print(states);Serial.print('\t');Serial.print(Speed);Serial.print('\t');Serial.print(
rpm);Serial.print('\t');Serial.println(isReady);
  if(prevState != states){
     ++count;
    prevState = states;
  }

  if(count >= 45){
    endTime = millis();
    Time = endTime - startTime;
    Time = Time / 1000;
    rpm = 60/Time;
    count = 0;
    startTime = millis();
    noSpeedCount = 0;
  }

  if(isReady > 0){
   switch (states) {

   case 1:  //ify
     analogWrite(5, OFF);     //BN
     analogWrite(2, OFF);     //YP
     analogWrite(7, OFF);     //YN
     analogWrite(3, OFF);     //RP
     analogWrite(6, Speed);   //RN
     analogWrite(4, Speed);   //BP
     break;

   case 2: //GOOD FOR SURE
     analogWrite(6, OFF);   //RN
     analogWrite(7, OFF);   //YN
     analogWrite(4, OFF);   //BP
     analogWrite(3, OFF);   //RP
     analogWrite(5, Speed); //BN
     analogWrite(2, Speed); //YP
     break;

   case 3: //GOOD FOR SURE
     analogWrite(7, OFF);   //YN
     analogWrite(4, OFF);   //BP
     analogWrite(3, OFF);   //RP
     analogWrite(5, OFF);   //BN
     analogWrite(6, Speed); //RN
     analogWrite(2, Speed); //YP
     break;

   case 4:
     analogWrite(6, OFF);   //YN
     analogWrite(4, OFF);   //BP
     analogWrite(2, OFF);   //RP
     analogWrite(5, OFF);   //BN
     analogWrite(7, Speed); //RN
     analogWrite(3, Speed); //YP
     break;

   case 5:
     analogWrite(6, OFF);   //RN
     analogWrite(7, OFF);   //YN
     analogWrite(4, OFF);   //BP
     analogWrite(3, OFF);   //RP
     analogWrite(7, Speed); //BN
     analogWrite(4, Speed); //YP
     break;
    case 6://good to self iffy
      analogWrite(6, OFF);   //RN
      analogWrite(7, OFF);   //YN
      analogWrite(4, OFF);   //BP
      analogWrite(2, OFF); //YP
      analogWrite(5, Speed); //BN
      analogWrite(3, Speed);   //RP
      break;

    case 7:
      turnOff();
      break;
     }
     //delay(10);
    //--isReady;
   }
   else{
     turnOff();
   }
 }

 void turnOff(){
   digitalWrite(2, LOW);
   digitalWrite(3, LOW);
   digitalWrite(4, LOW);
   digitalWrite(5, LOW);
   digitalWrite(6, LOW);
   digitalWrite(7, LOW);
 }

 void serialEvent(){
   while(Serial.available()) {
     char inChar = (char)Serial.read();
     if(inChar == ' '){
       isReady = 10000;
       Serial.println("Going ");
       return;
     }
   }
   return;
 }
```

The Arduino code works by using simple case statements to determine what state the motor currently is in. To know what case the motor is on, we use a hall effect sensors that are built into the motor. We use the arduino digital pins to read in the bits and determine what state the motor is in. From there the case statements take over and depending on what the state it, the arduino will send logic highs on certain pins which will power the correct gates on the power MOSFETs. This will drive our motor to the next state. This process repeats over and over thousands of times a second. This is how our basic code works to drive our motor. We than incorporate our throttle which maps a analog value to 0 - 255 which will be our PWM duty cycle. This allows us to control the speed of the motor.

The next step was programming it in the microprocessor. We decided not to use a Arduino bootloader but just load our program on the microprocessor barebones without any bootloader. To program the microprocessor we used AVR dragon which has an ISP port to allow us to program the processor.

The main IDE used was Atmel Studio 7, which is based off of Microsoft Visual Studios. This IDE is very similar to Visual studios which makes it much easier for users that are familiar with Visual Studios.

Below is the code for the AVR which is based off our Arduino code.

