Design of a Sensorless Closed-Loop Speed Control Method for Brushed DC Cordless Power Tools

by

Daniel Ertl

A Report Submitted to the Faculty of the

Milwaukee School of Engineering

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Engineering

Milwaukee Wisconsin

February 2020

Abstract

This project final report contains the project background, specifications, results, and analysis for a Master of Science in Engineering (MSE) capstone design project. The purpose of the project was to design and implement sensorless speed detection in cordless power tools which are powered by a brushed DC motor (i.e., a permanent magnet DC motor, or PMDC). Project requirements included a speed detection operating range of 500 to 18,000RPM, and a steady-state error of 5%. The implementation needed to operate on 8-bit microcontrollers in high dynamic load conditions. The cost of the components utilized in the sensorless design needed to be less than 50% of the cost of the hall encoder components used in the existing design.

Traditional methods of motor speed detection in cordless power tool designs involve either a hall sensor and a ring magnet or a rotary encoder. Implementing a sensorless design would not only reduce the total size and complexity of the electronics hardware design, but also significantly reduce the overall cost of the electronics. This project report features a review of the relevant literature, design of various speed measurement methods, test results, and analysis of the performance of each method.

A new implementation method was developed for detecting motor speed by modifying and combining components of the three established methods for detecting motor speed: Back Electromotive Force (BEMF), Current pulse counting, and inductive BEMF spike measurement. Real world loading data were collected using a modified Milwaukee Tool M18 Multitool design. The original hall sensor and ring magnet encoder provided a reference for the actual speed that the new implementation was checked against.

Performance of the new sensorless speed detection method was compared to the original sensor-based implementation in controlled loading scenarios and real-world applications.

The BEMF method met all the project goals and requirements. The performance in the open-loop, closed-loop, and application testing all fell well below the targets. The cost associated with this method was only 0.8% of the original component cost. The BEMF inductive spike method was found to be technically feasible, but impractical because of the large amount of motor specific variable mapping required. The current ripple method was the most accurate algorithm and does not require the use of individual motor parameters, but the method is limited by the sampling rate and clock frequency of the microcontroller.

A significant result is that autocalibration was used to successfully combine the BEMF method and the current ripple method, such that the BEMF method could be automatically calibrated by the current ripple method.

The current ripple method could be used exclusively in microcontrollers with high clock speeds. As the performance of low-cost microcontrollers continues to increase, this method will likely become the best option for sensorless speed detection for power tool embedded systems.

Acknowledgments

I would like to thank my adviser Dr. Luke Weber for reviewing my work and providing valuable insight and improvement ideas throughout the project.

I would like to thank Professor Gary Shimek for reviewing my report format, grammar, sentence structure, and detailed proofreading comments.

I would like to thank Dr. Subha Kumpaty for facilitating the project process and reviewing my project each quarter.

Table of Contents

List of Figures

List of Tables

Nomenclature

Symbols

2p – The number of pole pairs

C_n – Capacitor Designators

D – Duty cycle

D_n – Diode designators

E_a – Motor BEMF

f_r – Ripple current frequency

i_a – Motor armature current

k – Number of commutator segments

K_n – Combined circuit constants used for curve fitting

K_v – Motor Constant $\left[\frac{RPM}{V \cdot s}\right]$

L_a – Motor armature inductance

n – Motor speed

R_a – Motor armature resistance

R_n – Resistance designators

s – Seconds

V – Volt

V_{ADC1} – Scaled negative motor terminal voltage

V_{ADC2} – Scaled battery voltage

V_{ADC3} – Voltage across current shunt resistor

V_d – Voltage drop across freewheeling diode

V_M – Voltage at the negative motor terminal

V_{PS} – Power supply or battery voltage

η – the largest common divisor of 2p and k

Abbreviations

ADC – Analog to Digital Converter

BEMF – Back Electromotive Force

BOM – Bill of Materials

CE – Conformité Européene (European Conformity)

DC – Direct Current

DUT – Device Under Test

FFT – Fast Fourier Transform

FPGA – Field Programmable Gate Array

I/O – Input/output

ISR – Interrupt Service Routine

MOSFET – Metal-Oxide-Semiconductor Field-Effect Transistor

PC – Personal Computer

PCB – Printed Circuit Board

PCBA – Printed Circuit Board Assembly

PI – Proportional Integral

PID – Proportional Integral Derivative

PLL – Phase Locked Loop

PMDC – Permanent Magnet Direct Current

PWM – Pulse Width Modulation

SLIP – Serial Line Internet Protocol

SPDT – Signal Pull Double Throw

SPST – Single Pull Single Throw

UART – Universal Asynchronous Receiver/Transmitter

UL – Underwriters Laboratory

Chapter 1: Introduction and Background

1.0 Introduction

1.0.1 Overview

The transformation of the power tool industry from corded and gasoline-powered designs began in 2007 when Milwaukee Tool filed a patent application for a lithium-ion based battery powered power tool [1]. Up until this point, cordless power tools were considered underpowered, unreliable and only useful for small tasks. Since the change to the higher power, longer-life lithium ion batteries starting in 2007, the cordless power tool market has expanded to take over most traditionally corded power tool and some gas power tool categories [2].

The first lithium ion cordless power tools were based on brushed DC motor designs. Since then, the market has shifted to more power dense and efficient (but more costly) brushless DC motors for many of the premium high-performance products [3]. However, many entry-level and specialty tool designs still feature brushed DC motors. In fact, 40% of the professional grade cordless power tool market is still brushed [4].

Many specialty tools that use brushed DC motors, such as multi-tools and random orbit sanders, need to maintain a constant speed as the load changes to operate correctly. Additionally, drilling applications, such as hole-saws or step-bits, have set RPM values that are optimal for clean cutting and longevity of the life of the bit. Sensing the speed of the motor is critical to this functionality.

Speed sensing in power tools that utilize brushed DC motors has traditionally been done using position encoders or ring-magnets with a hall

sensor. This method is easily implemented but has two major drawbacks. First is size – the ring magnet or encoder must be placed on the rotor, which makes the tool larger. Secondly, the additional components (encoder or ring magnet and hall sensor) increase the cost of the design.

Sensorless speed detection methods would eliminate the need for external components and would only utilize ADC channels already present on the microcontrollers that are used to control the electronics system. Sensorless speed detection reduces size and cost without the need for any additional components.

1.0.2 Background

Milwaukee Tool has used sensor-based speed detection methods exclusively for speed detection of brushed DC designs [5]. Speed detection is an important component of designs that require closed-loop output speed control. There are a variety of reasons a design may require closed-loop speed control.

Maintaining optimal cutting blade speed on cutting tools, such as chainsaws or reciprocating saws, is important to achieving a consistent cut. Users apply different amounts of force to the tool while cutting. Closed-loop control of the output speed allows the tool to detect higher or lower force and adjust the speed of the tool accordingly. As long as the applied force doesn't result in a saturation of the control system, the optimal cutting speed will be maintained.

Another common application of closed-loop speed control is to maintain consistent tool performance over a battery discharge cycle. As lithium-ion cells discharge, the cell voltage drops from 4.2V (fully charged) to about 2.7V

(discharged). In a five-cell series connected battery, this means that the voltage will drop from 21V at full charge to 13.5V at end-of-discharge. On impact wrenches, string trimmers, and other tools where consistent tool performance is more important than peak performance at a full charge, closed-loop speed control is used to limit the performance of the tool at full charge such that the performance is consistent across the entire, or a significant portion, of the battery discharge curve.

The third common application of speed-detection in brushed DC tools is the adaptive feature category. An adaptive feature is a feature that performs a function or task automatically using various inputs (often including motor speed, position, or acceleration). On impact drivers, the self-tapping screw and concrete anchor adaptive features utilize the motor speed along with several other inputs to detect when the screw or anchor head has seated. This mode shuts down the tool automatically to prevent the fastener from stripping the threads. On rivet tools when an application completes, the motor needs to reverse in order to reset the mechanism. Motor position is used to determine the location of the mechanism and reset it to the appropriate starting location for the next rivet.

Closed-loop speed control can also be used to optimize accessory performance. Accessories such as hole saws and step bits have optimal speeds at which they will cut most effectively and at which the bits will stay sharp for more total applications. Closed-loop speed control can be used to automatically control the speed to keep the accessory in its optimal speed range.

Sensorless detection of motor speed is not a new concept but it has not been evaluated for use in power tools by Milwaukee Tool. Additionally, there do not appear to be any patents for utilizing these methods in power tool applications. The idea for this project came from the Milwaukee School of Engineering Specialty Electric Machines course where part of the requirement was to model the current ripple of brushed DC motors. There is a relationship between current ripple frequency and the motor speed. The initial idea was to try to use this ripple frequency to detect the motor speed. However, this report proposes a new combined approach that utilizes several methods to determine the speed of the motor.

1.0.3 Description of the Project

The optimal method or combination of methods for sensorlessly determining the speed of a brushed DC motor in a power tool application needed to be determined. In order to facilitate that process, in this project, the circuit board for the M18 Multitool model number 2626-20 was modified. The Multitool was chosen because it supports a wide range of applications covering cutting and sanding applications that require closed-loop speed control. The existing microcontroller has been replaced with a similar Atmega328, which has more I/O and a UART interface for reading out data to a PC or laptop.

The existing design has a hall sensor and magnet-based encoder. The magnet has four poles. In one rotation, the hall sensor will change states four times. This encoder is used as the reference that the new implementations have

been compared against. Circuits have also been added to measure the current across a shunt resistor through a current sense amplifier circuit, and voltage dividers were used to shift the battery and motor voltages down to within the range of the ADC on the ATMega328 microcontroller. The ATMega328 was chosen because it is an 8-bit microcontroller very similar to the controllers in the original system, but it also supports UART communication which is useful for allowing the tool to communicate to a PC and log data. Milwaukee Tool proprietary circuits that are not relevant to this project have been grayed out with the function purpose of the circuit listed. Figure 1 shows the proposed starting schematic for this project. The shaded regions are other circuits from the original product design and are not relevant to the project.

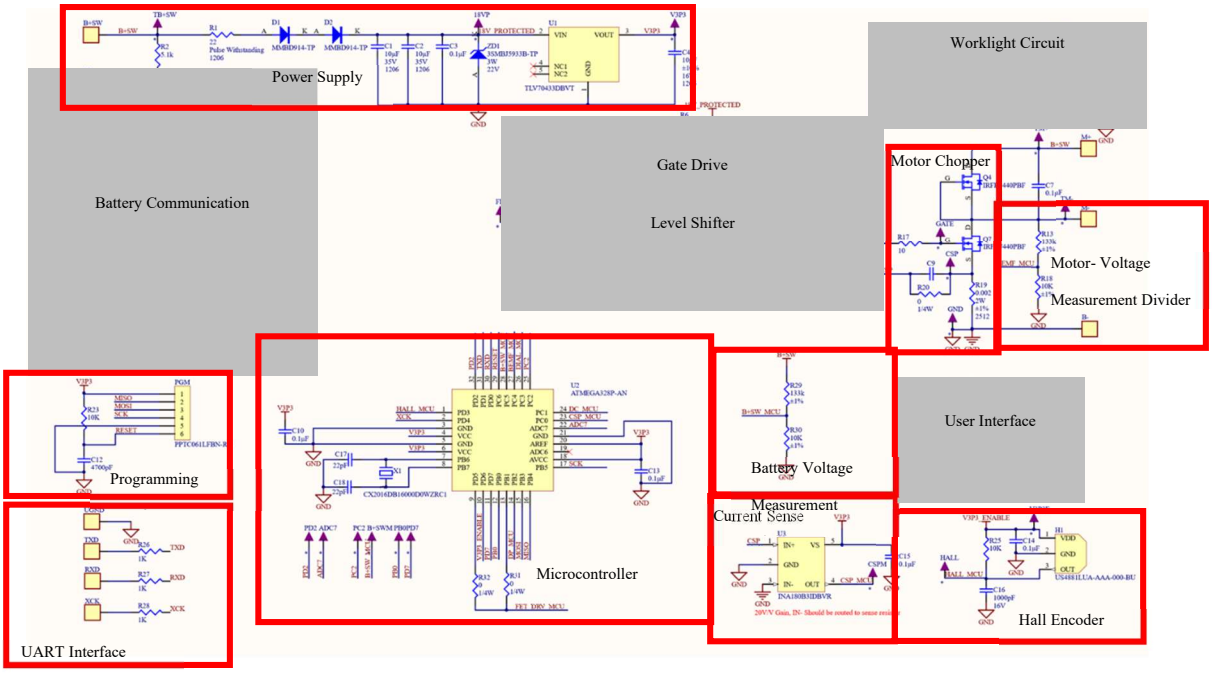

Figure 1: Updated Schematic for M18 Multitool.

The updated layout fits in the original tool and has the same outline. In order to accommodate all the changes, the board was changed from two layers to four layers. All major components, such as the MOSFETs, heatsink, wire locations, and speed dial potentiometer, were left in the original locations. Figure 2 shows the layout and three-dimensional rendering of the updated circuit board according to the schematic in Figure 1.

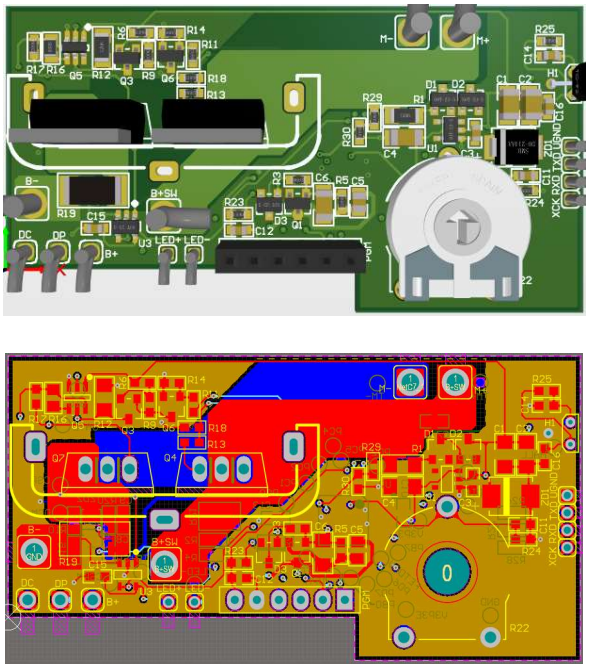

Figure 2: Updated Layout for 2626-20 Circuit Board.

This updated platform has custom firmware written to test and compare different speed detection methods. This report details the investigation of available sensorless speed measurement methods and selection of the most suitable method for brushed DC cordless power tools. Hardware was designed, and software algorithms were written to sensorlessly determine the motor speed. Full 2626-20 electronics have been replaced with new electronics that implement

sensorless control while maintaining the original sensor-based speed measurement method for comparison. Data were collected with the new sensorless method in real applications. Performance and accuracy in application against the original method were compared.

1.0.4 Justification of the Project

Forty percent of Milwaukee Tool's cordless power tool business is still brushed [4]. Of those tools, about 20% already have sensor-based closed-loop speed control (sanders, multitools, grease and caulk guns). The current Bill of Materials (BOM) for the sensor-based speed detection components (magnet, hall sensor and related resistors and capacitors) is broken out in Table 1.

Table 1: BOM Cost of Sensor-Based Speed Measurement Components [5].

Component	Unit Cost
Hall Sensor	$0.36
Ring Magnet	$0.95
Resistors	$0.001
Capacitors	$0.03
Total	**$1.34**

Brushed tools are often entry-level tools with very low margin. Reducing the cost of these designs can either allow the product to be priced lower than those of competitors or increase the margin at an existing price point. Eliminating the speed sensing components in a typical brushed design would save around 20% of the total electronics cost in a design. A conservative estimate for the annual savings of implementing a sensorless based design is $268,000. This estimate is based on the total component cost savings from Table 1 and the annual volume of 200,000 units (volume for existing Milwaukee Tool sensor-based designs) [5].

In addition to cost savings on existing designs, the sensorless speed measurement method allows for new innovations in products where the addition of the sensor components previously were cost prohibitive. Features that had previously only been available on brushless tools – such as hole-saw control on drills and TEK screw auto-seating, hand-tight, bolt-removal, and wrench-tight modes on impacts – are now possible on brushed designs without significant cost additions to the existing design.

1.0.5 Project Specifications and Goals

This project focused on the application of sensorless speed control methods to PMDC motor-based cordless power tools. The primary research goals were:

1. To implement a sensorless speed control in a brushed DC motor power tool.
2. The speed control sensing shall function in a range of 500 RPM – 18,000 RPM motor speeds
3. The system shall have a steady-state error of 5% or less (full-scale value) across the operational speed range. For a motor with a maximum speed of 20,000 RPM, the allowable steady-state error is:

$$FSE = 20{,}000[RPM] \times 0.05 = 1{,}000\ [RPM].$$

    a. A stretch goal for the error is 5% of the speed target at steady-state.

4. The implementation shall be compatible with similar or existing microcontrollers already used in the system (8-bit microcontrollers such as the ATMEGA328)
    a. The system will not feature high-performance microcontrollers or Field Programable Gate Arrays (FPGAs) that have been employed by previous researchers [6].
5. The system must work with dynamically varying loads on the power tool motor (drilling, sanding, and cutting applications).
6. The total component cost for the additional sensorless components must be less than 50% of the original sensor-based method cost.
7. The total size of the system printed circuit board (PCB) outline must be the same as the sensor-based design and the PCB must fit into the existing mechanical mounting system.

1.0.6 Existing Technology

Texas Instruments has a design for a ripple-current-based position detection system for use in automotive applications. This design is intended for brushed motors that power windows, sliding-doors, mirrors and liftgates [7]. These applications are more focused on position than speed, but the methods used to determine either speed or position are the same. The primary difference between the technology developed by Texas Instruments [7] and the proposed project is that the Texas Instruments technology focuses on applications that are under constant loads at low rotational speeds. Moreover, the Texas Instruments

method is limited to a ripple frequency of 1.3kHz [7] and requires significant hardware signal conditioning. In contrast, this project focused on highly dynamic loads and high ripple frequencies (high speed) with only limited hardware signal conditioning.

In Vejlupek, Grepl, Matejasko, and Zouhar [8], a current ripple method is described for detecting faults in automotive fuel pumps in a manufacturing setting utilizing a Fast Fourier Transform (FFT) of the current ripple to determine if the pump is spinning at the appropriate speed. This method employs an FFT of the current ripple over a two-second sample time and then sets bounds, where if the FFT falls outside the bounds, then the part will be labeled as failed during manufacturing. Vejlupek *et al.*'s method [8] is very different from the design detailed in this report, because this project requires more precise speed information, which is used to update a PID controller that will control the speed of the tool at an update rate of about 1ms.

Finally, in a patent application for a brushed DC bilge pump, direct BEMF measurement was used to estimate the motor speed [9]. This method employed a low-side switching element. In this method, the motor is run at 100% duty cycle. Periodically, the motor switched off so that the BEMF measurement can be made. Once the measurement has been completed, the motor is returned to 100% duty cycle. The motor speed is estimated from the BEMF. If the motor speed reaches a critical limit, the pump can be shut down to prevent damage to the system. The purpose of this design is to detect large changes in motor speed at a constant duty

cycle for overload purposes. In contrast, the method proposed in this document controls the motor speed directly using the calculated motor speed as the feedback for that control system.

1.1 Sensorless Speed Detection Theory

There are three common methods for sensorless speed detection: Back Electromotive Force (BEMF) amplitude, inductive spike amplitude, and current pulse frequency. Each of these three methods are described in this section.

1.1.0 BEMF Amplitude Speed Measurement

By far, the most straightforward method of inferring motor angular velocity is the BEMF measurement method, which uses the relationship between angular velocity and back EMF of PMDC motors. According to Precision Microdrives [10], this is

$$RPM = BEMF \cdot K_v, \tag{1}$$

where K_v is the motor constant and RPM is the angular speed in revolutions per minute. Although the physics behind the speed/BEMF relationship is straightforward, the theory behind the implementation is more involved. The first complication comes from the fact that most PMDC motor-based cordless power tools use low-side switched chopper circuits. This means that the BEMF is not directly measurable by a microcontroller with a single measurement.

The solution to this problem is – instead of measuring the BEMF directly – to measure the negative side of the motor when the Pulse Width Modulation (PWM) is low and the BEMF can be calculated from that value. The circuit in

Figure 3 demonstrates the type of measurement method proposed by Precision Microdrives in Application Note AB-021 [10] and by Kumar and Radcliffe [11].

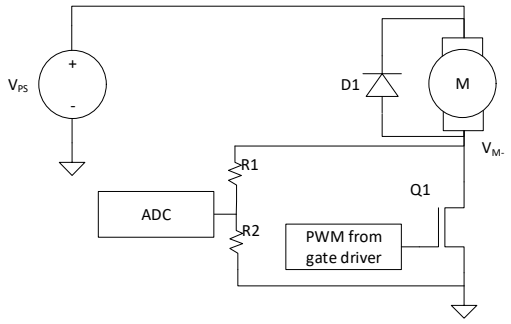

Figure 3: V_M Measurement Circuit.

In this low-side switched design, when Q_1 is off, the negative connection to the motor will be pulled up to the BEMF voltage. The applied voltage to the motor is much higher than the rated voltage of the microcontroller Analog-to-Digital Converter (ADC) pins, so a resistor divider is used to scale the voltage. The equation for the measured voltage at the ADC is:

$$V_{ADC1} = V_M \cdot \frac{R_2}{R_1 + R_2}. \tag{2}$$

Solving for V_M:

$$V_M = V_{ADC1} \frac{(R_1 + R_2)}{R_2}. \tag{3}$$

The motor does not have zero armature resistance and therefore will have a voltage drop across the resistance based on the current going through the motor. Figure 4 demonstrates the same circuit as before but with the motor block

replaced with the PMDC motor equivalent circuit, where E_a is the BEMF voltage, R_a is the armature resistance, and L_a is the armature inductance described in Application Note AB-021 by Precision Microdrives [10] and Kumar and Radcliffe [11].

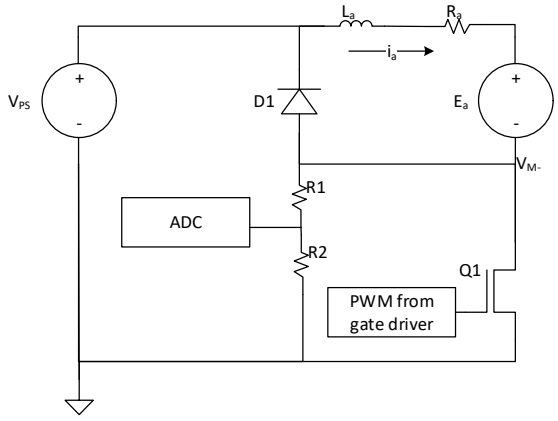

Figure 4: Motor Block Replaced with PMDC Motor Equivalent Circuit.