```
#define F_CPU 16000000UL         //Says to the compiler which is our clock frequency, permits the delay
functions to be very accurate
#include <avr/io.h>              //General definitions of the registers values
#include <util/delay.h>          //This is where the delay functions are located
#include <avr/interrupt.h>

#define FOSC 16000000            // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD -1
#define OFF 255;

volatile int counts = 0;
volatile int counts2 = 0;

volatile uint8_t ADCvalue;
volatile long Speed = 255;
uint8_t  states = 7;
volatile int duration = 0;

#define pin7 OCR0B
#define pin6 OCR0A
#define pin5 OCR1A
#define pin4 OCR1B
#define pin3 OCR2A
#define pin2 OCR2B

#define Freq 300


///////////////////////////////////////////////////////////////////////////////////////////////

void turnOff() {
       OCR0B = OFF; //PD5, pin 7
       OCR0A = OFF; //PD6, pin 6
       OCR1A = OFF; //PB1, pin 5
       OCR1B = OFF; //PB2, pin 4
       OCR2A = OFF; //PB3, pin 3
       OCR2B = OFF; //PD3, pin 2
}

long mapp(uint8_t x, long in_min, long in_max, long out_min, long out_max) {
        return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}


ISR(TIMER1_OVF_vect)
{
        counts++;
        if ( counts > Freq) { //control freq.

                //this controls ON time
                if(duration == 0) {
                        Speed = OFF;
                }
                else {
                        Speed = 1;
                }
                counts = 0;
                counts2 = 0;
        }
        if ( counts2 >= duration) {
                Speed = OFF;
        }
        ++ counts2;
}



int main(void)
{

        int isReady = 1;

        //SET pin 2-7 as OUTPUT
        DDRB |= (1 << DDB1) | (1 << DDB2) | (1 << DDB3);   // sets port 3,4,5 as
        DDRD |= (1 << DDD3) | (1 << DDD5) | (1 << DDD6);    // pins 2,6,7 as output

        //TIMER0
        TCCR0A |= (1 << COM0A1) | (1 << COM0A0)| (1 << COM0B1) | (1 << COM0B0);  // set inverting mode
        TCCR0A |= (1 << WGM01) | (1 << WGM00);   // set fast PWM Mode
        TCCR0B |= (1 << CS00);                   // set pre-scaler to 8 and starts PWM

        //TIMER1
        TCCR1A |= (1 << COM1A1) | (1 << COM1B1) | (1 << COM1A0) | (1 << COM1B0); // set non-inverting mode
        TCCR1A |= (1 << WGM10);               // set fast PWM mode 8 bit
        TCCR1B |= (1 << WGM12);              // set fast PWM mode 8 bit
        TCCR1B |= (1 << CS10);               // START the timer with no pre-scaler

        //SET UP FOR TIMER2 PWM PINS
        TCCR2A |= (1 << COM2A1) | (1 << COM2B1) | (1 << COM2A0) | (1 << COM2B0);
       // set non - inverting mode for both lines
        TCCR2A |= (1 << WGM21) | (1 << WGM20);             // set fast PWM Mode
        TCCR2B |= (1 << CS20);                          // sets pre-scaler to 8 and starts PWM

        //SET UP FOR HALL EFFECT PINS
        DDRC &= ~(1 << DDC2) | ~(1 << DDC3) | ~(1 << DDC0);
       //sets PC2 & PC3 & PC0 as inputs for HALL_A, HALL_B, HALL_C
        PORTC |= (1 << PORTC2) | (1 << PORTC3) | (1 << PORTC0);
       //enable pull up resistor on PC2 & PC3 & PC0 for HALL_A, HALL_B, HALL_C

        ADMUX = 6;                 // use ADC6
        ADMUX |= (1 << REFS0);    // use AVcc as the reference
        ADMUX |= (1 << ADLAR);    // Right adjust for 8 bit resolution
        ADCSRA |=  (1 << ADPS1) | (1 << ADPS0); // 128 prescale for 16Mhz
        ADCSRA |= (1 << ADATE);   // Set ADC Auto Trigger Enable
        ADCSRB = 0;               // 0 for free running mode
        ADCSRA |= (1 << ADEN);    // Enable the ADC
        ADCSRA |= (1 << ADSC);    // Start the ADC conversion

        turnOff();

        //==================================//
        //TEST AREA//

        TIMSK1 = (1 << TOIE1);
        sei();


        while (1) {

                ////gets state of hall sensors
                states = (PINC & (1 << PC2)) + (((PINC & (1 << PC3))) >> 2 ) + (PINC & (1 << PC0));


                ADCvalue = ADCH;
                duration = mapp(ADCvalue, 70, 215, 0, Freq - 1);

                if (duration > Freq - 1)
                        duration = Freq - 1;
                else if (duration < 1)
                        duration = 0;



                if (isReady > 0) {

                        switch (states) {

                                case 1 :
                                pin5 = OFF;      //BN
                                pin2 = OFF;      //YP
                                pin7 = OFF;      //YN
                                pin3 = OFF;      //RP
                                pin6 = Speed;    //RN
                                pin4 = Speed;    //BP
                                break;