The equation for the BEMF, according to Precision Microdrives [10] and Kumar and Radcliffe [11], is

$$E_a = V_{PS} - V_M - i_a R_a - L_a \frac{di_a}{dt}. \quad (4)$$

Cordless Power Tools using PMDC motors with armature resistance of 50mΩ can draw more than 40A in application. Therefore, the value of $i_a R_a$ is significant (>2V) and should not be ignored. The inductive term depends on how quickly the load is expected to change. If the load is constant or changes slowly, then there is no voltage drop due to the inductive term. If the load changes quickly, then the inductive term may be important.

The next issue is that voltage of the battery changes over the discharge cycle and will also change based on the applied load due to the internal resistance of the battery cells. An additional measurement circuit can be used to measure the actual terminal voltage of the motor. Therefore, it is necessary to know the positive terminal voltage in addition to the negative terminal voltage of the motor to achieve an accurate measurement. Figure 5 shows a more accurate method of measuring the BEMF by using a second ADC for the V_{PS} voltage measurement and a third ADC for the current measurement.

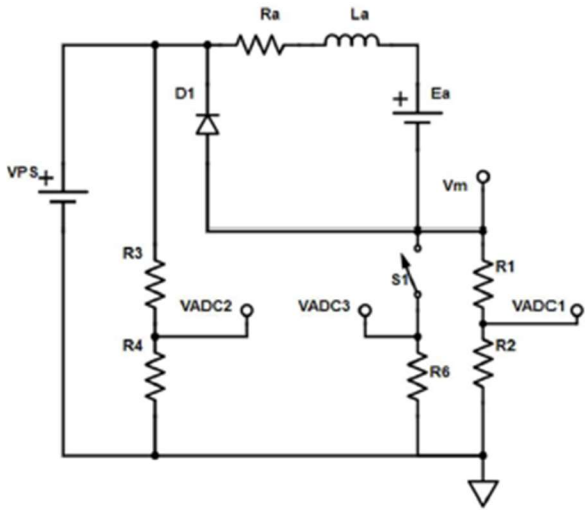

Figure 5: More Accurate Method of BEMF Measurement.

The voltage drops due to L_a and R_a require the motor current to be known. Because the measurements for V_{M-} and V_{PS} will be taken right after Q_1 turns off, measuring i_a right before Q_1 turns off is a good approximation for the motor current right after Q_1 turns off.

Measuring current in this manner requires only one additional ADC channel be used and a very simple calculation can be done to find i_a. Thus,

$$V_{ADC3} = i_a R_5. \tag{5}$$

Rearranging yields the equation

$$i_a = \frac{V_{ADC3}}{R_5}. \tag{6}$$

The battery voltage V_{PS} can be found in terms of V_{ADC2}; that is,

$$V_{ADC2} = V_{PS} \cdot \frac{R_4}{R_3 + R_4}. \tag{7}$$

Solving Equation (7) for the battery voltage yields the equation

$$V_{PS} = \frac{V_{ADC2}(R_3 + R_4)}{R_4}. \tag{8}$$

The BEMF voltage is given by

$$E_a = V_{PS} - V_M - i_a R_a - L_a \frac{di_a}{dt}. \tag{9}$$

Substituting in V_{ADC} equations and setting $R_4 = R_2$ and $R_3 = R_1$ yields

$$E_a = \frac{(R_1 + R_2)}{R_2}(V_{ADC2} - V_{ADC1}) - \frac{V_{ADC3}}{R_5} R_a - \frac{d}{dt}\left(\frac{V_{ADC3}}{R_5}\right) L_a. \tag{10}$$

RPM is calculated as the BEMF multiplied by the motor constant,

$$RPM = \left[\frac{(R_1 + R_2)}{R_2}(V_{ADC2} - V_{ADC1}) - \frac{V_{ADC3}}{R_5}R_a - \frac{d}{dt}\left(\frac{V_{ADC3}}{R_5}\right)L_a\right]K_v. \quad (11)$$

This final equation is now written in terms of the measured ADC values ($V_{ADC1}, V_{ADC2}, \text{and } V_{ADC3}$) and motor constants ($R_a, L_a$ and K_v). This can be directly used along with the final hardware design to estimate the RPM of the motor.

1.1.1 Inductive Spike Amplitude Speed Measurement

Radcliffe and Kumar [11] proposed two new methods for measuring the speed of a PMDC motor by utilizing two properties of the inductive spike that occur when the low-side chopper is turned off during PWM. The first method evaluates the rise time of the spike, whereas the second method measures the duration of the spike. Their methods, along with some modifications for operation in a battery-powered system, are summarized in this section.

1.1.1.0 Inductive Spike Duration Method

Similar to the BEMF amplitude method, this speed detection design relies on the measurement of the low side terminal voltage of the PMDC motor. The circuit that Radcliffe and Kumar used is shown in Figure 6.

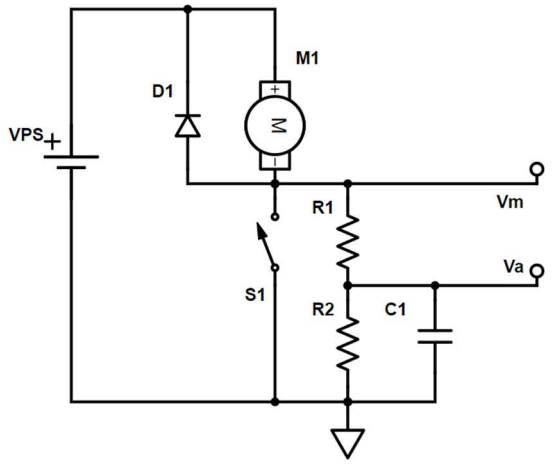

Figure 6: Measurement Setup for Negative Motor Terminal [11].

In this circuit, V_m is the motor negative terminal voltage, and V_a is the voltage at the ADC pin of the microcontroller. A resistor divider is used in the same manner as in the BEMF method, to scale the terminal voltage down to a range that is within the rating of the ADC pin of the microcontroller. In this design, the authors recommended C1 be added to create a low pass filter; however, later in the paper, they recommend removing it for a more accurate measurement of pulse width. The second recommendation was followed for the purposes of this project.

Because a battery is used, the same two modifications that were made to the BEMF circuit were also made here so that the battery voltage can be measured using a second ADC and current can be measured using a shunt resistor and a third ADC. C1 is removed to achieve the most accurate measurement of the spike duration (any filtering needed can be added in software). The circuit voltages

were also renamed to match the naming convention of this document. These circuit modifications are shown in Figure 7.

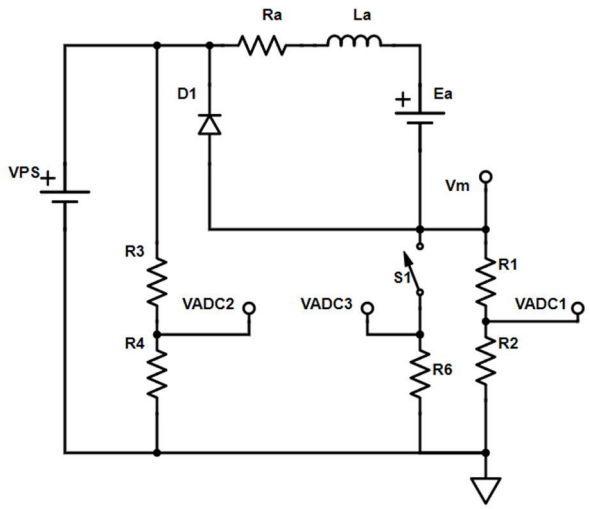

Figure 7: Updated Circuit Diagram.

The following derivation closely follows what Radcliff and Kumar proposed [11]. Several modifications to the equations have been made to present the equations in terms of the ADC voltages present in the modified circuit, but the concept is the same as in the work of Radcliff and Kumar [11].

At steady-state, assuming the switch is closed, according to Radcliffe and Kumar [11], the battery voltage is given by

$$V_{PS} = i_a R_a + E_a + i_a R_6. \tag{12}$$

Per Radcliffe and Kumar [11], at steady-state assuming the switch is open:

$$V_{PS} = i_a R_a + E_a + V_d, \tag{13}$$

where V_d is the voltage drop across diode D_1.

The speed of the motor is proportional to the BEMF voltage Ea, therefore, by rearranging, it is found that

$$Speed\ \alpha\ V_{PS} - i_a R_a - i_a R_6. \tag{14}$$

Set

$$R_2 = R_4\ and\ R_1 = R_3. \tag{15}$$

The equation for V_{ADC1} is

$$V_{ADC1} = \frac{R_2}{R_1 + R_2} V_m. \tag{16}$$

The equation for V_{ADC2} *is*

$$V_{ADC2} = \frac{R_2}{R_1 + R_2} V_{PS}. \tag{17}$$

The relationship between the measurable values V_{ADC1} , V_{ADC2} and E_a (ignoring R_a and L_a for the moment) is

$$V_{ADC1}\frac{R_1 + R_2}{R_2} = D \cdot (V_{on} + i_a R_6) + (1 - D)\left[\frac{R_1 + R_2}{R_2} V_{ADC2} - E_a\right], \tag{18}$$

where D is the duty cycle and V_{on} is the voltage drop across the switch.

Note: in Radcliffe and Kumar [11], the equivalent to Equation (18) incorrectly states that the scaling is $\frac{R_1+R_2}{R_1}$. This has been corrected here.

When the switch is turned off, the current flowing through the motor inductance will create a voltage spike corresponding to the current and motor armature inductance. When this happens, E_a will exceed V_{PS}. This effect is shown in Figure 8.

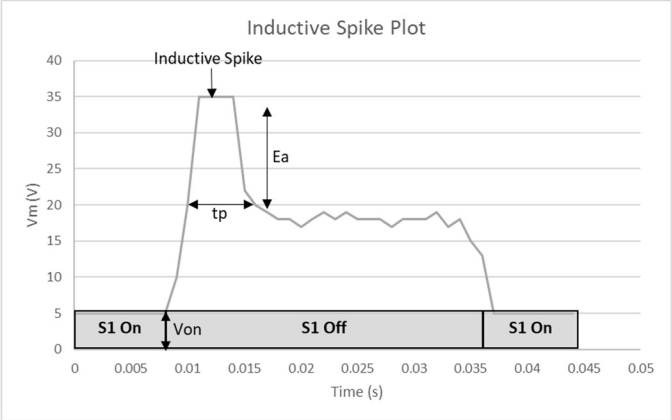

Figure 8: Inductive Spike After S1 Turn Off Event [11].

According to Radcliffe and Kumar [11], the energy lost by the inductor over time δt is:

$$E_L = 0.5L_a i_a^2 - 0.5(i_a + \delta i_a)^2 = L_a i_a \delta i_a, \tag{19}$$

$$L_a di = i_a^2 R_a \delta t + V_d i_a \delta t. \tag{20}$$

Integrating:

$$I = K_1 e^{-\frac{tR_a}{L_a}} - V_d R_a, \tag{21}$$

where $K_1, K_2, \ldots, K_n$ are constants.

Because the inductive spike is related to the current right at the time the switch is turned off, $i_a = i_o$, and when the inductive spike ends, the current stops flowing ($i_a = 0$). Thus, the equation can be written as:

$$i_o = K_2 + K_3 e^{t_p K_4}, \tag{22}$$

where $K_2, K_3, and\ K_4$ are constants and t_p is the pulse duration.

The applied voltage to the motor is proportional to the PWM duty cycle, and is

$$V_{Motor} = D \cdot V_{PS}. \tag{23}$$

Combining equations and solving for speed yields

$$Speed = K_5 + K_6 \cdot D + K_7 e^{t_p K_8}. \quad (24)$$

Rather than calculate $K_5, K_6,$ and K_7, the authors recommend collecting experimental data and curve fitting the constants. Curve fitting the constants will consider more of the non-ideal behavior of the motor than trying to calculate what the constants would be able to achieve.

1.1.1.1 Inductive Spike Rise Time Method

This method requires an RC snubber to be added across the motor. Without this snubber, the rise time would be too fast for typical ADCs operating at less than 200kHz to detect. Because adding an RC snubber circuit of sufficient size would unnecessarily add cost to the cordless power tool design, this method is not practical for this use case.

The equation that Radcliffe and Kumar developed for this method is similar to Equation (24), except it relies on pulse rise time instead of pulse duration. This equation is

$$Speed = K_{18} + K_{19} \cdot D + K_{20} e^{t_r K_{21}}, \quad (25)$$

where $K_{18}, K_{19},$ and K_{20} are constants and t_r is the inductive pulse rise time [11].

1.1.2 Current Pulse Frequency Speed Measurement

The methods for measuring speed discussed up until this point require that the duty cycle of the switching element be less than 100%. In practical systems, this is undesirable for several reasons. Constantly switching a device like a MOSFET or IGBT creates a lot of heat from the switching events. This can add

cost to a design, as a better part may have to be used to compensate for the additional heat generation. The second reason is that always operating below 100% duty cycle is undesirable because it limits system performance. The maximum duty cycle of the inductive spike method must allow the switch to be off for the maximum inductive pulse duration. In the case of BEMF measurement, the off time must be longer than the pulse duration and settling time for the V_{ADC1} voltage to reach a stable value.

This is where the current pulse frequency method shows its advantages. This method relies on the relationship between current pulses and speed. According to Vasquez-Sanchez, Sottile, and Gomez-Gil [12], this relationship is

$$f_r = \frac{2p \cdot k \cdot n}{60 \cdot \eta}, \tag{26}$$

where $2p$ is the number of pole pairs, f_r is the ripple frequency, k is the number of commutator segments, n is the motor speed, and η is the largest common divisor of $2p$ and k.

An example of current ripple on a PMDC motor is shown in Figure 9.

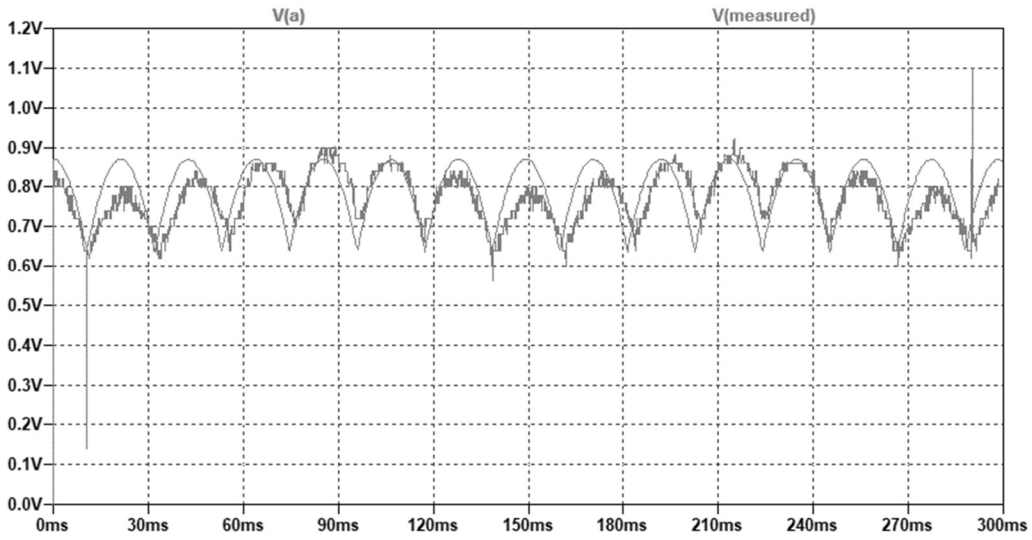

Figure 9: Current Ripple of a PMDC Motor.

Because this method relies on a current measurement, there is no need to run the motor at less than 100% duty cycle. However, because current is generally a very noisy signal, and changes in load or inrush events during startup may be interpreted as ripple by an algorithm, most methods require significant processing power and overhead to accurately measure the ripple frequency.

1.2 Literature Review

Attempts to detect the speed of a motor in a sensorless manner, and reported in the literature, usually utilize one of the three methods discussed in the previous section on the theory associated with sensorless speed detection. The three methods include Current Ripple, BEMF Amplitude, Inductive Spike Duration or Rise Time. The approaches reported in the literature vary widely in terms of implementation. For each general category, specific implementations reported in the literature are discussed.

1.2.1 Current Ripple Methods

Vazquez-Sanchez, Sottile, and Gomez-Gil [12] and Khoo, Mariappon, and Saad [13] both use the principal underlying Equation (26) that current ripple is related to the speed of the motor. Khoo *et al*. [13] utilize a preamplification and then a Sallen-Key high-pass filter to hardware condition the current measurement signal. After amplification, the signal is fed into an artificial neural network that evaluates at the width and height of the current ripple pulses. Khoo *et al*. [13] were able to achieve a speed measurement error of between 0.18% and 0.41%. This method does have the significant limitation of only focusing on controlled startup events where there is a lot of signal and no external load variation. The motor was powered by a power supply in very specific conditions that may have led to more accurate results than what might be found in a real-world application. The researchers also did not specify the maximum angular velocity at which this method would be effective.

In their research, Vasquez-Sanchez *et al*. [12] take a much more analytic approach than the method employed by Khoo *et al*. [13]. Instead of training a neural network to figure out a relationship between current pulses and speed, the authors use analytical models of brushed DC motors to develop a method directly based on Equation (26). Figure 10 details the block diagram for the speed estimator used by Vasquez-Sanchez *et al*. [12].

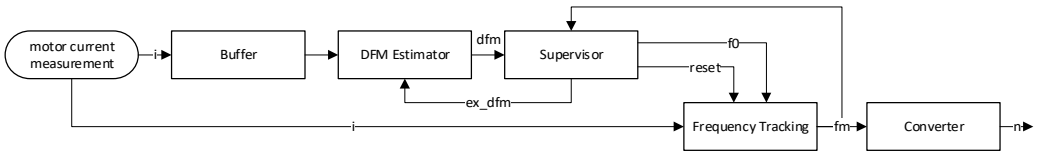

Figure 10: Current Pulse Speed Estimation Method [12].

The input to the system is the motor current. The current samples are stored in a buffer for a set time period so that enough samples to perform a speed estimation are collected. Next, the df_m estimator block performs an estimation of the distance between the frequencies contained in the buffered current samples. It does this by performing a Fast Fourier Transform (FFT), windowing the data, running autocorrelation, comparing to a threshold, detector peak, measuring the distance between the peaks, discarding any impossible distances given the maximum motor speed, and then calculating the mean distance between frequency peaks. The frequency tracker is a phased locked loop that tracks a peak in the spectrum and outputs the value of the frequency. The supervisor contains the logic that determines when to restart the df_m estimator or to set a new frequency for the tracking block. Finally, the converter block uses Equation (26) directly to calculate the estimated motor speed. The researchers report a maximum average error of 0.011% at 2,998 RPM, with testing between 2,000 RPM and 3,000 RPM [12].

Although these results are impressive, the significant processing power required to perform the necessary calculations – in addition to the processing delay – means the application of this or similar methods to a low-cost

microcontroller design is unlikely to be successful. Additionally, the motors used in cordless power tools generally spin two or three times faster than the 3,000 RPM featured in the research. This would force use of processors with even higher clock speeds and specialized hardware math functions than those commonly used in cordless power tool applications.

1.2.2 BEMF Amplitude Methods

In Kamdar, Brahmbhatt, Patel, and Thakker [6], a method is described for controlling the speed of a brushed DC motor. Kamdar *et al*. [6] claim results featuring less than 5% overshoot and a steady-state error of less than 1% (fullscale). The algorithm used was a tuned PID controller that updates every 50ms. A voltage divider is used to level shift the BEMF voltage down to the microcontroller ADC. A 10-bit ADC was used to measure the BEMF. They also employ a high-side motor driver, which makes the BEMF measurement much easier than a low-side motor driver. The BEMF measurement is converted into an angular speed value using a speed correlation factor. The converted angular speed value is then fed into the feedback of the PID controller which controls the motor controller duty cycle.

There are several notable limitations in the research of Kumdar *et al*. [6]. The researchers only tested the system in no-load conditions, evaluating the step response of the system. Performance under various static loads and dynamic loads was not evaluated. Additionally, the sampling time (motor off time) required for this method is 60ms, which will cause major output speed fluctuations in even

moderate loading conditions. It is also unclear why the PID updates every 50ms when the sampling time required to update the measurement is 60ms.

Although Kumar *et al*. [6] were able to achieve some impressive accuracy numbers, the tests conducted were at no-load conditions. It is unlikely that this particular implementation of the BEMF method would be directly suitable for any real-world application.

1.2.3 Inductive Spike Methods

Because this method is relatively new, the only applicable paper on the method was by Kumar and Radcliffe [11] and was discussed in detail in the section on the theory associated with sensorless speed detection. Refer to that section for details on this method.

The literature indicates that sensorless speed detection of brushed DC motors is a current area of research concern. A variety of sensorless speed detection methods have been developed, with the three main categories of development focused on electrical current ripple measurements, BEMF amplitude measurements, and inductive voltage spike measurements. However, no literature was located concerning the application of sensorless speed detection in cordless power tools. Cordless power tools are associated with design demands and conditions including highly variable loading, and high motor no-load speeds, that are currently not considered in the sensorless speed detection literature.

Chapter 2: Test Bench and Data Collection

2.0 Test Bench Overview

A Test Bench was created to repeatably test the various methods under identical conditions. The fixture consists of a load motor connected to ten 5Ω resistors, each connected to a relay. The relays are controlled by an ATMega328P microcontroller via 2N7000 MOSFETs. The powered motor is coupled to the load motor via a 5mm shaft coupler. The ATMEga328P can control the load by switching the resistors in and out individually. A control signal from the motor controller starts the automated test sequence.

2.1 Test Bench Hardware Design

The fixture block diagram with important connections is shown in Figure 11. The Relays used are ORWH-SH-112D1F,000 General Purpose SPDT 10A 12V relays. Only one of the connections for each relay were used, effectively making it a SPST design. The power resistors used are KAL25FB5R00 5Ω 1% 25W. Both the load and powered motors were taken directly from Milwaukee Tool model number 2626-20 Multitools. Figure 11 shows the details of the test bench design.

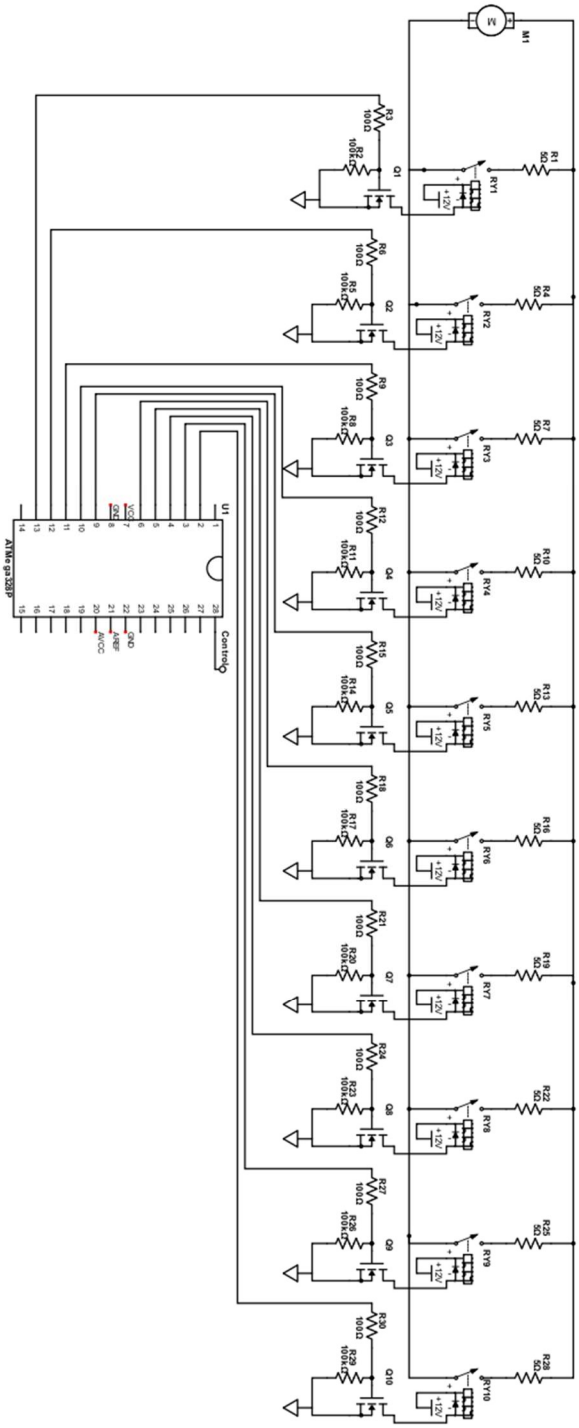

Figure 11: Test Bench Fixture Diagram.

The completed Test Bench connected to the Device Under Test (DUT) is shown in Figure 12.

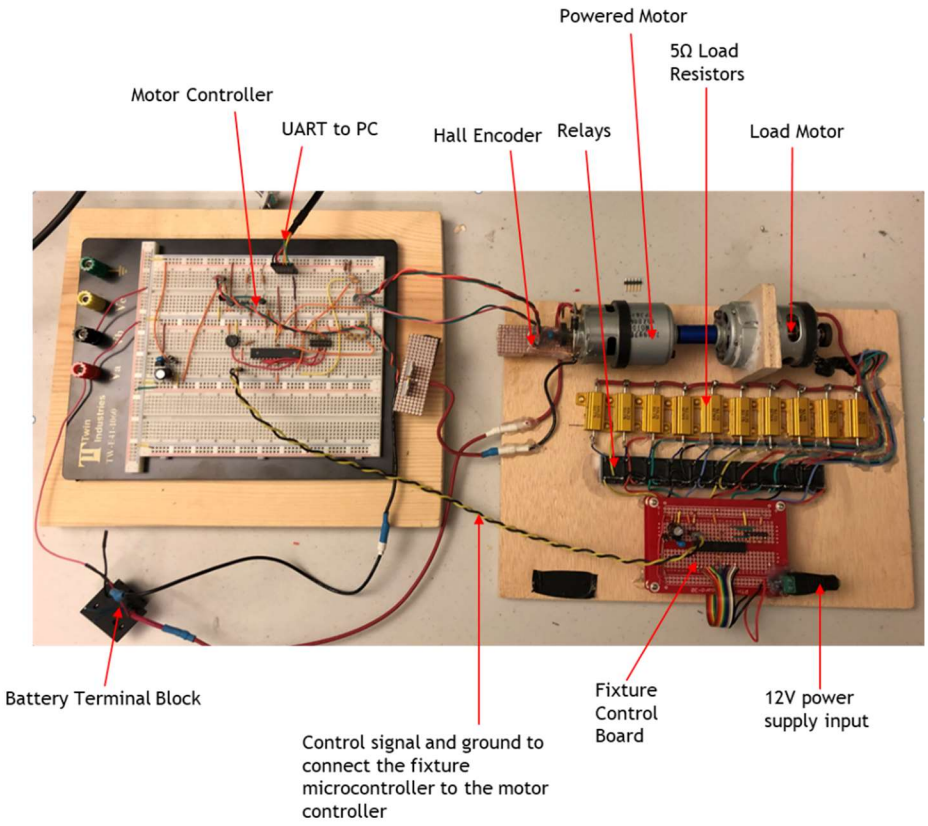

Figure 12: Test Bench Design.

Updates were then made to the original test bench design to incorporate the motor controller and test bench in the same fixture. The result is shown in Figure 13.

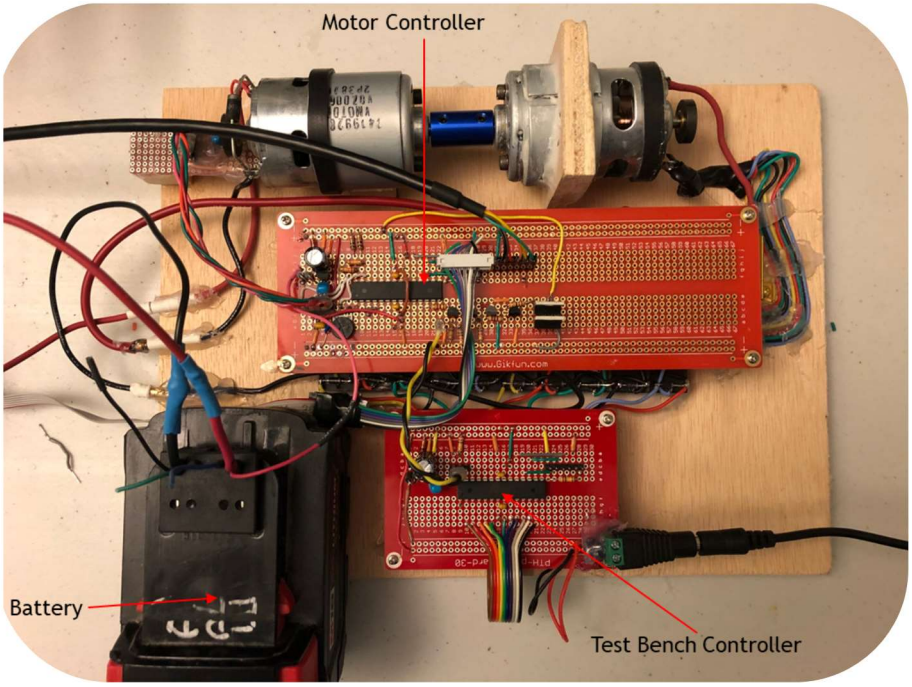

Figure 13: Combined Test Bench and Motor Controller.

2.2 Test Bench Firmware Design

The firmware running on the Test Bench waits for a signal from the motor controller before it starts closing the relays in sequence. The sequence contains three parts. The first part of the sequence simulates the load increasing as the tool is brought in contact with the workpiece. The second part is the application performed at maximum load for three seconds (30A load here simulates a cut in oak). Finally, the load is decreased back down to no-load in ten steps to simulate breaking through the back of the workpiece. The current load profile and control signal are shown in Figure 14.

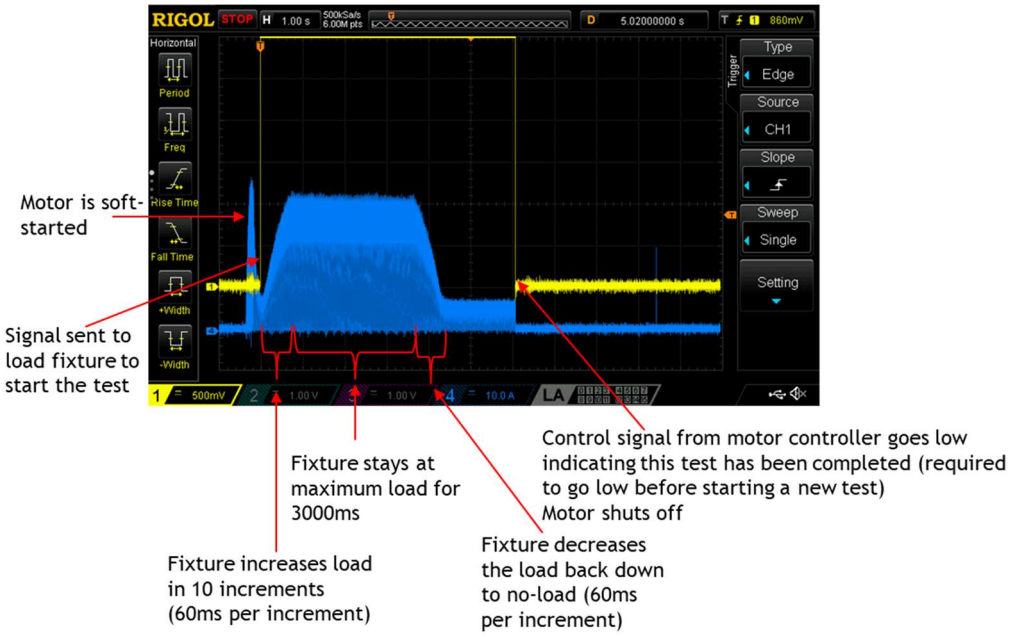

Figure 14: Automatic Load Profile – CH1: Control Signal, CH2: Drive Motor Current.

Consecutive load profiles may be commanded by the drive motor controller by sending the control signal low and then high again. If at any time the control signal goes low during a test, that test is immediately terminated, and all loads are disconnected.

The firmware runs on a one millisecond super-loop similar to the motor controller design. The flowchart for the Test Bench firmware is shown in Figure 15.

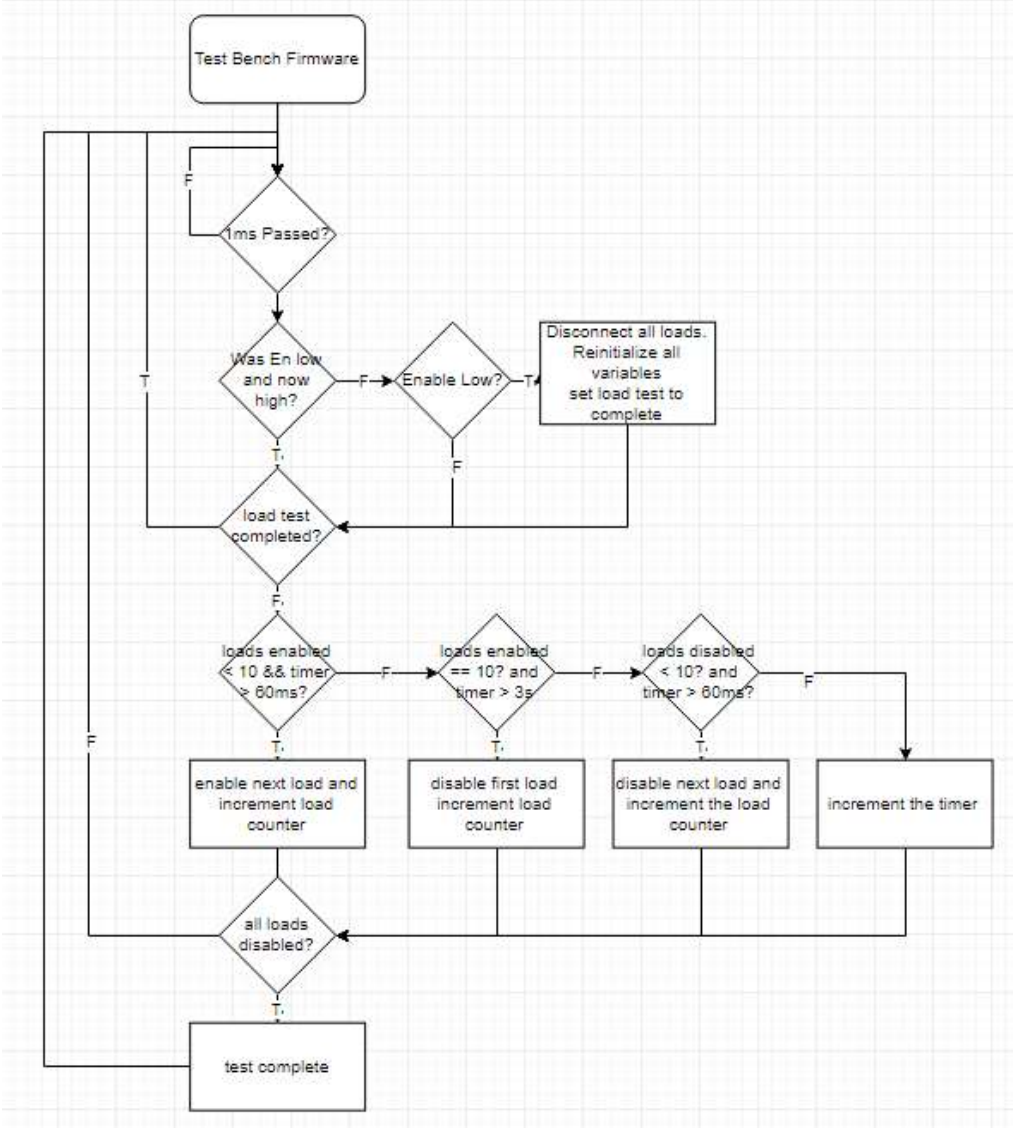

Figure 15: Test Bench Firmware Flowchart.

2.3 Standardized Testing for Comparing Methods

A standardized test procedure was developed for comparing the different methods. It consists of six tests. The first test is a no-load speed ramp. The speed is varied from zero RPM to the maximum RPM of the motor and then back down to zero RPM over a nine-second period. Then five loaded tests are performed at various duty cycles. The first test is 10%, followed by 25%, 50%, 75%, and

finally 100% duty cycle tests. The load profile is created using the Test Bench, switching in and out all 10 load resistors. Each test is initiated by the motor controller. The Test Bench runs the same resistor switching profile for each test. During the no-load test, the Test Bench is not running any load profile (all loads disconnected).

An example standardized testing profile result is shown in Figure 16. The MATLAB script also tracks error relative to the hall encoder measurement, and provides average percent, and absolute error for the entire test.

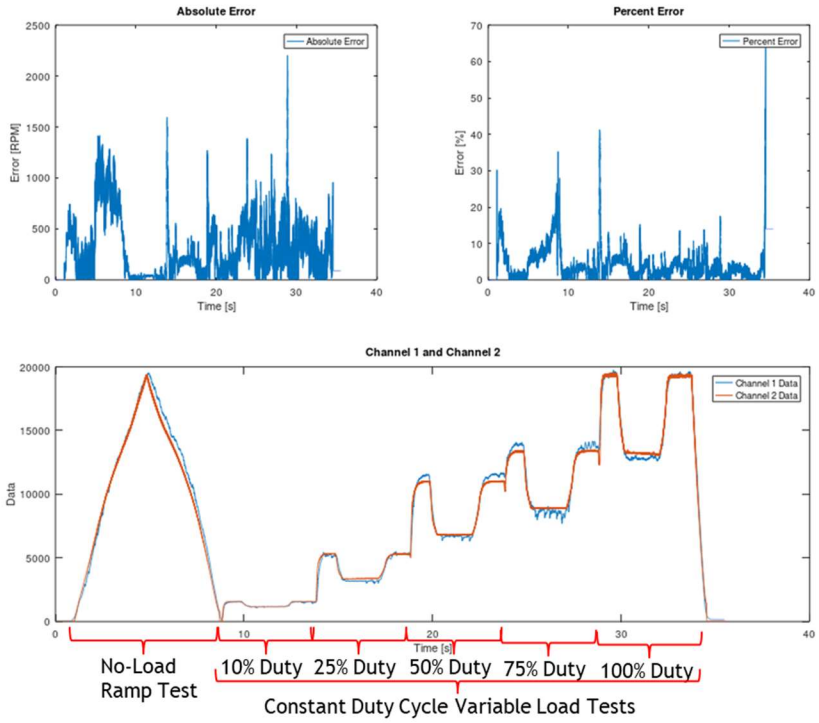

Figure 16: Example Standardized Test.

2.4 MATLAB Scripts and RealTerm Terminal

The UART hardware built into the ATMega328P was used to export data from the microcontroller to a UART-to-USB converter. These raw data were

captured using the RealTerm Terminal Capture Program. Once the data were stored in a text file, a custom MATLAB script was run to parse the data, perform any necessary calculations, and plot the data. This process is depicted in Figure 17.

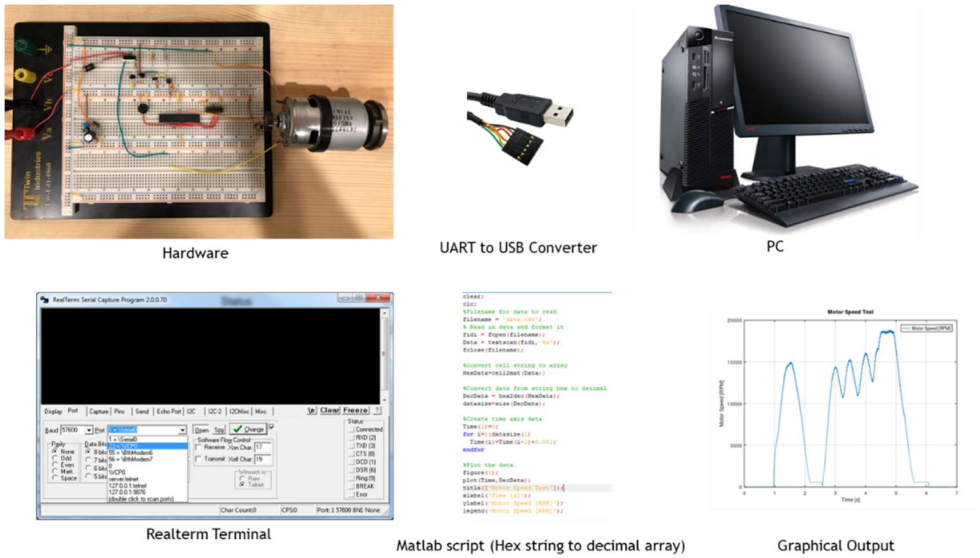

Figure 17: Data Collection Method.

Details on the exact process for collecting and converting data can be found in Appendix A.

Chapter 3: Firmware Design

3.0 Firmware Architecture

The firmware that runs on the motor controlling microcontroller is split into several source and header files by function. There are three folders. The Baseline Header and Baseline Source folders contain all of the header and source files for functions that are common to all measurement methods. The Measurement Methods folder contains all the source and header files for the various measurement techniques. Figure 18 shows this file structure.

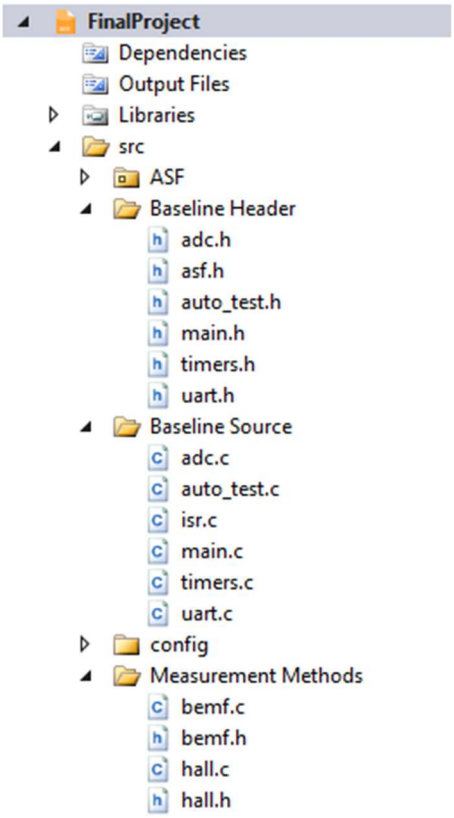

Figure 18: Firmware File Structure.

The adc .c and .h files contain all functions for controlling and sampling the ATMega328P's ADC. The main .c and .h files contain the main loop, speed

dial, and battery control functions. The timers .c and .h files contain the timer configuration functions for the super-loop, hall encoder, and measurement method timers. The isr.c file contains all the interrupt vectors for pin-based and timer-based interrupts. The auto_test .c and .h files contain the functions for the standardized automatic tests used to compare the various measurement methods. The uart .c and .h files contain the functions needed to transmit data across the UART interface.

In the Measurement Methods folder, the files contain the functions for each of the various measurement methods. Hall .c and .h files contain the functions for the hall encoder measurement method, bemf .c and .h contain the functions for the BEMF measurement method, and so on.

3.1 Main Program

The main program runs in the main.c file. Upon power-up, initialization functions are run for the timers, UART, auto-test, and measurement methods. The program then enters the super-loop which runs once every millisecond. Within the super-loop, the PWM duty cycle is set via either the speed dial or the auto-test functions, motor speeds are calculated, and data are sent to the PC over the UART. The flowchart for the main function is shown in Figure 19.