                                case 2 :
                                pin6 = OFF;    //RN
                                pin7 = OFF;    //YN
                                pin4 = OFF;    //BP
                                pin3 = OFF;    //RP
                                pin5 = Speed; //BN
                                pin2 = Speed; //YP
                                break;

                                case 3 :
                                pin7 = OFF;    //YN
                                pin4 = OFF;    //BP
                                pin3 = OFF;    //RP
                                pin5 = OFF;    //BN
```

```
                              pin6 = Speed; //RN
                              pin2 = Speed; //YP
                              break;

                              case 4 :
                              pin6 = OFF;   //YN
                              pin4 = OFF;   //BP
                              pin2 = OFF;   //RP
                              pin5 = OFF;   //BN
                              pin7 = Speed; //RN
                              pin3 = Speed; //YP
                              break;

                              case 5 :
                              pin6 = OFF;   //RN
                              pin7 = OFF;   //YN
                              pin4 = OFF;   //BP
                              pin3 = OFF;   //RP
                              pin7 = Speed; //BN
                              pin4 = Speed; //YP
                              break;

                              case 6 :
                              pin6 = OFF;   //RN
                              pin7 = OFF;   //YN
                              pin4 = OFF;   //BP
                              pin2 = OFF; //YP
                              pin5 = Speed; //BN
                              pin3 = Speed;   //RP
                              break;

                              default:
                              turnOff();
                              break;