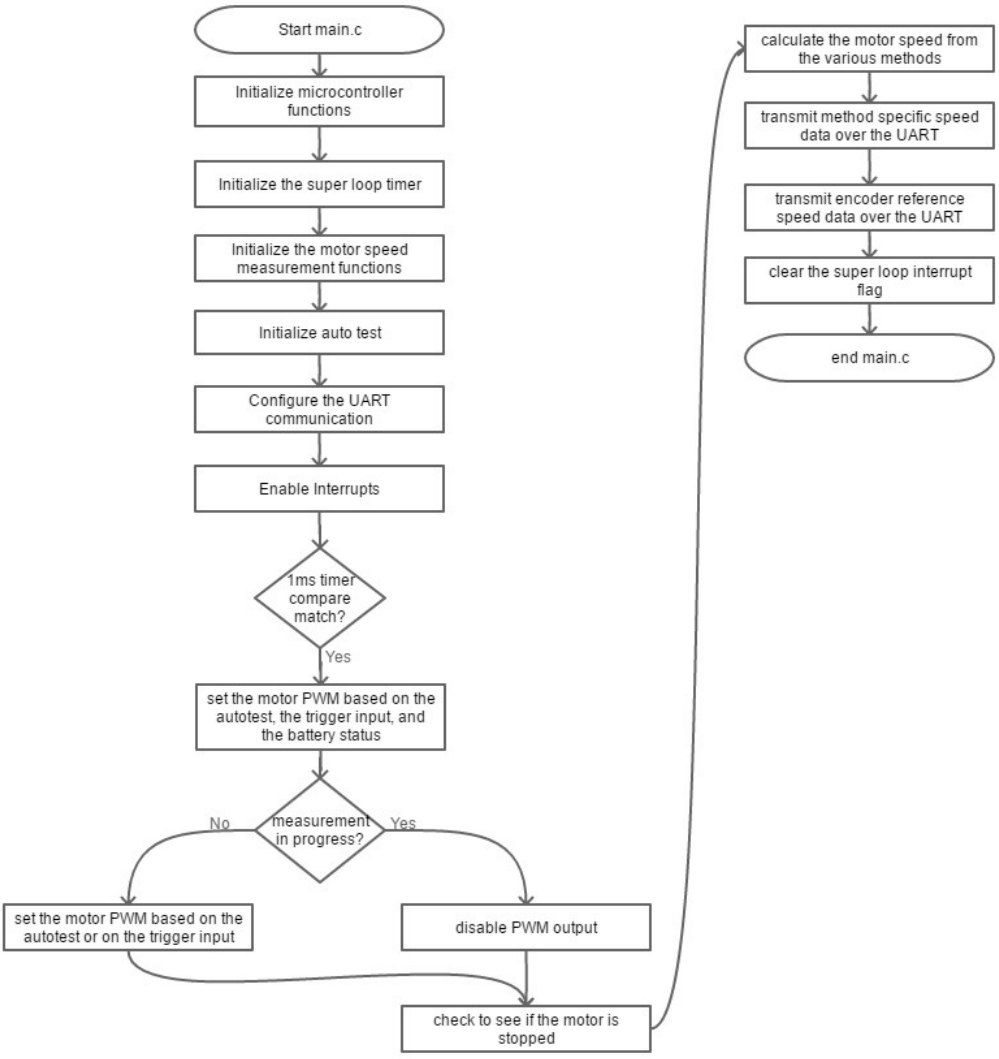

Figure 19: Main Program Flowchart.

3.2 Interrupt Service Routines

The only Interrupt Service Routine (ISR) implemented is for the hall encoder pin interrupt. This ISR calls the calculate_motor_speed_hall_isr() function. The flowchart for the ISR is shown in Figure 20.

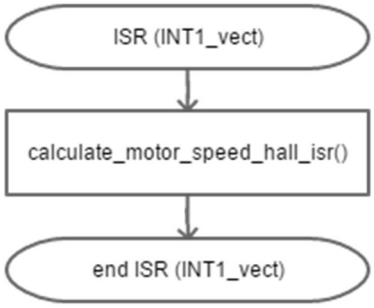

Figure 20: Hall Encoder ISR.

The flowchart for the calculate_motor_speed_hall_isr() function can be found in the Hall Encoder Speed Detection section of this report.

3.3 Analog-to-Digital Converter

The adc.c file contains one globally accessible function for reading the ADC. This function configures the ADC registers, and starts an ADC reading. Once a conversion has been started, the function waits for the result to complete and then returns the result. The desired ADC channel is passed into the function. Figure 21 shows the details of the function operation.

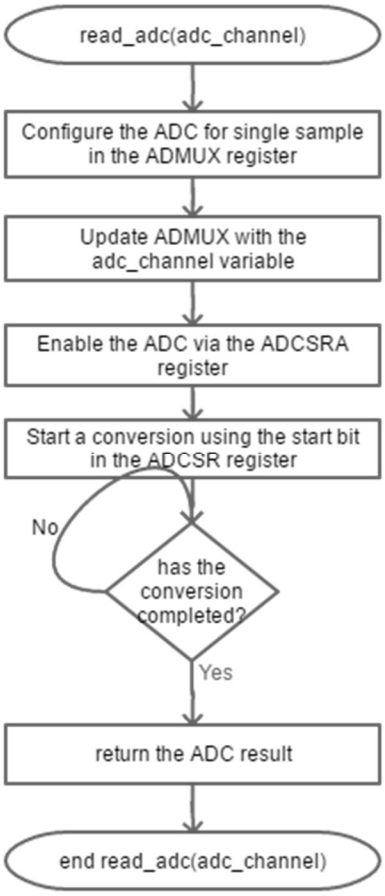

Figure 21: Read ADC Function.

3.4 Timers

There are five functions in the timer.c file related to configuring, starting, and stopping various timers. The initialize_super_loop_timer() function configures the Timer Counter Two registers to kick off the super-loop every one millisecond. Figure 22 shows the flowchart for this function.

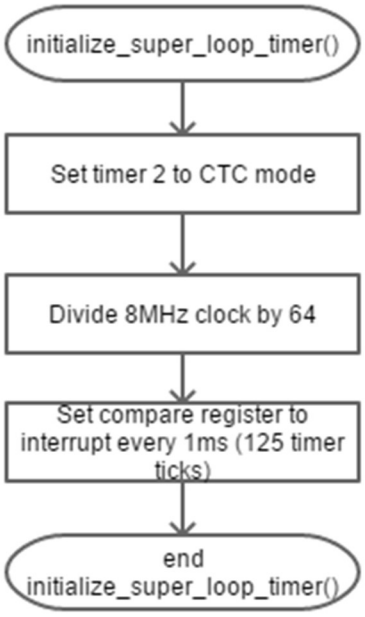

Figure 22: Super-loop Initialization Function.

The set_pwm(duty_cycle, frequency) function takes two arguments: the duty cycle and desired switching frequency. If the duty cycle is zero percent, the PWM duty cycle is disabled and the PWM output pin is cleared. If the desired duty cycle is 100%, the PWM duty cycle is disabled and the PWM output pin is set. Otherwise, the timer compare register is set to a value corresponding to the desired duty cycle. The switching frequency can be set to three different settings: 33kHz, 3kHz, or 500Hz. Figure 23 shows the flowchart for this function.

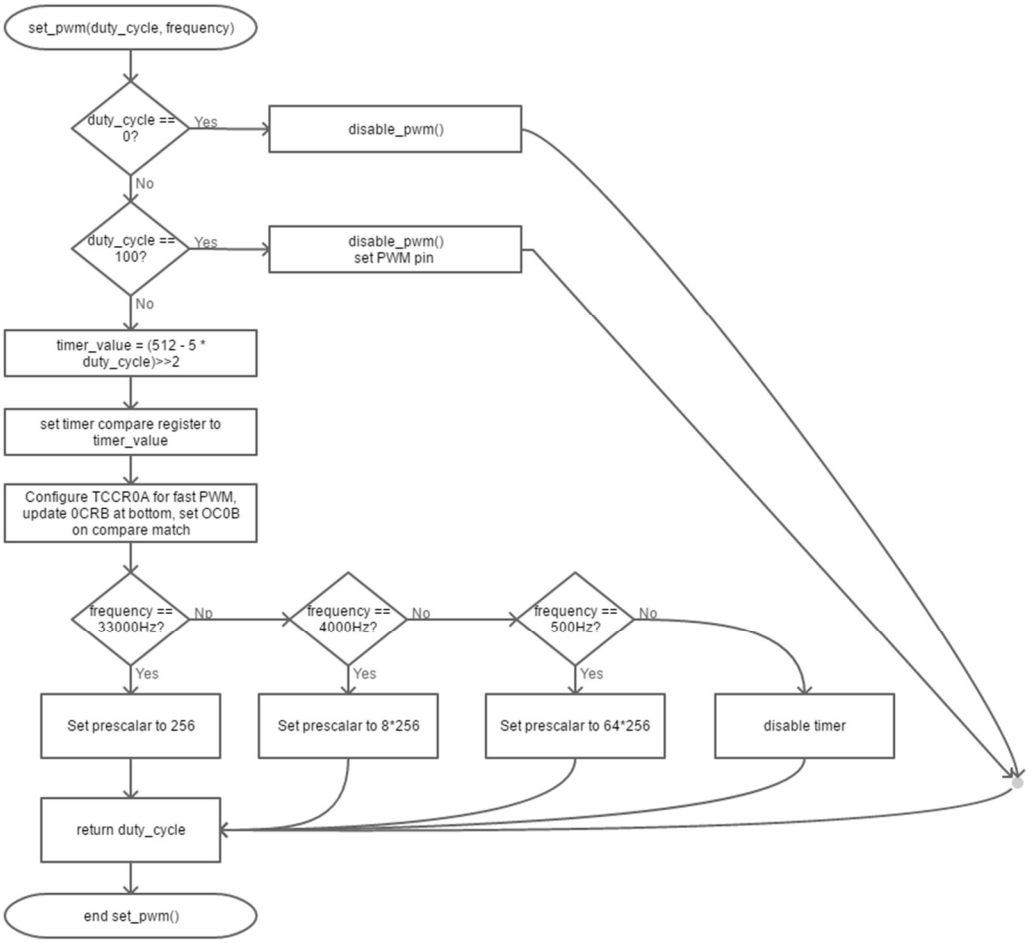

Figure 23: Flowchart for set_pwm() Function.

The disable_pwm() function disables the timer-counter 0 timer and clears the PWM output pin. Figure 24 shows the flowchart for this function.

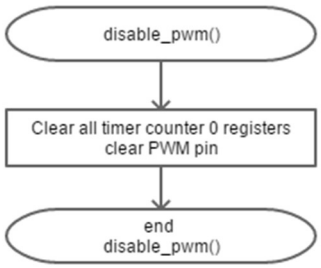

Figure 24: Flowchart for disable_pwm() function.

The get_hall_timer_value() reads the value of the hall timer-counter value register (TCNT1). Then it resets and restarts the timer-counter and returns the value of the TCNT1 register. Figure 25 shows the flowchart for the get_hall_timer_value() function.

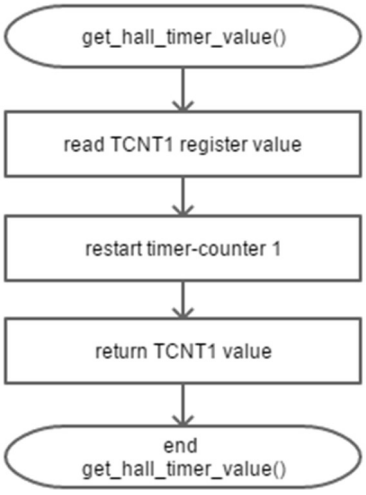

Figure 25: Flowchart for get_hall_timer_value() Function.

The stop_hall_timer() function stops timer-counter one and clears the registers. The flowchart for the stop_hall_timer() function is shown in Figure 26.

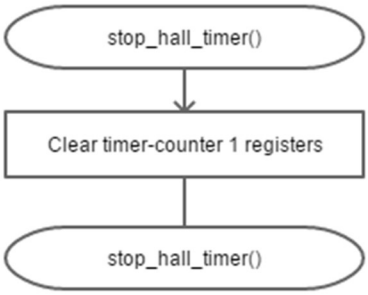

Figure 26: Flowchart for stop_hall_timer() Function.

3.5 Universal Asynchronous Receiver Transmitter

The UART source file contains three functions. The uart_init(ubrr) function initializes the UART baud rate, enables the receiver and transmitter and sets the frame format to eight data bits and two stop bits. The flowchart for the uart_init(ubrr) function is shown in Figure 27.

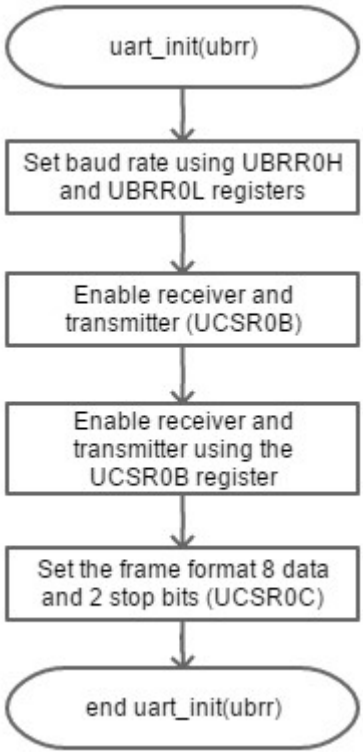

Figure 27: Flowchart for uart_init() Function.

The uart_tx_data(data) function formats the data to be sent using the Serial Line Internet Protocol (SLIP). Once the data are formatted, this function sends the data out over the UART transmit line. Figure 28 shows the flowchart for the uart_tx_data(data) function.

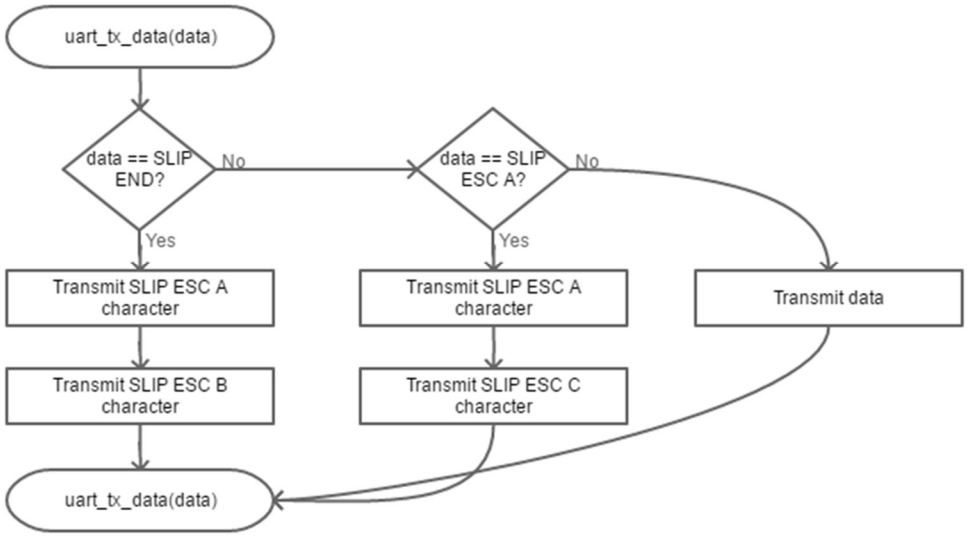

Figure 28: Flowchart for uart_tx_data() Function.

The uart_txt_start() function transmits a SLIP start byte.

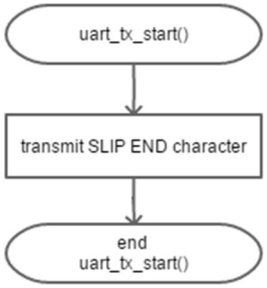

Figure 29: Flowchart for uart_tx_start() Function.

3.6 Hall Encoder Speed Detection

The hall.c file contains functions for measuring the motor speed directly using a traditional magnet and hall sensor-based encoder. There are four functions for this feature.

The initialize_motor_speed_hall() function clears all the static variables used in the hall encoder speed detection feature. The flowchart for this function can be found in Figure 30.

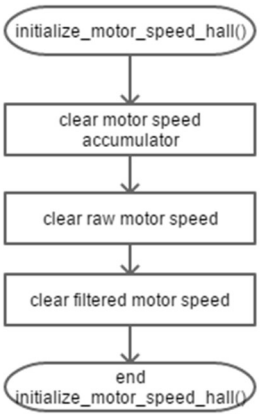

Figure 30: Flowchart for initialize_motor_speed_hall() Function.

The calculate_motor_speed_hall_isr() function is the function that calculates the motor speed every time the hall sensor interrupt fires (when a new magnet pole passes the sensor). This function also filters the motor speed. Figure 31 shows the flowchart for this function.

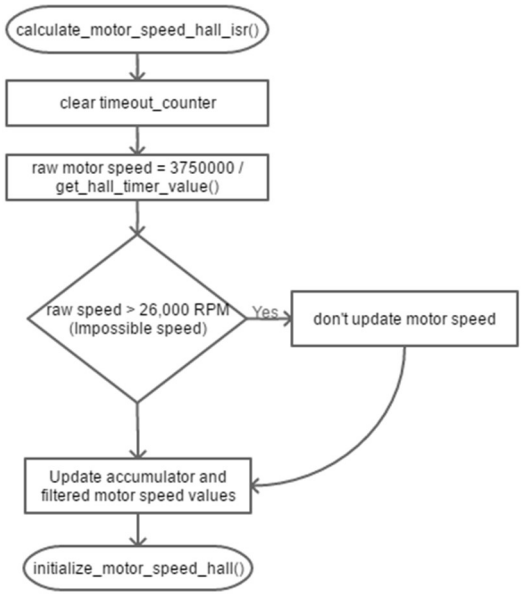

Figure 31: Flowchart for calculate_motor_speed_hall_isr() Function.

The check_for_motor_stopped() function zeros out the motor speed if the time between hall interrupts approaches the overflow limit of the timer-counter. The flowchart of this function is shown in Figure 32.

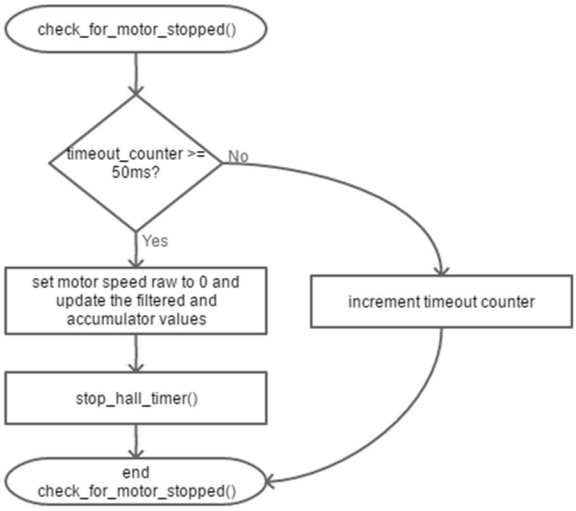

Figure 32: Flowchart for check_for_motor_stopped() Function.

The get_motor_speed_hall() function just returns the filtered motor speed value. The flowchart for this function is detailed in Figure 33.

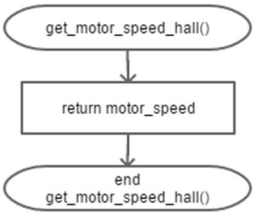

Figure 33: Flowchart for get_motor_speed_hall() Function.

3.7 Closed-Loop Speed Control

A simple Proportional Integral (PI) controller was created to perform the closed-loop speed control functionality. The inputs to the PI controller are the

measured speed (any measurement method can be used for this input), and the target speed. Figure 34 is the flowchart for the PI controller.

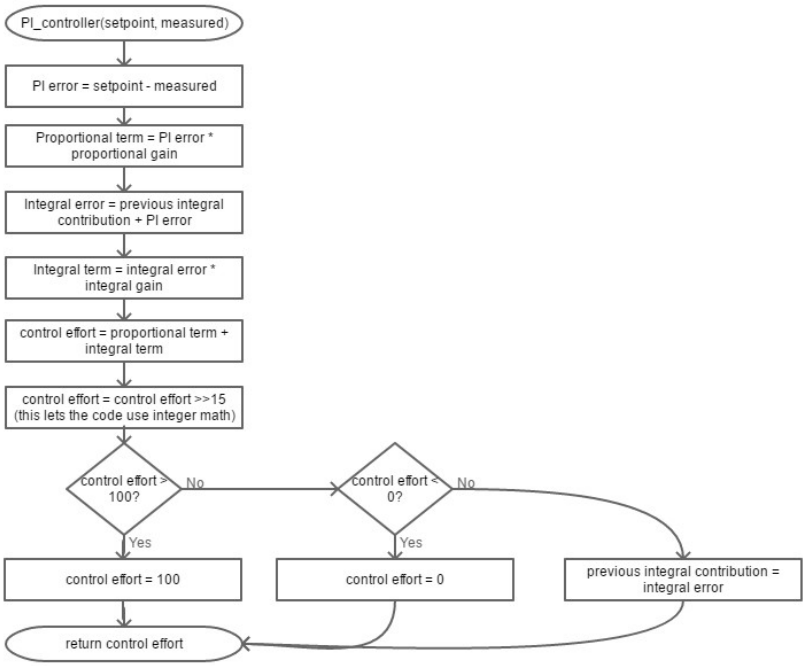

Figure 34: Flowchart for PI Controller.

The PI controller can be used in conjunction with the auto test routine to test the closed-loop speed control performance of the various methods and compare them to the performance of the same closed-loop speed control algorithm using the traditional hall encoder.

Chapter 4: Sensorless Implementation Methods

4.0 BEMF Measurement

4.0.1 Overview

The BEMF measurement method utilizes the relationship between the BEMF of the motor and the motor speed. There are several significant challenges that must be overcome for a BEMF method to work properly. In order to measure the BEMF, the motor current must drop to zero. This means this measurement method will not work at a one-hundred percent duty cycle. Additionally, in different loading situations, it takes different amounts of time for the motor current to drop all the way to zero. Finally, the motor constant must be known in order to accurately convert the measured BEMF voltage to a speed in RPM.

4.0.2 Design Details

Two measurements are needed to calculate the BEMF voltage. These include the battery voltage at the positive motor terminal and the voltage at the negative terminal of the motor. Resistor dividers are used to scale the voltage down from the 0 to 21V battery voltage to a 0 to 3.3V signal that can be read by the ADC.

In order to maximize the possible duty cycle, periodic measurements are taken every 4ms. This allows a much higher duty cycle to be maintained than if samples were taken every PWM cycle.

While the freewheeling diode across the motor terminals is conducting, the motor negative terminal voltage is very close to the motor positive voltage (one diode-drop difference). Once the diode stops conducting, the negative

terminal voltage will fall to a voltage relative to the motor speed. The firmware algorithm waits for this drop to occur before taking measurements. This method allows for a variable measurement time that will always be the minimum possible for a given motor load. Figure 35 shows this method in action using an oscilloscope.

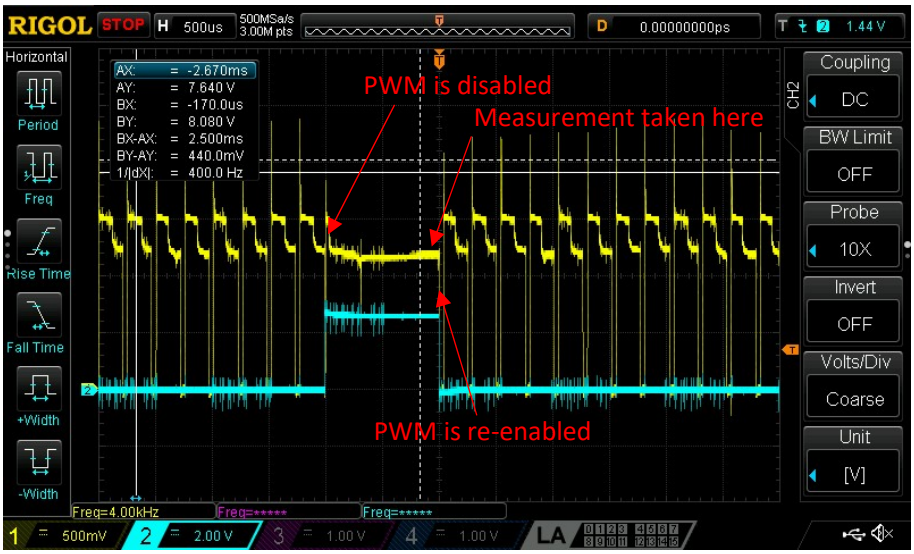

Figure 35: Measurement Example – CH1: Input to Motor- ADC Pin.