                       }
               }
               //else
               //turnOff();
               //body of while loop
       }
}
```

To program the AVR we first had to change a fuse bit to allow us to set the system clock. Since the uP has a 8Mhz RC oscillator built it we needed to change the bits so that it will select the external clock. Our external clock is a 16MHz crystal oscillator. Changing the bits is relatively easy with Atmel Studio, we just needed to go into the program menu and select from a drop down menu for what clock our processor uses.

Now that our system clock is set we can start programing using the Arduino code as a base. The only hard part is setting up the timer which allows us to set the PWM signals off the pins on our processor. You have to go into the manual for the processor into the timer section and from there you can see the different options for setting the hardware pwm. You must set up 3 timers each with the same bit precision. I used the 8 bit precision which will give me a range from 0 - 255. We also made our PWM waves inverting because we wanted to make sure the motor driver will never be completely on because that will cause problems for the circuit.

Once the PWM pins are set you can follow down the code and see that we set up the pins for the Hall effect sensors and then set up the pins for the analog input which will read our analog values and map them from 0 - 255 which allows us to control the duty cycle of our PWM wave.

The rest of the code is the same algorithm as our arduino code; however this code implements an interrupt service routine.

While testing our code and using different frequencies to run our PWM waves at we tested 500Hz, 1khz, 4khz, 8khz, 32khz and 64khz. We found that the 64khz gave us the smoothest and fastest rotation of our motor. To get that 64khz we just made sure the PWM settings had no pre scalers and our fuse bits did not prescale our clock by a factor of 8. WHich left our system clock to a 16Mhz clock, and the timer works by counting to 256 and resets. So if we divide 16Mhz by 256 we get ~64khz.

One problem with running on 64khz was that the signal was too fast for our gate capacitors to charge up, causing a ramping of our pulse signals. Because of these ramp of our pulse signals, if our duty cycle was not on long enough the gate capacitors would not have enough time to charge up the correct logic value that would turn on the gates. The result of ramping signals made our throttle turn into an on and off switch, thus we lost all speed control.

To combat this ramping of the signal we came up with the idea to put the 64khz pulse signal on full throttle all the time and we would modulate the 64khz. Basically doing a PWM signal on top of another PWM signal. While testing with the new PWM signal we found out at around 300Hz was the best fit for our motor that gave smooth spinning as well as fine speed control.

With all of that we use the ISR to implement our “software pwm” the software pwm basically counted up and gave us a rate of 300Hz.

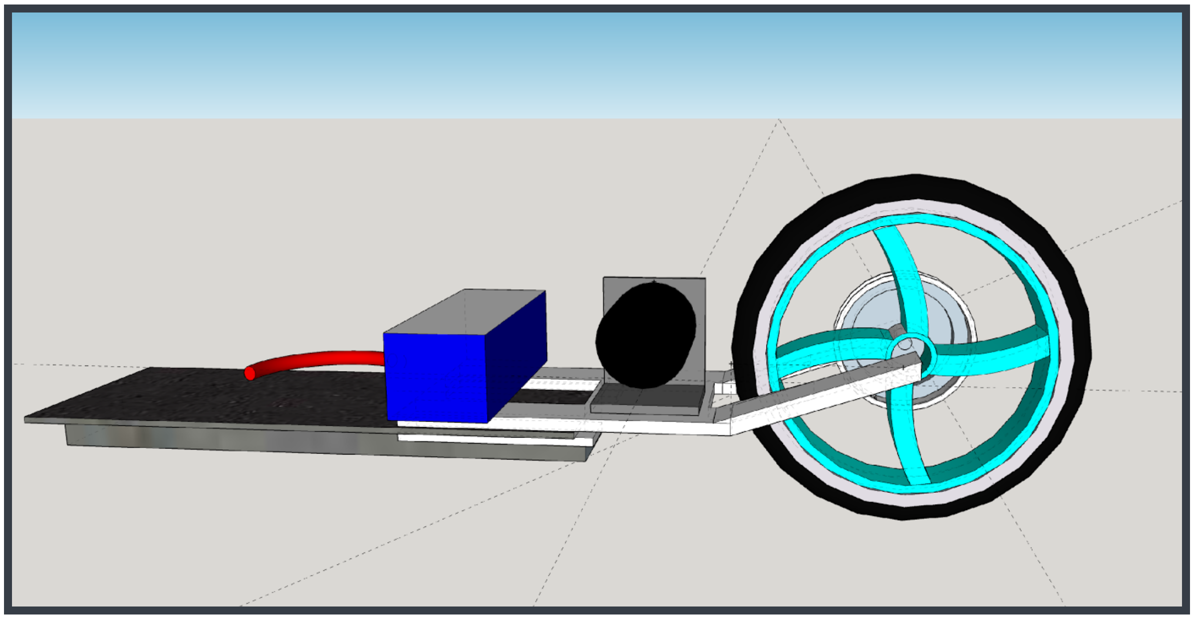
The software for the AVR is pretty self explanatory besides the pin assignments and pin activations which can all be found in the AVR manual for the Atmel 328p microprocessor.

Mechanical Design

One of the key differences between other practicum projects is the requirement being that the Scooter is fully rideable; this implies a frame and motor assembly must be built. Provided the motor controller properly powers the motor from the provided battery pack, It is up to the mechanical design to allow forward movement.

To begin, careful measurements were made on an existing NEXTSPORT scooter purchased online. This is a an adult kick scooter that is made of aluminum. Existing design was a deck with a flanged piece of aluminum that forked apart at the end to form the axle. This part would be chopped off. To compensate, our new mounting system must elevate the wheel to the point it was at previously for ride height and balance.

The design was first developed in Sketchup. Using all the measurements, we were able to visually design what we wanted our scooter to appear as, and took into consideration the ease of manufacturing such design.

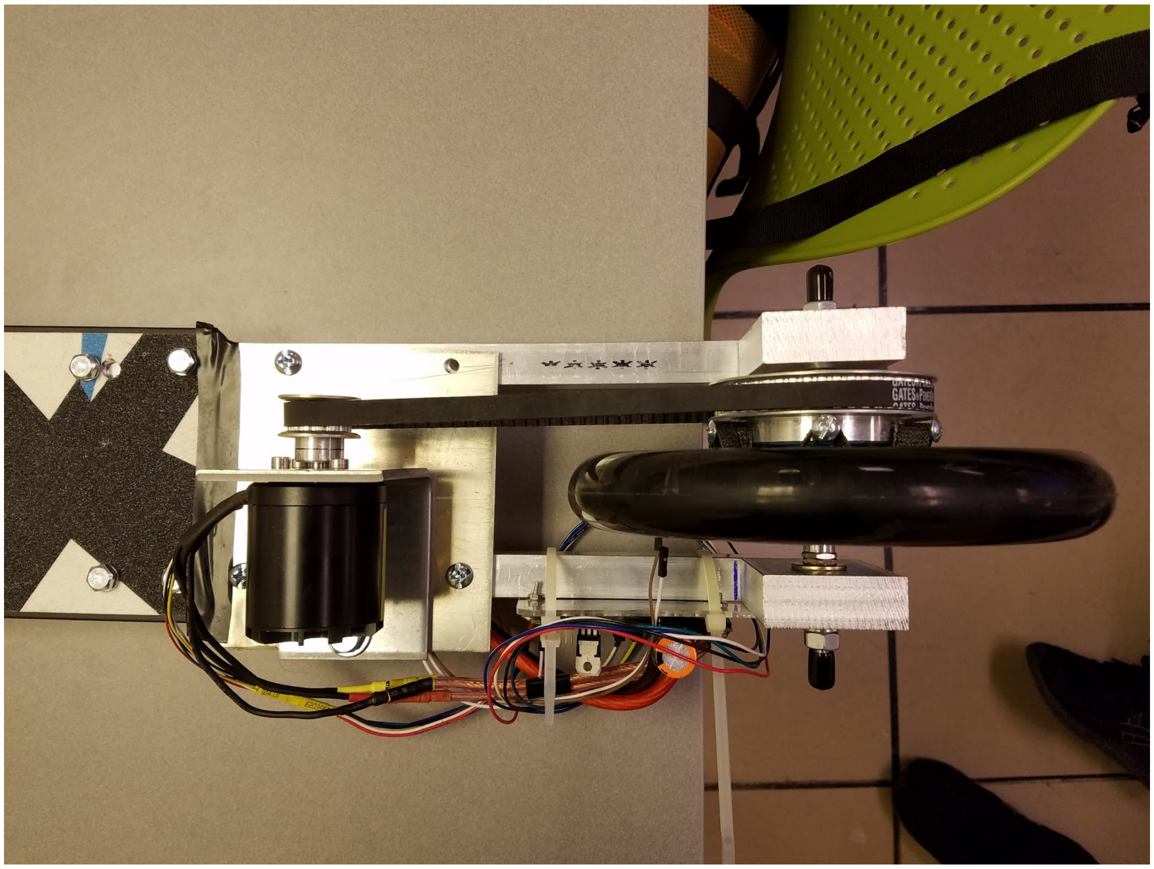


The design made in sketchup accounts for the wheel size and the deck size. Foot space was also an important consideration; it was decided to move the battery forward to the handlebars to give us more space. The motor bracket was found off the shelf as a 90 degree angle bracket. The Thick aluminum bars stayed, but they were mounted underneath the deck whilst a aluminum plate was bolted to the top, where the motor would mount to.

Since the belt drive system needs to be especially tight for it to function properly and deliver maximum power, we devised a system where the bracket had larger holes used to slide forward and back, the motor itself. THis way desired fit and alignment of the belt can be achieved. The belt itself was calculated based on a online formula, with the small gear having 15 teeth, and the large gear having 50 teeth. Measuring our desired length of about 22 centimeters gave us the belt size we needed.

The next issue was adhering the gear to the actual wheel. We needed no movement between the two, as the belt drove the gear which was attached to the wheel. Many options were explored, but the most structurally rigid that was achieved was by drilling five equidistant holes in the smooth portion of our gear. Threading them and attaching 8/32 machine screws, they served as pegs for a wire or strip to wrap around rodin coil style. We elected to use Fiberfix tape, similar to the material they make casts out of. After it hardened, it forms a strong bond. We like to consider this "Carbon fiber linked gear system".

The actual design differs slightly, as pictured below. But the basic premises are there. Slight modifications were made to ensure easier manufacturing. Some improvements that could be made are a better bearing system on the axle. It can either be too loose or to stiff, inducing friction.



Design Notes

- 10/17/1

To test out our knowledge of BLDC motor control theory, we're making a prototype PCB that should be able to spin the motor using NMOS 2n7000's to boost an Arduino's logic level to that of the IRFB4115's gate voltage. Note that in this current configuration, and because we didn't have any logic level PMOS devices, if the gate of the 2n7000 is left low or floating the power mosfets will fire causing short circuits to ground in the battery.

- 10/21/16

Starting design of the final board, questions for now:

- AREF has to be placed on a 22pF capacitor.
- IRFSXXXX are pretty good surface mount MOSFET's if the IRFB's take up too much space
- It appears that we will never be in a situation where we need to turn both mosfets on in a phase, so we might be able to half the number of pwm pins that we need depending on how the phase cycle goes, we will have to look into if this can be done safely.

-12/3/16

- Figured out we can mount the battery pack on the front as it is large enough.
- 22.2v battery connected backwards, discovered after partial fusing of wires upon attempt to connect. May have destroyed rectifiers.
- First test with a person, the motor arced and made smoke, board blown out. Wasn't even a load on it! Suspect may be internal damage from previous battery connect issues may be to blame

-12/5/16

- Used new voltage regulators to step down our voltage to 12V and run off of weaker 12.1 V battery. Motor still operates!