Figure 36 shows the slower measurement frequency relative to the PWM frequency.

Figure 36: Measurement Frequency - CH1: Negative Motor Terminal Voltage.

Eight total samples are taken for both the battery voltage and the negative terminal voltage. The eight samples are then averaged to help filter signal noise. The BEMF is converted into motor speed in RPM using the motor constant found for this motor.

The motor constant is found by measuring the BEMF voltage using the ADCs and measuring the motor speed using the hall encoder. The speed of the motor was varied across its entire speed range for this test. The slope of the BEMF versus motor speed linear fit was used for the conversion. This relationship is shown in Figure 37.

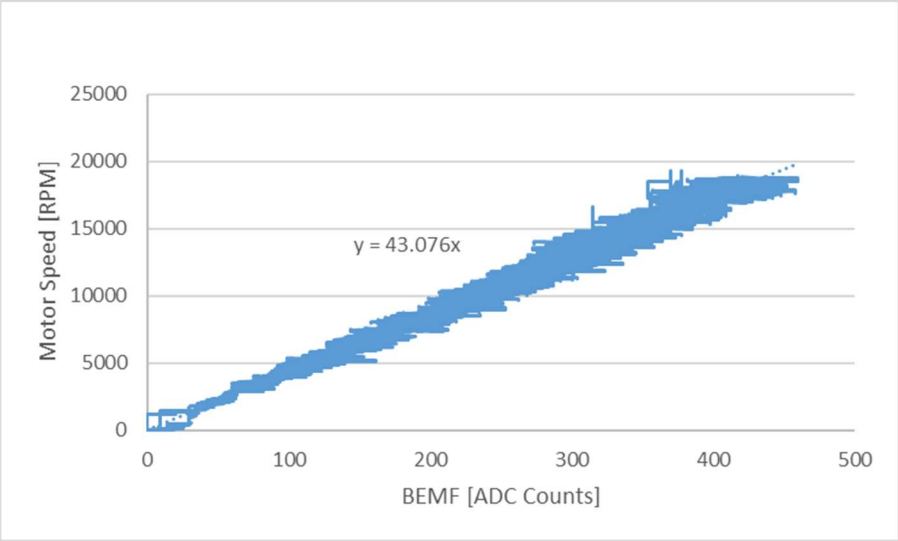

Figure 37: Motor Speed versus BEMF ADC Counts.

The slope of 34.749 was multiplied by 1024 for better resolution when performing integer math calculations. Once a result was found, the result was bitshifted back by 10 (divide by 1024) to yield the final speed in RPM. Finally, an IIR low-pass filter was used to further reduce signal noise. The flowchart for the BEMF measurement firmware is shown in Figure 38.

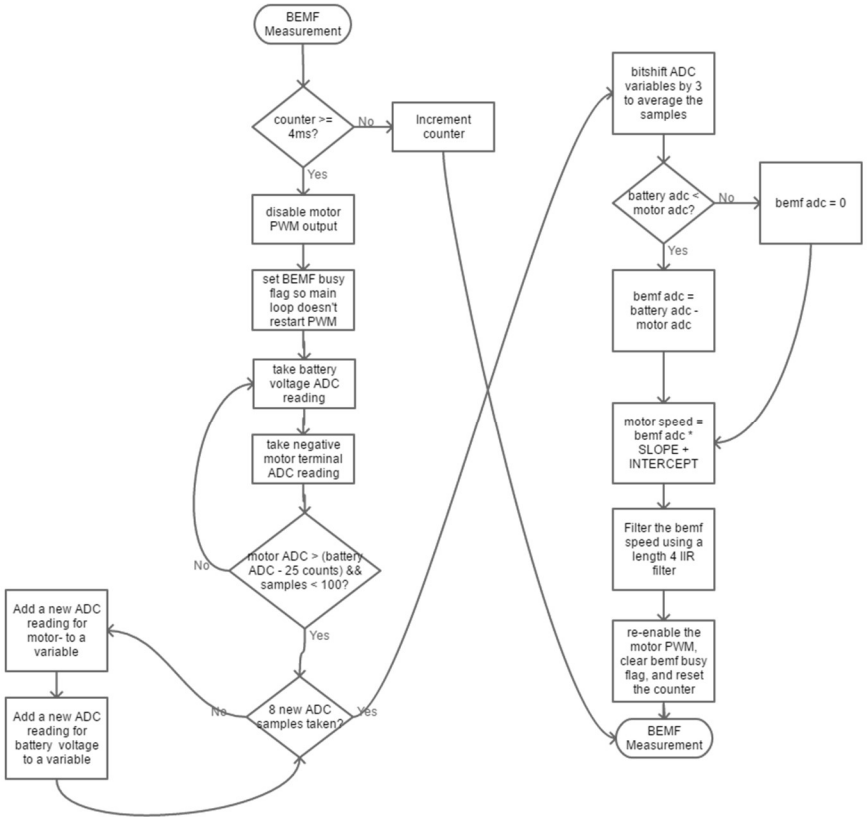

Figure 38: BEMF Measurement Flowchart.

The calculate_motor_speed_bemf() function is called once per main loop (once every millisecond). Performance results of the BEMF method can be found in the Results and Discussion section of this report.

4.1 Inductive Spike Duration

4.1.1 Overview

The inductive spike duration method utilizes the relationship between motor speed and inductive pulse duration proposed by Kumar and Radcliffe [11]. Every four milliseconds, a measurement is initiated by the main loop. The pulse is measured by first starting a timer, then taking several initial measurements of the

negative motor terminal voltage and averaging them together. Then, more ADC measurements are taken as quickly as possible. The new measurements are compared to the initial average. Once the new measurement is sufficiently far below the initial measurement, the end of the inductive pulse is found. After the end of the pulse is found, the timer is read to see how much time has passed since the pulse started.

After the pulse duration has been measured, the pulse duration is converted into a motor speed value using a piecewise linear approximation of the exponential relationship between motor speed and pulse duration.

4.1.2 Design Details

An inductive pulse contains three distinct features: A rise time (with diode turn-on ringing), a flat-top, and a decay. When the PWM pulse transitions from high to low (switching element is off), an interrupt starts the measurement. The initialization time is slightly longer than the diode ringing, so no additional blanking is needed. Several measurements are averaged together in the flat-top part of the pulse. Then the firmware waits for several measurements to fall below a pre-determined threshold before indicating a pulse edge has been found. Figure 39 illustrates this process.

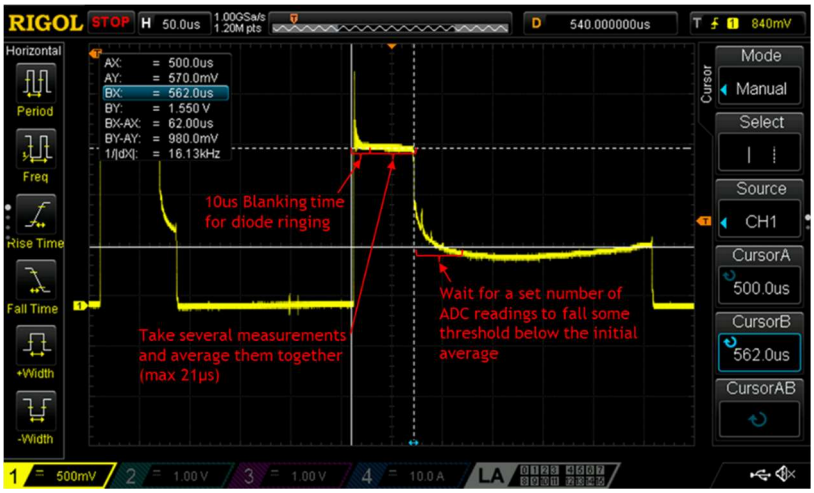

Figure 39: Measurement Methodology.

There are two parts to the inductive speed measurement firmware. These are the main loop configuration and the interrupt-based measurement. The main loop configuration is responsible for tracking when a measurement should be taken (every four milliseconds) – and when a measurement should be taken, it configures the interrupt, number of samples, drop threshold, and sets the ADC channel. Figure 40 shows the details of the initialization function.

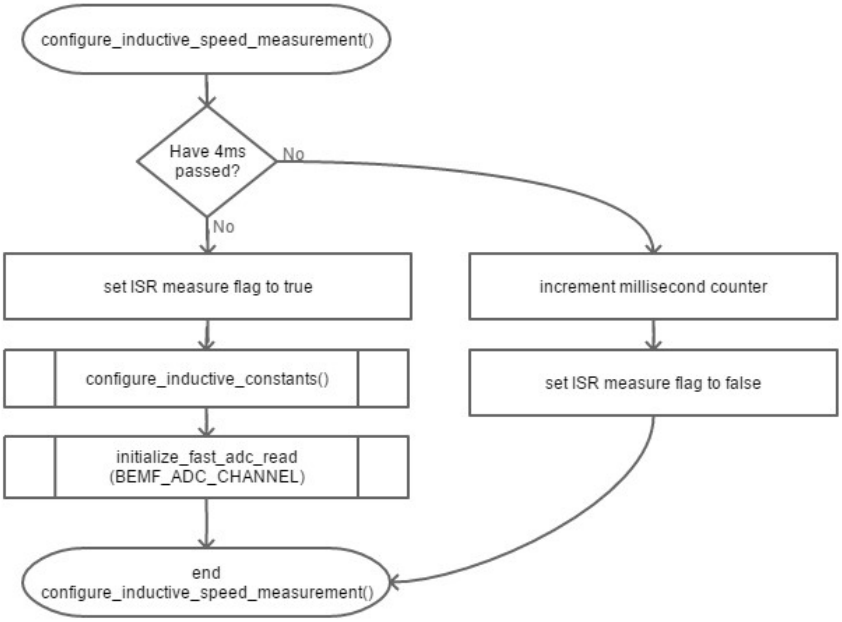

Figure 40: Flowchart for Inductive Measurement Initialization.

The inductive pulse measurement constants are number of samples, drop threshold, and bitshift. All of these parameters are based on the previously measured pulse duration. The number of samples parameter is the number of ADC readings in the flat-top region that are averaged together for the initial measurement. The drop threshold is how far samples taken after the initial measurement must be below the initial measurement in order for a pulse edge to be defined.

Figure 41 shows the difference between an inductive pulse at low speed and high speed. At low speeds, the pulse has a very long duration, but the voltage drop defining the pulse edge is relatively small.

At high speed, the opposite is true. The pulse duration is short, but the voltage drop defining the pulse edge is large. Note that this voltage is the voltage

at the ADC pin of the microcontroller – that is, the scaled down negative motor terminal voltage.

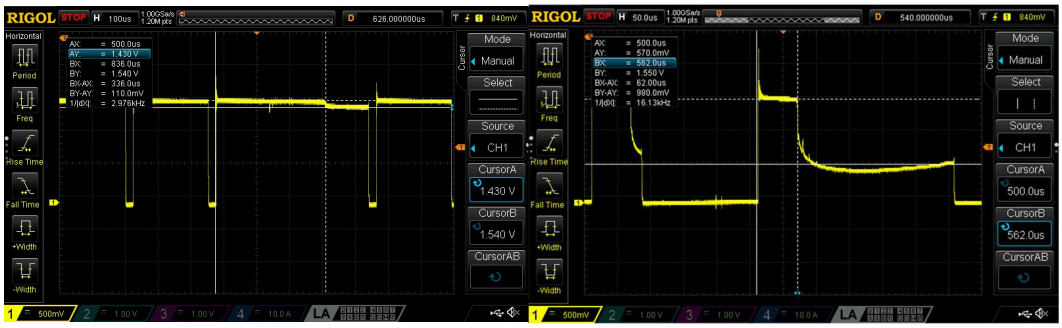

Figure 41: Inductive Pulse at Low Speed (Left) and High Speed (Right).

The firmware can take advantage of the long pulse duration at low speeds by taking more measurements for the initial average value. At high speed, the firmware can take advantage of the large voltage drop. This capability helps compensate for the noisier initial value. The bitshift value is the power of two related to the number of samples (the number of samples must be a power of two).

Three different sets of constants are used depending on the length of the previous pulse. Figure 42 shows the process for deciding which threshold to use.

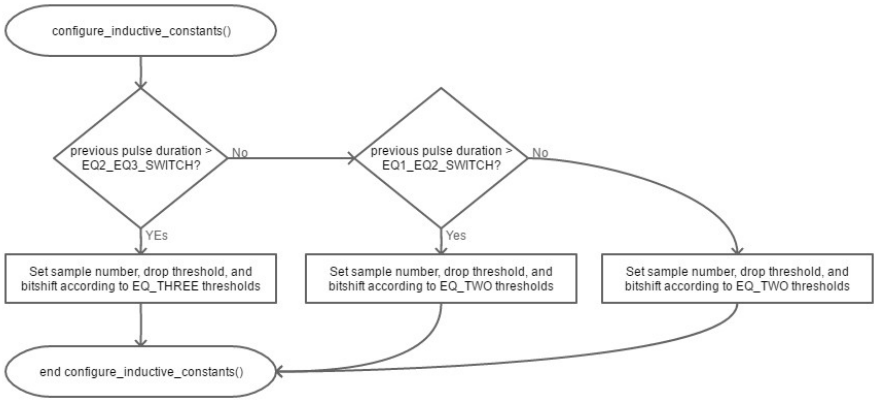

Figure 42: Flowchart for Configuring the Measurement Constants.

Now that the firmware has the configuration for the measurement set, on end of the next PWM pulse, an interrupt subroutine (ISR) starts the measurement. The details of the ISR measurement are shown in Figure 43 and continued in Figure 44.

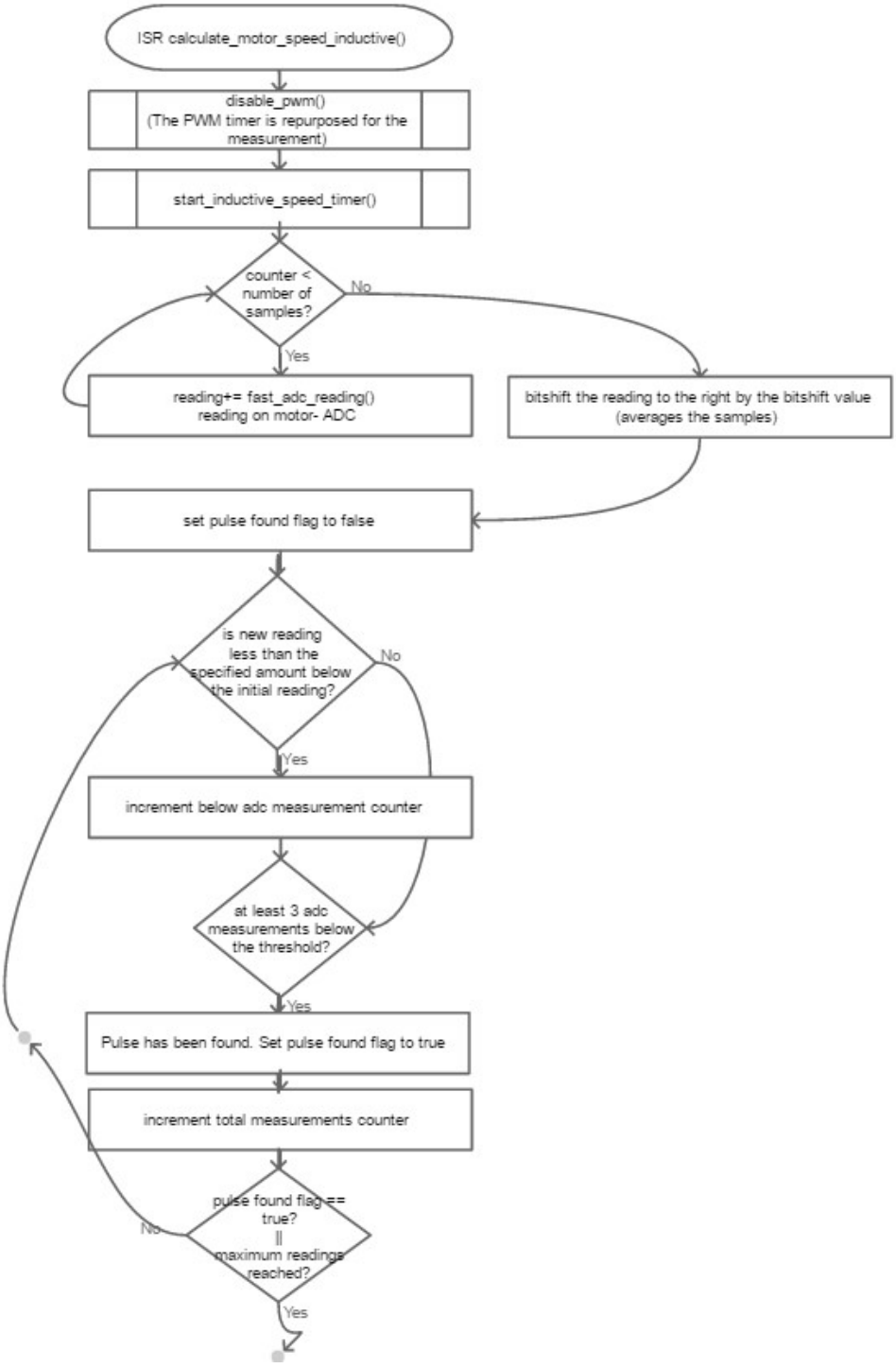

Figure 43: ISR Measurement Flowchart.

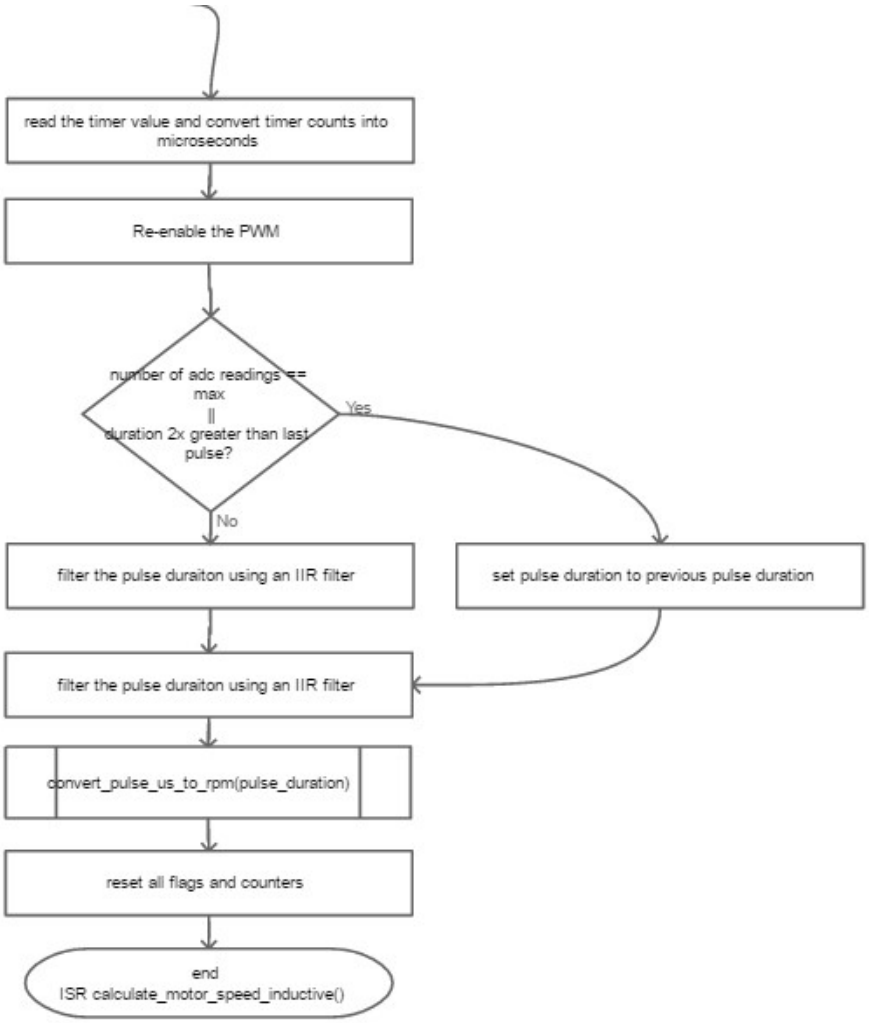

Figure 44: ISR Measurement Flowchart Continued.

In order to see if the measurement was working correctly, a pin was toggled when the firmware thought it had found a pulse edge. This was then measured on an oscilloscope to see how close the firmware was to the actual pulse edge. Measurements were taken at 10% duty cycle, 50% duty cycle and 100% duty cycle to check performance across the entire range of pulse durations and drop thresholds. In all three cases, the firmware accurately found the pulse edges.

Figure 45 shows the results for each case. Channel 1 is the pin toggle for the firmware estimate, Channel 2 is the actual scaled negative motor voltage terminal voltage.

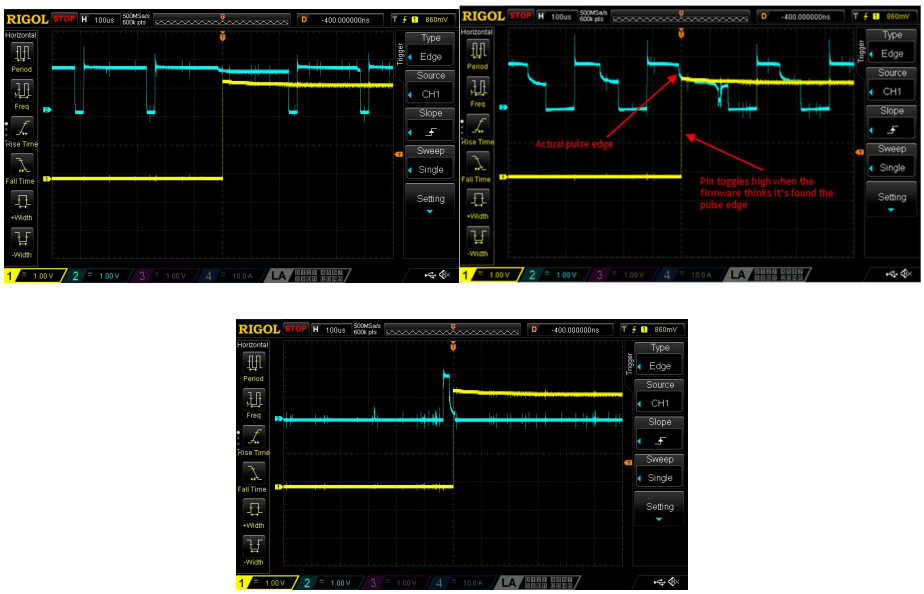

Figure 45: Firmware versus Actual – 10% Duty Cycle (Top-Left), 50% Duty Cycle (Top-Right), 100% Duty Cycle (Bottom).

The piecewise linear mapping used to convert the pulse duration into an RPM value is based on measured pulse duration versus hall encoder motor speed performed at no-load. Figure 46 shows the no-load piecewise linear mapping of pulse duration and motor speed. Piecewise linear mapping was used because the true exponential relationship would take too much processing time in the chosen microcontroller.

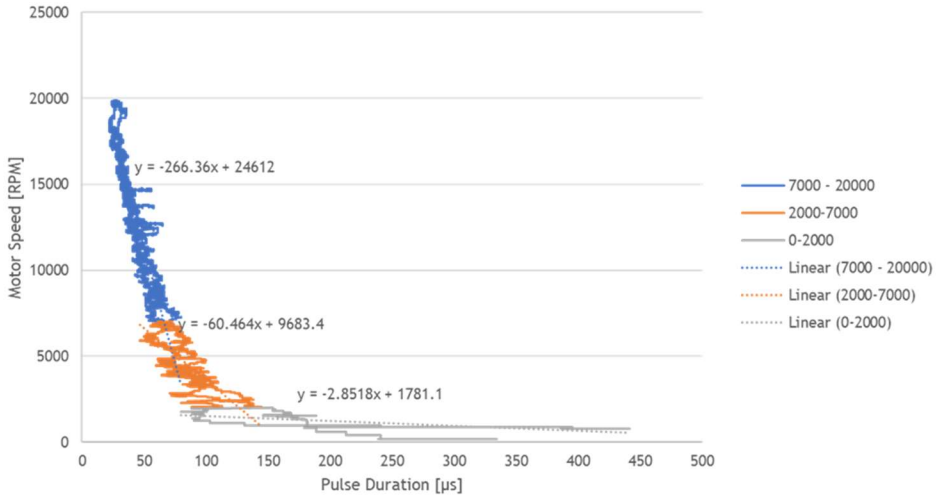

Figure 46: Piecewise Linear Mapping.

Table 2 shows the slope and intercept values for each equation as well as the switchover points in pulse duration where the slope and intercept values change.

Table 2: Piecewise Slope and Intercept Values.

Equation	Slope	Intercept	Switchover
1	-266	24612	73
2	-60	9683	137
3	-3	1781	NA

The firmware flowchart for the convert_pulse_us_to_rpm(pulse_duration) function used to select the slope and intercept values based on the measured pulse duration is shown in Figure 47.

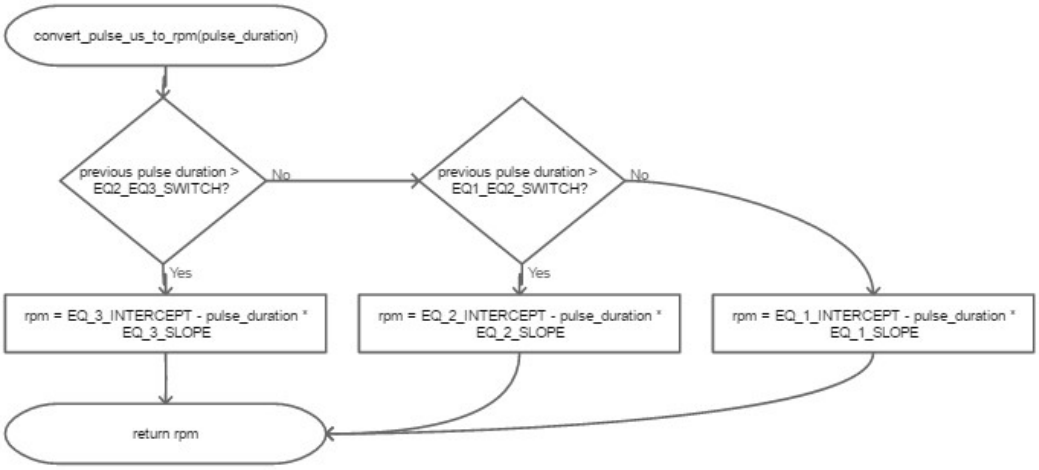

Figure 47: Slope and Intercept Setting Flowchart.

4.2 Current Pulse Counting

The current pulse counting takes a measurement and runs an algorithm every PWM pulse. The ISR runs at the middle of the PWM pulse. Figure 48 shows the ISR execution relative to the current. Channel 2 goes high when the ISR starts and goes low when the ISR execution ends. Channel 4 shows the current through the drive MOSFET during the ISR execution.

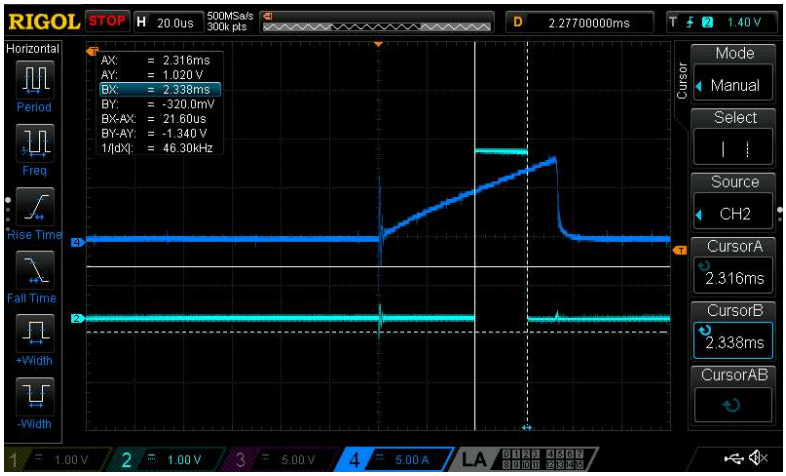

Figure 48: Current Ripple ISR Timing.

Because the current measurement is taken every PWM pulse, the effective current sample rate is equal to the PWM frequency. The maximum usable PWM frequency is limited by the ISR execution time, and the execution time of code in the super-loop and other ISRs. For this system running on an 8MHz clock, the maximum sampling frequency is 4kHz. The ISR measurement and algorithm take 21.6µs to complete (when an 8MHz clock is used).

The ripple current methods used in the literature rely on taking and storing a lot of current sample data over several seconds, and then performing an FFT on that data to determine the motor speed.

This method is unsuitable for a system that needs feedback every millisecond to update a PID controller. So instead, a peak and valley detection algorithm is used in conjunction with a timer to measure the time between the individual peaks. This method updates at the PWM frequency, which at a minimum, is four times faster than the update rate of the PID controller.

The details of the current ripple detection algorithm are shown in Figure 49.

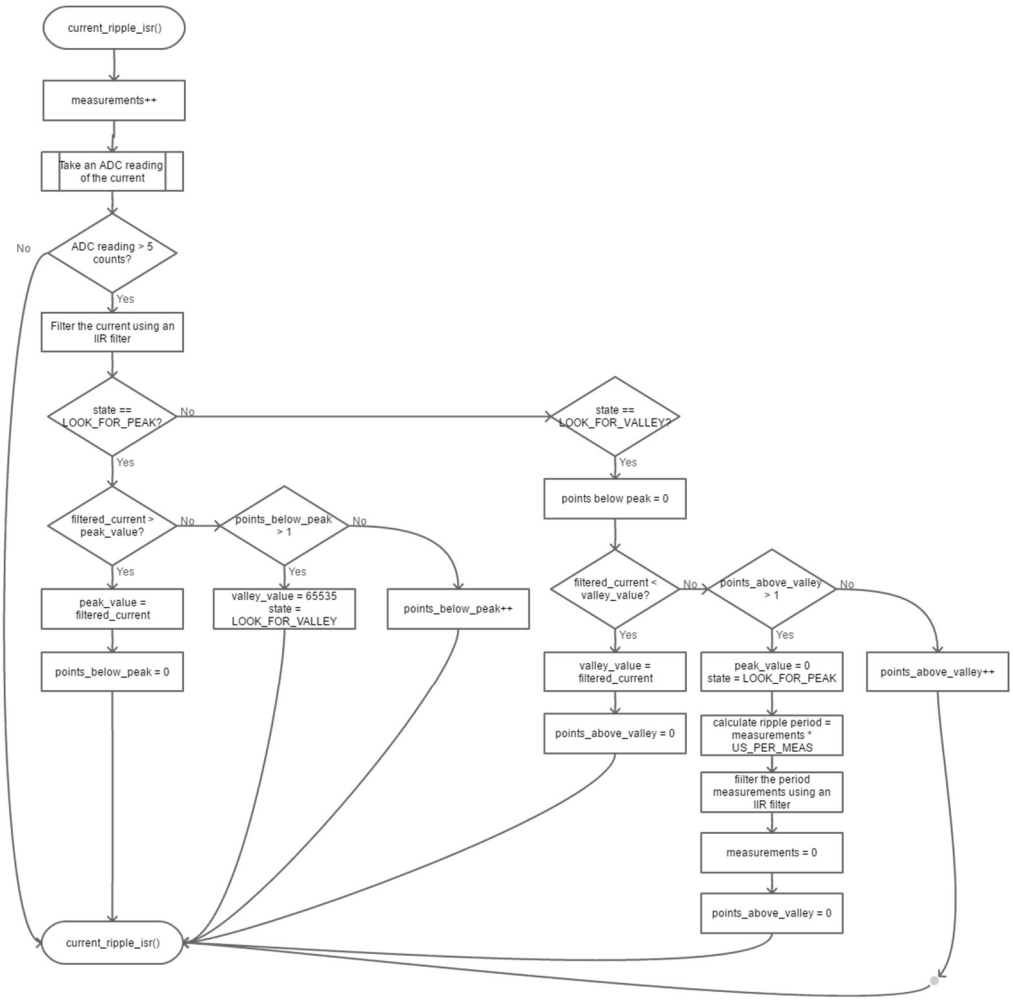

Figure 49: Current Ripple ISR Algorithm.

4.3 Automatic Calibration

Automatic calibration is a combination of the BEMF method and the current ripple method. This combined approach uses the current ripple method at low speeds to calibrate the BEMF method. This allows the BEMF method to be used at high speeds where the current ripple method is limited by the processor frequency.

When the system is powered on, the firmware will check to see if the tool has been calibrated. If the tool has been calibrated, the firmware will run the tool normally. If the tool has not been calibrated, the firmware will set the output speed to two levels (a low speed and a higher speed) where the current ripple method is accurate.

At both speeds, BEMF voltage measurements and the current ripple speed values are taken. The slope of the BEMF versus speed is calculated and used to calibrate the BEMF method.

This transforms the BEMF method into a design where the motor constant does not need to be known. The flowchart for the automatic calibration is shown in Figure 50.

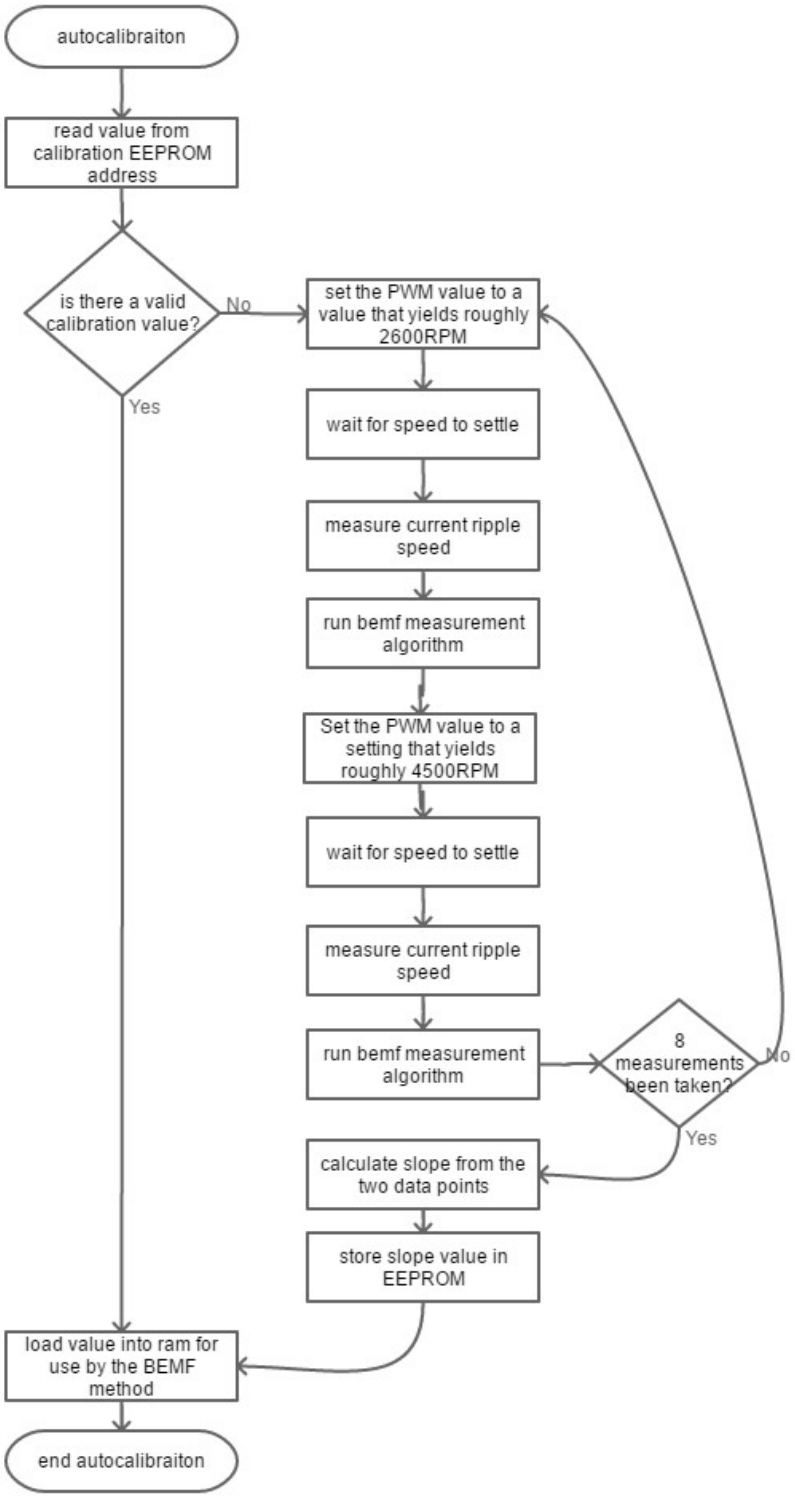

Figure 50: Automatic Calibration Flowchart.

Automatic calibration was tested using both the hall encoder as a reference and the current ripple as a reference. The hall encoder speeds were set to 2,600 RPM and 4,500 RPM. The current ripple calibration speeds were also set to 2,600 RPM and 4,500 RPM. The results are summarized in Table 3.

Table 3: Automatic Calibration Results.

Trial	Hall Encoder Calibration $[(\mathbf{RPM}) \times \mathbf{2^9/Counts}]$	Current Ripple Calibration $[(\mathbf{RPM}) \times \mathbf{2^9/Counts}]$
1	34546	32238
2	34546	33085
3	37106	34998
4	34546	34468
5	35400	32102
6	35400	34870
7	36253	32102
8	36253	31099
9	37106	31132
10	36253	31463
Average	**35741**	**32756**
Standard Deviation	**1002**	**1517**

The correct value for the motor used was approximately 34,894. The hall encoder calibration closely approximated this value with an average calibration value of 35,741. The current ripple method yielded a similar calibration value result of 32,756. The standard deviations between the hall encoder calibration and current ripple calibration were also very similar at 1,002 for the hall encoder and 1,517 for the current ripple method.

Based on these results, the automatic calibration is possible and the BEMF method can now be implemented without guessing the motor constant of the

particular motor paired with a particular set of electronics at the time of manufacturing.

Chapter 5: Results and Discussion

5.0 Automated Test Bench Results

5.0.1 BEMF Method

The error results for the BEMF method are summarized in Table 4.

Table 4: BEMF Results.

Parameter	Result	Specification	Units
Average Relative Error	329.1	<1,000	[RPM]
Average Percent Error	8.87	N/A	[%]

The average absolute error is well within the specification of less than 1,000 RPM. The average relative percent error was 8.87%.

The Automated Test Bench full results for the entire test are shown in Figure 51. In the bottom graph, the blue (Channel 1) is the speed calculated using the BEMF method and the orange (Channel 2) is the directly measured speed using the hall encoder. The top two graphs show the relative error in RPM (left), and the relative percent error (right).

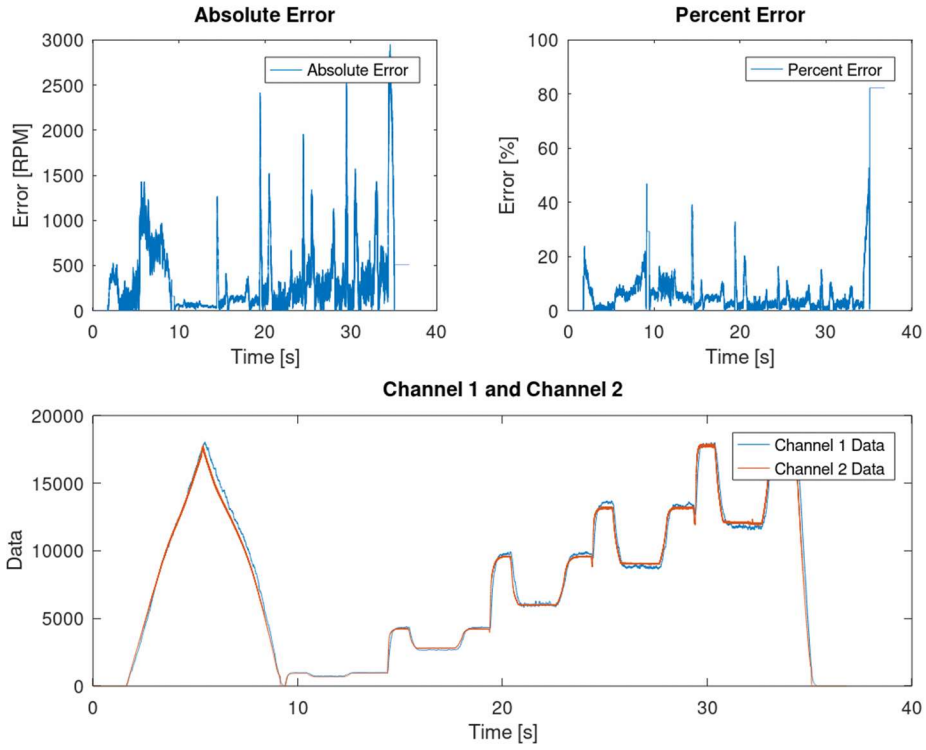

Figure 51: Automated Test Fixture Results for BEMF Method.

Overall, the BEMF method tracks the hall encoder relatively well. However, on rapid changes in speed, the BEMF method has some delay due to the low-pass filtering. Decreasing the filter length any further starts to introduce too much noise. The settings configured in this run produce the result with the least average error.

5.0.2 Inductive Pulse Duration Method

When the inductive pulse method was tested on the automated test bench, it was very apparent that the mapping at no-load was not applicable to loaded conditions. The no-load mapping worked well at no-load, but not at all under

load. The test results are shown in Figure 52. Channel 1 is the inductive pulse calculated speed and Channel 2 is the actual motor speed.

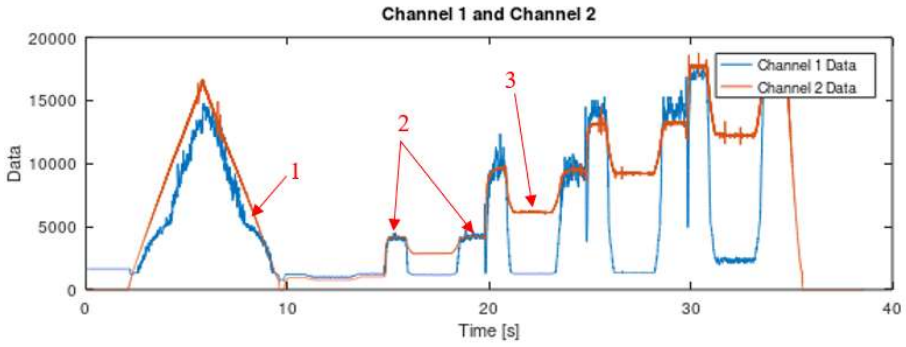

Figure 52: Automated Test Bench Results.

There are several interesting results labeled in Figure 52.

1. The inflection point seen in the no-load speed ramping is due to the piecewise linear approximation of the exponential relationship. At the point highlighted, the slope and intercept switch from one mapping to another.
2. During the steady-state no-load portions of the test, the mapping works well. During the fast ramp test, the calculated speed lags the actual speed, but still works reasonably well.
3. During the loaded tests, the pulse duration to speed mapping does not work at all.

The test was rerun measuring the pulse duration during the automated test bench. Interestingly, the relationship between pulse duration and motor speed is

not only dependent on the speed of the motor, but on how the speed of the motor is generated.

There are two ways the speed of the motor can be changed. The first method is by changing the effective applied voltage to the motor. This is done by reducing the duty cycle, effectively “chopping” the applied voltage. The second way the speed can be changed is by applying a load to the motor. This increases the torque demand and shifts the motor speed according to the speed-torque relationship of the motor.

The pulse duration is a function of motor current. The no-load speed change is a function of changing the applied voltage, not the load current. Therefore, even though the current in the motor changes, it is changing in accordance with the effective applied voltage, not the load.

In contrast, when the motor load is increased at a constant effective applied voltage, the corresponding change in current is proportional to the applied load. This effect is shown in Figure 53.

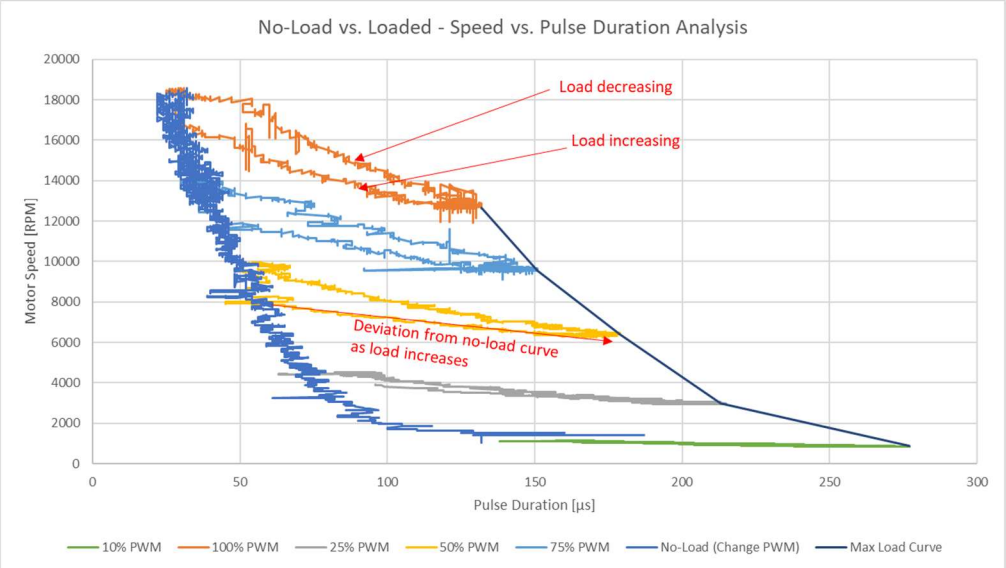

Figure 53: Motor Speed and Pulse Duration Relationship.

Note: The difference in load decreasing and load increasing is due to the software low-pass filtering. The maximum load curve is based on the maximum fixture load of approximately 30A at 100% duty cycle.

In order to accurately map the speed/pulse duration relationship, a three-dimensional surface of load current, duty cycle, and motor speed values would have to be used. Although this type of conversion is technically possible given a powerful enough processor, it is not feasible in low-cost microcontrollers like the ATMega328P used in this project.

Instead, three or four piecewise linear maps could be generated at different load currents and used to get a slightly better speed approximation without going to a full three-dimensional table of values. This method would require a current measurement. However, the amount of curve fitting approximation required, and

the much poorer accuracy than the BEMF method make this method unattractive for implementation in an actual product.

5.0.3 Current Ripple Method

The current ripple method is limited in the speed range where it can be accurate. On the low-speed end of the spectrum, the current ripple enters the amplitude noise threshold around 1,600 RPM. This is where the amplitude of the current signal is indistinguishable from the noise of the sense lines.

On the high-speed end of the spectrum, the algorithm stops working when the ripple frequency increases to about on eighth of the sample frequency. This corresponds to a speed of around 3,000 RPM at a sample frequency of 4kHz.

While the system is operating between 1,600 RPM and 3,000 RPM, the current ripple method has an average of 0.5% error relative to the hall encoder. Figure 54 shows the current ripple method compared to the hall encoder. The current ripple method is in blue and the hall encoder method is orange.

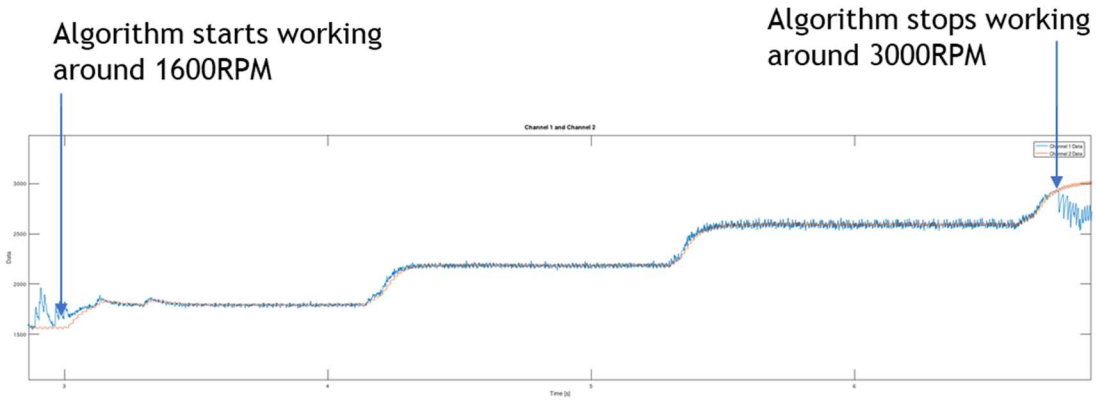

Figure 54: Current Ripple Speed Accuracy.

Because the system is running test code to handle the hall encoder method and send data to a PC over the UART, the maximum sample frequency is limited to 4kHz. Removing these functions could allow the sample frequency to increase to at least 8khz. This would allow the current ripple method to work all the way up to at least 6,000 RPM.

Using an external 16MHz crystal oscillator allowed the sample frequency to be increased to 8kHz. As expected, the range in which the current ripple method increased from a maximum speed of 3,000 RPM to a maximum speed of 6,000 RPM. Figure 55 shows the accuracy of the current ripple method sampling at 8kHz.

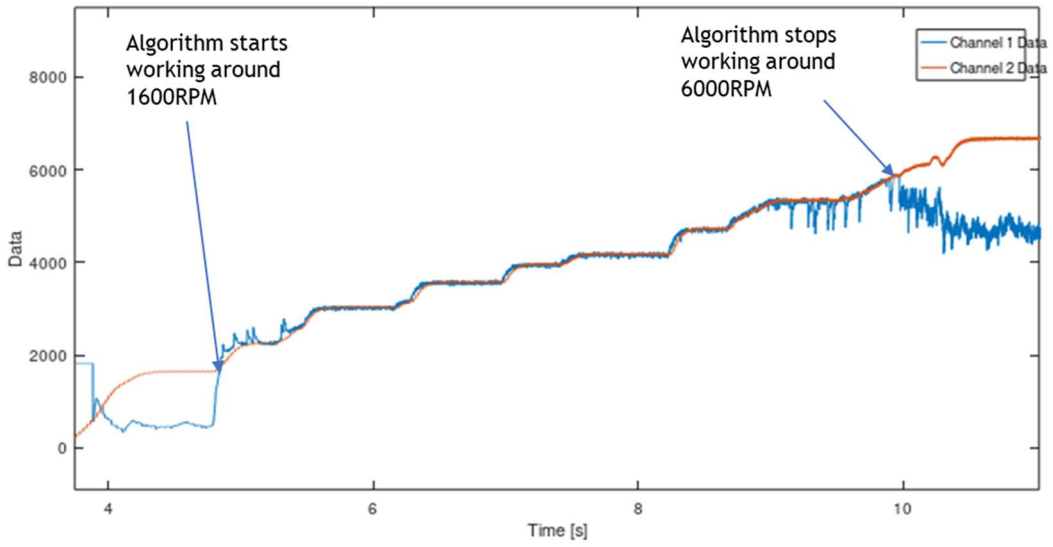

Figure 55: Current Ripple Algorithm Results at an 8kHz Sample Frequency.

In its current form, this method is accurate enough across a wide enough PWM range to use the current ripple method to calibrate the BEMF method. In this way, the BEMF method can be used without any parameterization of the

motor required. This system will automatically calibrate the BEMF method using the ripple current method on first power-up.

5.1 Automated Test Bench with Closed-Loop Speed Control

5.1.1 BEMF Method

Closed-loop speed control was used in conjunction with the auto-test routine to test the performance of a BEMF-based closed-loop speed control method and compare it to the performance of the traditional hall encoder-based method. For each test, the PI controller was allowed one second to stabilize. After the one second had passed, the auto test fixture switched in and out all ten loads in the same sequence that was used for the open loop speed tests.

Retuning of the PI controller constants was allowed for each method. Figure 56 shows the PI controller parameters and the final tuning runs for the hall encoder method, and the parameters for the BEMF method.

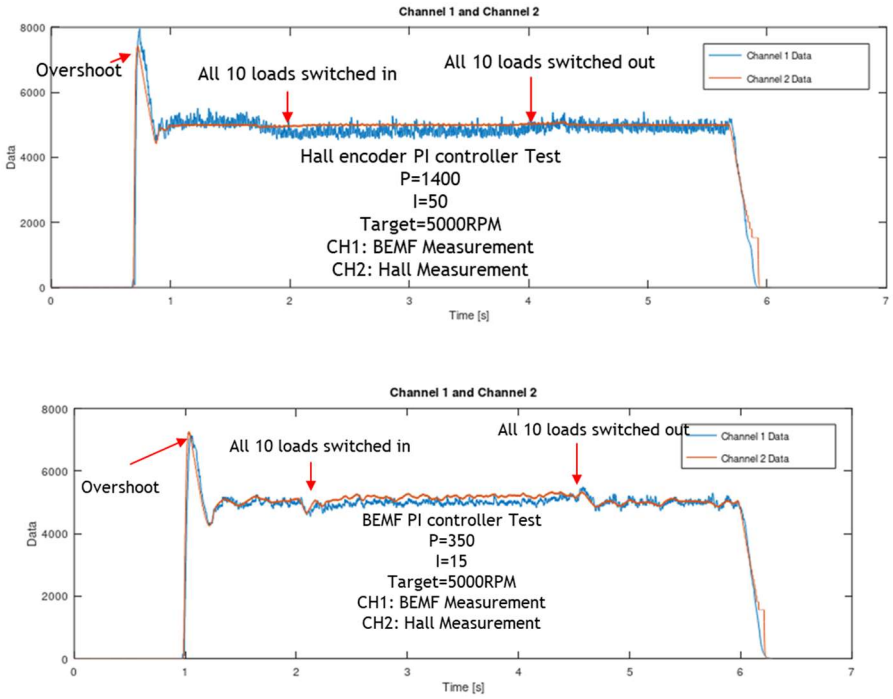

Figure 56: PI Parameters for Hall (Top) and BEMF (Bottom) PI Controllers.

Figure 53 shows the performance of the hall encoder-based method in this test, when the target was set to 5,000 RPM. Figure 57 shows the performance of the BEMF-based method in this test with the same target speed.

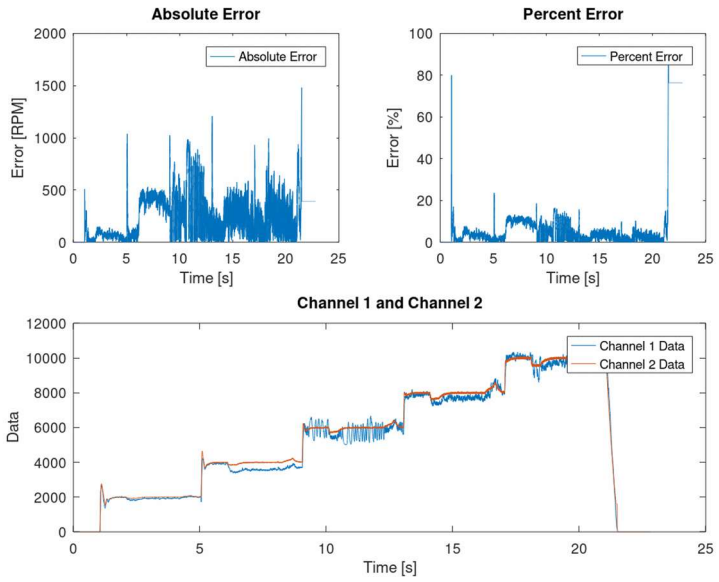

Figure 57: Closed-Loop Speed Control Performance (Hall Encoder).

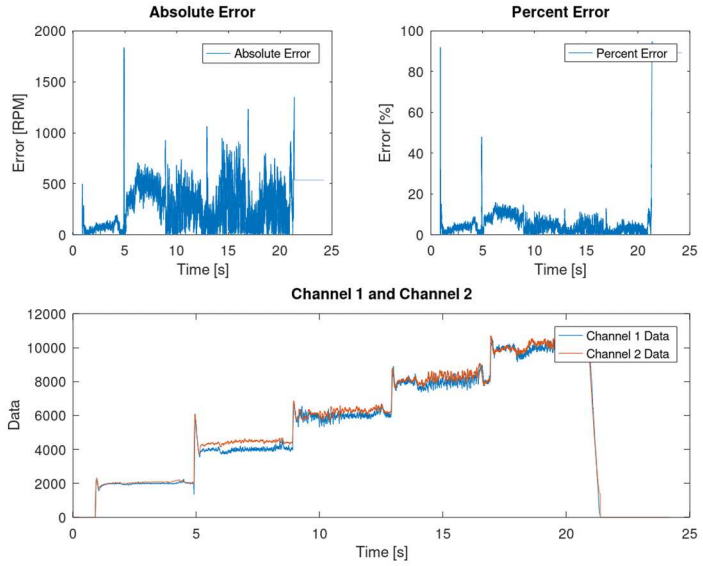

Figure 58: Closed-Loop Speed Control Performance (BEMF Method).

The BEMF method did have more oscillation in the test than the hall encoder method. Additionally, the PI constants could not be as aggressive as the

hall encoder method, or the system would become unstable. However, this method did keep the speed deviation within the allowable 1,000 RPM range of the target speed. The results of the closed-loop speed control tests are summarized in Table 5.

Table 5: BEMF Closed-Loop Test Summary

Parameter	Result	Specification	Units
Average Steady State Relative Error	201.6	<1,000	[RPM]
Average Steady State Percent Error	4.06	N/A	[%]

5.1.2 Inductive Pulse Duration Method

The inductive pulse duration method was not able to achieve stability when used as the feedback for the closed-loop speed PID controller.

Various proportional, integral, and derivative gain values were used to attempt to achieve stability. The PID update rate was slowed down to every twenty milliseconds instead of every single millisecond. Even with that adjustment, the PID was still unstable.

The issue is that the inductive pulse method is a very sluggish measurement method. Even under no-load conditions, the speed oscillates significantly. This effect is shown in Figure 59. Channel 1 is the inductive pulse speed estimate and Channel 2 is the actual motor speed (measured using the hall encoder).

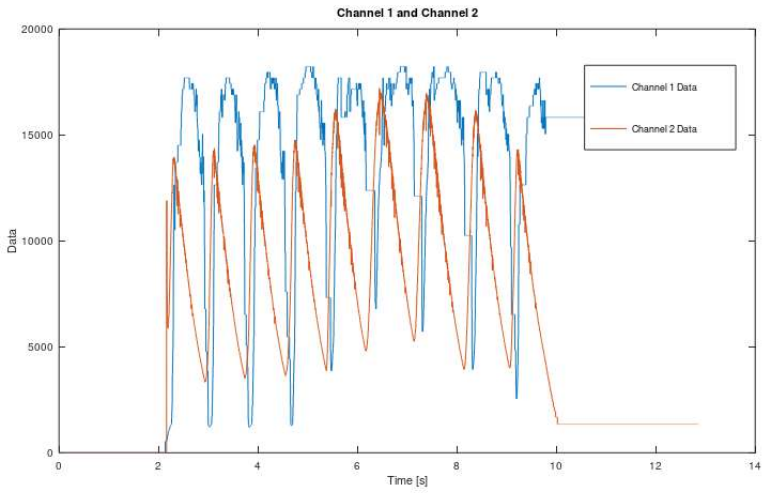

Figure 59: Speed Oscillation when PID is Controlled by the Inductive Pulse Method.

The no-load testing has shown that the inductive pulse method is not suitable as a PID feedback input in its current form. As a result, no further loaded testing was performed using this method.

5.1.3 Current Ripple Method

Because the current ripple method was limited by the microcontroller frequency, it was not effective in a wide enough range to perform the automated bench tests. If a more expensive higher frequency processor were used, this method could be very effective. This is an area that would be interesting for future research with higher frequency processors.

5.2 In-Tool Application Results

Two common Multitool applications were selected to evaluate the performance of the various methods in real-world scenarios. The first application consisted of four consecutive plunge cuts into half-inch drywall. The second application featured sanding plywood for ten seconds. Every two seconds, the pressure applied to the tool was alternated between heavy pressure and no pressure. Plunge cutting drywall is a light load application for the tool. Heavy pressure sanding is a heavy load application.

For each method, the hall encoder speed measurement was used to evaluate the performance. The PID controller was controlled by each of the methods and by the hall encoder for reference. The average relative error and maximum relative error for each method in each application was recorded.

The speeds chosen to run the tests were 5,000 RPM, 7,000 RPM, 9,000 RPM, 11,000 RPM, 13,000 RPM, and 15,000 RPM. A speed of 5,000 RPM was chosen as the minimum because it was the minimum speed that could perform complete plunge cut applications. A speed of 15,000 RPM was chosen as the maximum so that in heavy sanding applications, the tool would not reach its maximum power (which would cause error to the target that isn't due to the methods). The target was to be within five percent of the target speed on average throughout the application.

Figure 60 shows the tool with the two attachments that was used for application testing.

Figure 60: Test Tool with Sanding and Plunge Cutting Attachments.

Figure 61 shows the in-tool application data collection test setup. The tool was connected to a PC via the same UART-to-USB interface used in the fixture testing. Updates to the control method were made by reprogramming the ATMega328P microcontroller using the standard Atmel programming interface.

Figure 61: Application Test Setup.

Figure 62 shows the test tool plunge cutting into half-inch drywall.

Figure 62: Plunge Cutting Drywall.

Figure 63 shows the sanding application. In this testing, the tool was sanding plywood.

Figure 63: Sanding Application.

5.2.1 BEMF Method – Plunge Cutting

The motor speed over time for each of the RPM settings are shown in Figure 64. The speed shown is from the hall encoder. This was used to accurately measure how far off the BEMF measurement controlling the PID was (the BEMF calculated speed is the reference speed fed into the PID controller).

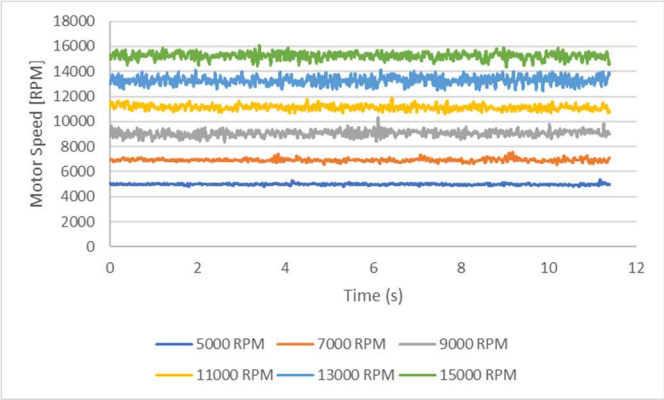

Figure 64: Plunge Cutting Application Time Data (BEMF Method).

For comparison, the hall encoder-controlled method data for the same plunge cutting applications are shown in Figure 65.

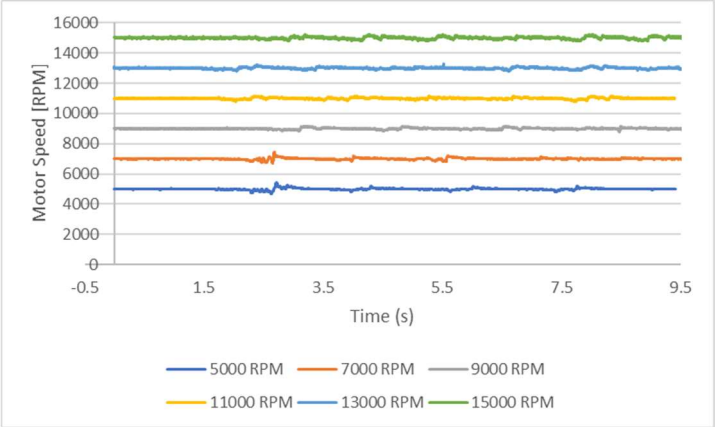

Figure 65: Plunge Cutting Application Time Data (Hall Encoder Method).

The average relative percent error for the duration of each test is shown in Figure 66. The error is relative to the set target speed. Note: The average hall encoder error is also shown, but the average error was on the order of 0.01%, so the bars are not visible. The average error for the BEMF method was higher than the hall encoder method. But the error was still well below the five percent target in all cases.

Figure 66: Average Relative Error – Plunge Cutting.

Figure 67 shows the maximum percent error observed at any point relative to the target speed. The results for the hall encoder method are shown on the same plot for reference (no target is associated with this graph).

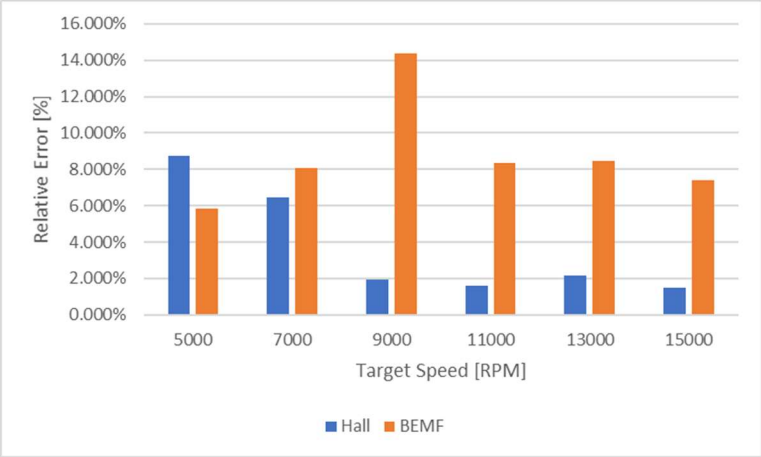

Figure 67: Maximum Error – Plunge Cutting.

5.2.2 BEMF Method – Sanding

The time data for the BEMF controlled method is shown in Figure 68. The legend shows the target speed for each application.

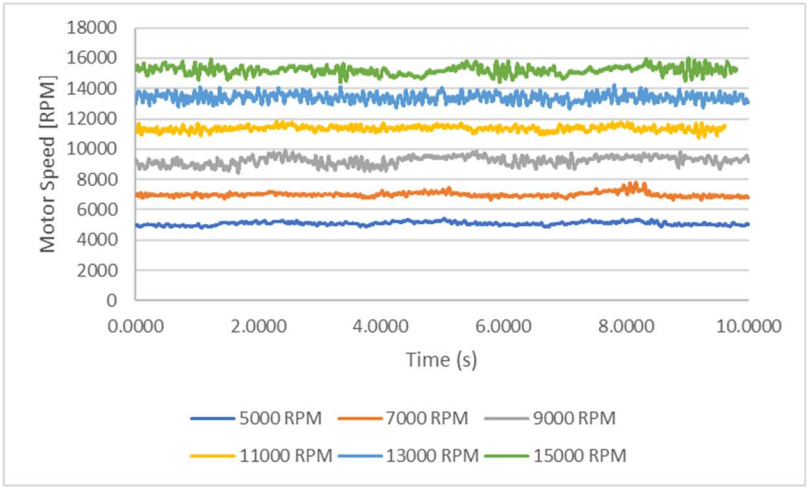

Figure 68: Sanding Application Time Data (BEMF Method).

The hall encoder-controlled time data are shown in Figure 69 for reference.

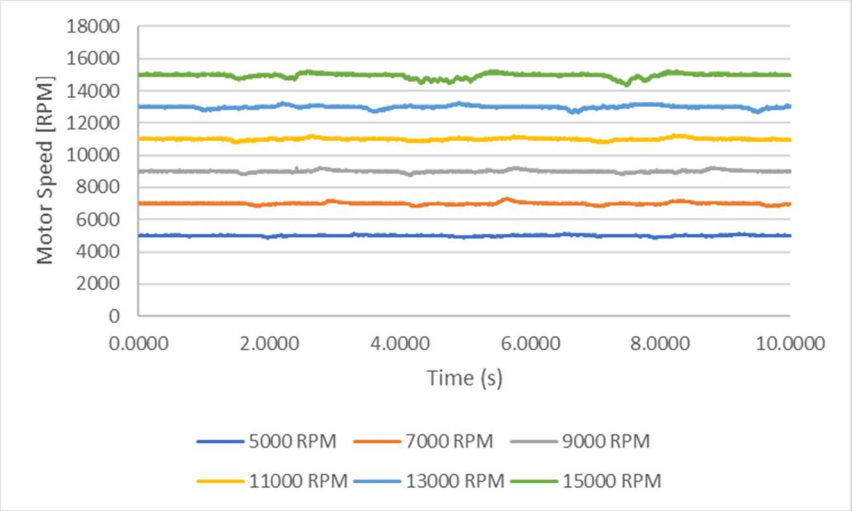

Figure 69: Sanding Application Time Data (Hall Encoder Method).

Because the sanding application has higher load swings and higher maximum loads than the plunge cutting application, the error was worse for both the hall encoder method and the BEMF method.

Figure 70 shows the relative error for each application in each speed target setting. In all cases, the BEMF method is below the five percent relative error stretch target.

Figure 70: Sanding Application Average Relative Percent Error.

The maximum relative percent errors observed in all the tests are shown in Figure 71. There is no target associated with these data; the data are for reference only.

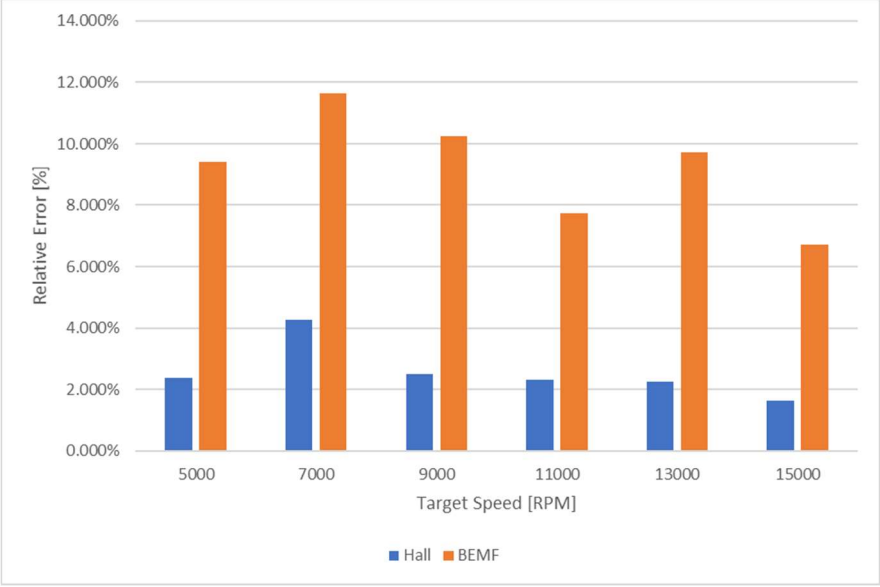

Figure 71: Sanding Application Maximum Relative Percent Error.

5.2.3 Inductive Pulse Duration Method

The inductive pulse method was not able to achieve stability in the closed-loop no-load tests. Therefore, the application tests could not be performed for this method.

5.2.4 Current Ripple Method

Because the current ripple method was limited by the microcontroller frequency, it was not effective in a wide enough range to perform the in-tool tests. If a more expensive higher frequency processor were used, this method could be very effective. This is an area that would be interesting for future research with higher frequency processors.

5.3 Discussion of Results

A summary of the various methods and their relative performance is shown in Table 6.

Table 6: Pew Matrix of Methods.

Parameter	BEMF	Inductive Pulse	Current Pulse
Motor Specific Mapping Required?	Yes	Yes	No
Number of Mapping Parameters Required	1	18	2
Auto Test Percent Error [%]	8.87	Note 1	0.5 Note 2
Auto Test Absolute Error [RPM]	329	Note 1	120 Note 2
Closed-Loop Speed Control Test Percent Error	4.06	Note 3	Note 3
Application Test Percent Error [%]	3.35	Note 3	Note 3

Note 1: Method was not accurate under load.

Note 2: Method only accurate between 1,600RPM and 6,000RPM.

Note 3: Test not performed due to limitations of the method implementation

The BEMF method was the only method that was able to successfully perform all the required tests and meet all the specification criteria. The inductive pulse method was very inaccurate in loaded conditions due to the pulse width dependency on both load and duty cycle. The current pulse method was the most accurate of all the methods. But the processor clock speed limited the effective range of this method to between 1,600 RPM and 3,000 RPM.

Chapter 6: Conclusions and Recommendations

6.0 Conclusions

The development and evaluation of novel implementations of the three known methods for sensorless speed detection did yield a design that met the initial project criteria. The BEMF method design met the initial design criteria and only required one mapping parameter be known for the implementation to work properly.

The BEMF method had very good accuracy across all speed ranges and loads. Additionally, the response time of this method was fast enough that a PID controller could use the output of the BEMF speed calculation as the error signal and remain stable.

The inductive pulse method had several major drawbacks. Many parameters were required to map pulse duration to motor speed. The relationship between motor speed and pulse duration is exponential. This is difficult to accurately map in software with only integer math operations available.

The relationship between inductive pulse duration and speed is a function of both applied duty cycle and the load applied to the motor. The implementation used in this project mapped the relationship at no-load. If this method were used across a wider range of load points, additional mapping parameters would need to be added for each desired load point. The quantity of parameter mapping required for this method to perform correctly makes it an unattractive solution for an embedded system with limited performance and memory constraints.

The response time of this method was also very slow compared to the update rate of the PID controller due to the large amount of signal conditioning and filtering required. As a result, the speed signal from this method is not suitable as the source of the error signal back into the PID controller, unless the PID update rate is substantially reduced.

The current ripple method was the most accurate of the three algorithms developed. It was the only design that was able to determine the motor speed through a direct measurement that did not require a conversion through a motor constant. It only required the number of poles and slots for the particular motor in the design be known.

This method has a limitation directly related to the speed of the processor used. In order to accurately detect the ripple current of the motor, a sample frequency of at least eight times the ripple frequency was required.

This means that not only must the ADC sampling frequency be eight times higher than the ripple frequency, but there must also be time left for the processor to handle the other tasks associated with running the embedded system. For an 8MHz microcontroller clock frequency the maximum motor speed that could be measured using this method was 3,000 RPM. Increasing the clock frequency could allow this method to be effective of a larger speed range.

Automatic calibration that combined the current ripple method and the BEMF method together was successfully implemented. This combined approach allowed the BEMF method to work across motor variation without requiring the

motor constant of each motor on the production line to be known. The tool will automatically detect the motor constant and store that value the first time that it is powered on.

6.1 Recommendations

The performance of the BEMF method makes it a good candidate for replacing hall encoder-based speed detection methods in PMDC power tool applications. It performed well in the automated tests as well as several application specific tests. Automatic calibration also eliminates the tuning of any motor parameters on the assembly line.

For embedded systems that have high clock speeds (greater than 32MHz) or low motor speeds (less than 3,000 RPM), the current ripple method would be the best solution.

6.2 Lessons Learned

When using external oscillators on microcontrollers, the system voltage may need to be increased for the microcontroller to stay in its safe operating area.

Performance of each of the methods increases when implemented on the final circuit board where the effect of parasitic components is much lower than on a breadboard.

6.3 Suggestions for Future Research

The current pulse method has a lot of promise in systems that already have microcontrollers with higher clock frequencies. As the performance of low-cost microcontrollers continues to increase, this method will likely become the best

option for sensorless speed detection on PMDC-based power tool embedded systems.

Mapping the inductive pulse method at different load points could be an interesting exercise to see how the method could perform under load. This effort would require significant three-dimensional mapping of the system parameters.

References

[1] T. Johnson, D. Grzybowski, M. Kubale, J. Rosenbekcer, K. Schucher, G. Meyer, J. Zeiler and K. Glasgow, "LIthium-Based Battery Pack for a Hand-Held Power Tool". United States Patent 7,554,290 B2, 30 June 2009.

[2] Dun and Bradstreet, "Power-Driven Handtool Manufacturing," First Research, Austin, 2019.

[3] Milwaukee Tool, "Milwaukee Tool Products," 2019. [Online]. Available: https://www.milwaukeetool.com/Products. [Accessed 28 6 2019].

[4] C. Conrad, Interviewee, *Vice President of Product Management, Milwaukee Tool.* [Interview]. 3 July 2019.

[5] Milwaukee Tool, "Internal Milwaukee Tool Data," Milwaukee Tool, Brookfield, 2019.

[6] S. Kamdar, H. Brahmbhatt, T. Patel and M. Thakker, "Sensorless Speed Control of High Speed Brushed DC Motor by Model Identification and Validation," in *IEEE - NUiCone2015*, Ahmedabad, 2015.

[7] Texas Instruments, "Automotive Brushed-Motor Ripple Counter Reference Design for Sensorless Position Measurement," Texas Instruments, 14 June

2018. [Online]. Available: http://www.ti.com/tool/TIDA-01421. [Accessed 8 July 2019].

[8] J. Vejlupek, R. Grepl, M. Matejasko and F. Zouhar, "Automotive Fuel Pump Fault Detection Based on Current Ripple FFT and Changes in Magnetic Field," *International Journal of Systems Applications, Engineering & Development,* vol. 7, no. 3, pp. 130-138, 2013.

[9] J. C. Walls, "Brushed Motor Controller Using Back EMF for Motor Speed Sensing, Overload Detection and Pump Shutdown, for Bilge and Other Suitable Pumps". United States of America Patent US20080258663, 23 October 2008.

[10] Precision Microdrives, "Application Note AB-021 Measuring RPM from Back EMF," October 2015. [Online]. Available: https://www.precisionmicrodrives.com/content/ab-021-measuring-rpm-from-back-emf/. [Accessed 18 June 2019].

[11] D. Kumar and P. Radcliffe, "Novel Sensorless Speed Measurement for Brushed DC Motors," *IET Power Electronics,* vol. 8, no. 11, pp. 2223-2228, 2015.

[12] E. Vazquez-Sanchez, J. Sottile and J. Gomez-Gil, "A Novel Method for Sensorless Speed Detection of Brushed DC Motors," *Applied Sciences,* vol. 7, no. 1, p. 14, 2017.

[13] B. Khoo, M. Mariappan and I. Saad, "A Sensorless Speed Estimation for Brushed DC Motor at Start-up," *International Joural of Innovative Science, Engineering & Technology,* vol. 3, no. 6, pp. 73-79, June 2016.

Bibliography

Ai, Z., Zhang, Ying, Zhang, Yingjie, Feng, Y., Murphey, Y., and Zhang, J., "Smart Pinch Detection for Car's Electric Sunroof Based on Estimation and Compensation of System Disturbance," *Control and Systems Engineering,* vol. 2, no. 1, April 2018.

Aydogmus, O. and Talu, M., "Comparison of Extended-Kalman- and Particle-Filter-based Sensorless Speed Control," *IEEE Transactions on Instrumentation and Measurement,* vol. 61, no. 2, pp. 402-410, September 2011.

Cupertino, F., Pellegrino, G., Giangrade, P., and Salvatore, L., "Sensorless Position Control of Permanent-Magnet Motors with Pulsating Current Injection and Compensation of Motor End Effects," *IEEE Transactions on Industry Applications*, vol. 47, no. 3, pp. 1371-1379, March 2011.

Ghosh, M., Ghosh, S., Saha, P., and Panda, G. "Sensorless Speed Estimation of Permanent Magnet DC Brushed Motor Considering the Effects of Armature Teeth-Slots and Commutation," *IET Power Electronics,* vol. 10, no. 12, pp. 1550-1555, October 2017.

Hilariret, M., and Auger, F., “Speed Sensorless Control of a DC-Motor via Adaptive Filters,” *IET Electric Power Applications,* vol. 1, no. 4, pp. 601-610, July 2007.

Microchip Technology Inc, “Application Note AN893 Low-Cost Bidirectional DC Motor Control Using the PIC16F684,” 2003. [Online]. Available: http://t-es-t.hu/download/microchip/an893a.pdf. [Accessed 11 July 2019].

Precision Microdrives, “Application Note AB-026 Sensorless Speed Stabiliser for a DC Motor,” October 2015. [Online]. Available: https://www.precisionmicrodrives.com/content/ab-026-sensorless-speed-stabiliser-for-a-dc-motor/. [Accessed 11 July 2019]

Appendixes

Appendix A – Data Collection Procedure

1. Connect the UART to USB converter to a USB slot on a PC.
2. Open RealTerm.
3. In the *Display* tab, change the *Display As* Selection to *Hex[Space]*.

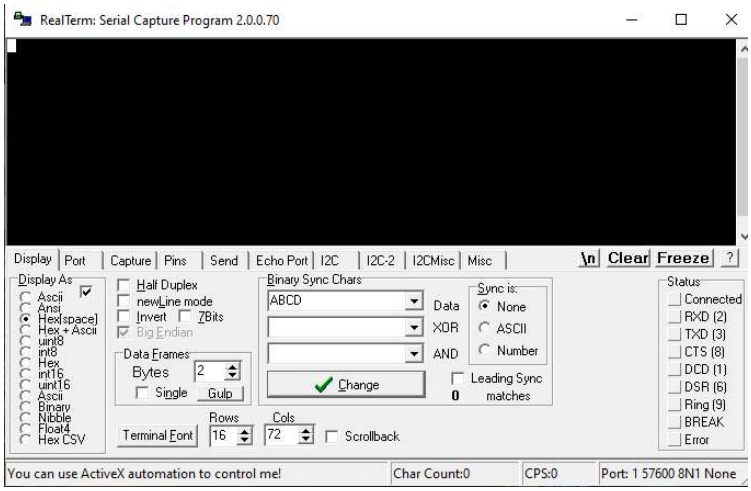

Figure A-1: Display Format.

4. In the *Port* tab, change the Baud rate to 76800. Change the port to the correct port for the USB to UART converter and select two stop bits.
5. Press *Change.*

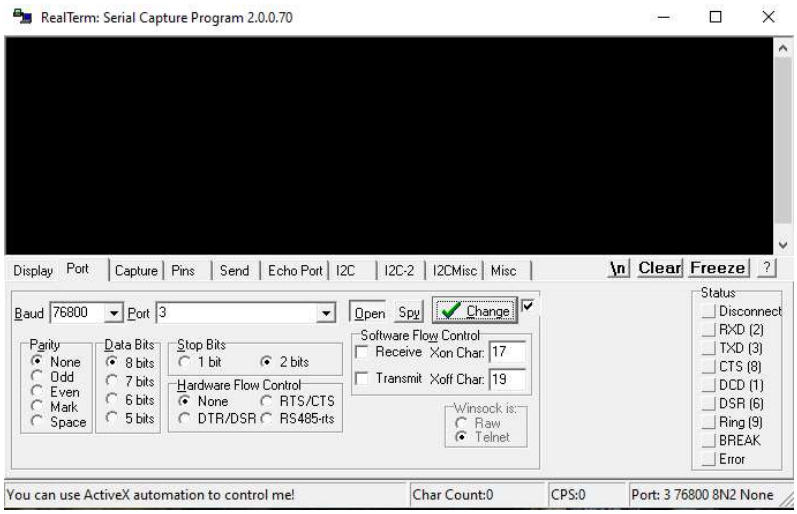

Figure A-2: Port Settings.

6. In the *Capture* tab, select *Capture as Hex.*

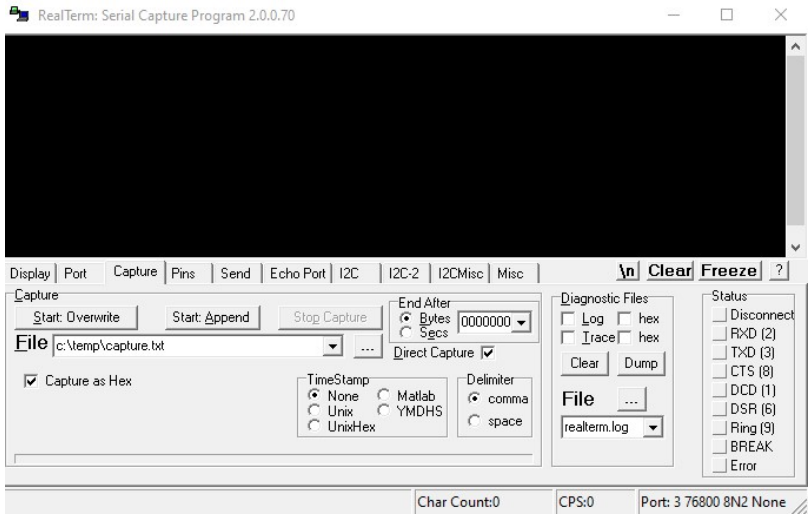

Figure A-3: Capture Settings.

7. When ready to capture data press the *Start Overwrite* button.
8. When ready to stop capturing data press the *Stop Capture* Button.
9. Place the *Hex_2_Dec.m* MATLAB file in the same folder as the data file (make sure it is named *capture.txt).*

10. Launch the *Hex_2_Dec.m* file and press the *Run* button.
11. The script will run and generate two .csv files for each of the parsed channels.

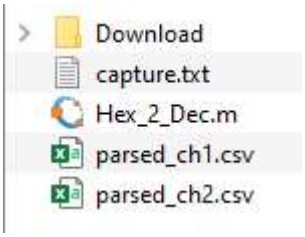

Figure A-4: Generated Parsed Data Files.

12. The script will also generate three graphs. One graphing both channels on the same graph and the other graphing the percent and absolute error.

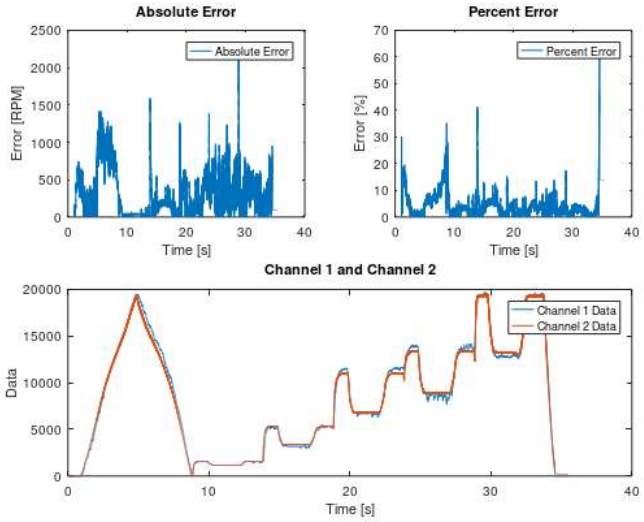

Figure A-5: Generated Graphs Comparing Channels.

Appendix B – Data Parsing MATLAB Script

The MATLAB data parsing and graphing script is shown in this appendix.

```
clear;
clc;
%Filename for data to read
filename = 'capture.txt';
% Read in data and format it
fidi = fopen(filename);
Data = char(textscan(fidi,'%s'));
fclose(filename);

%Undo SLIP Formatting
%Start by separating string into an array with the C0 delimiter
TempData1 = strsplit(Data,'C0');
%replace all DBDC entries with C0
TempData2=strrep(TempData1,'DBDC','C0');
%Replace all DBDD entries with DB
HexData=strrep(TempData2,'DBDD','DB');
%Find the size of the Hex Data
tempsize=size(HexData);
datasize=tempsize(2);

%Convert data from string hex to decimal
DecData = hex2dec(HexData);
datasize=size(DecData);

%Parse Data
j=1;
k=1;

for i=1:(datasize-1)
  p=sizeof(HexData(i));
  if p==6
    Nx = char(HexData(i));
    Nxc=mat2cell(Nx, 1, [2 4]);
    [channel, outdata]=Nxc{:};
    ch=hex2dec(channel);
    if ch == 1
      data1(j)=hex2dec(outdata);
      if j==1
        Time1(j)=0;
      else
        Time1(j)=Time1(j-1)+0.001;
```

```
      endif
      j=j+1;
    elseif ch == 2
      data2(k)=hex2dec(outdata);
      if k==1
        Time2(k)=0;
      else
        Time2(k)=Time2(k-1)+0.001;
      endif
      k=k+1;
    endif
  endif
endfor

PercentError(1) = 0;
AbsoluteError(1) = 0;
ErrorTime(1) = 0;
if k>j
  finalsize = j-1;
else
  finalsize = k-1;
endif
  for i=2:(finalsize)
    ErrorTime(i) = ErrorTime(i-1)+0.001;
    if data2(i)>500
    AbsoluteError(i) = abs(data2(i)-data1(i));
    PercentError(i) = abs((data2(i)-data1(i))/data2(i))*100;
    else
    PercentError(i) = PercentError(i-1);
    AbsoluteError(i) = AbsoluteError(i-1);
    endif
 endfor

%Save data to csv files
DataMatrix1 = [Time1', data1'];
DataMatrix2 = [Time2', data2'];
csvwrite('parsed_ch1.csv',DataMatrix1);
csvwrite('parsed_ch2.csv',DataMatrix2);

%Plot Figures
subplot(2,2,1);
plot(ErrorTime,AbsoluteError);
title(['Absolute Error']);
xlabel('Time [s]');
```

```
ylabel('Error [RPM]');
legend('Absolute Error');

subplot(2,2,2);
plot(ErrorTime,PercentError);
title(['Percent Error']);
xlabel('Time [s]');
ylabel('Error [%]');
legend('Percent Error');

subplot(2,2,[3,4]);
plot(Time1,data1,Time2,data2);
title(['Channel 1 and Channel 2']);
xlabel('Time [s]');
ylabel('Data');
legend('Channel 1 Data','Channel 2 Data');

AverageAbsoluteError = mean(AbsoluteError)
AveragePercentError = mean(PercentError)
```