DESIGN AND CONTROL OF A HIGH-PRECISION PERMANENT-MAGNET AC MOTOR USING A HALBACH MAGNET ARRAY

A Thesis

by

BRADFORD WILLIAM STRICKLIN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Won-jong Kim
Committee Members,	Sivakumar Rathinam
	Hamid Toliyat
Head of Department,	Andreas Polycarpou

December 2017

Major Subject: Mechanical Engineering

ABSTRACT

This work presents, designs, models, and tests a two-phase permanent-magnet (PM) external-rotor AC motor utilizing a Halbach array for the purpose of precision positioning. The motor is made up of twelve coils and nine pitches of the Halbach array and controlled through the use of a National Instruments data acquisition (DAQ) board paired with MATLAB's DAQ toolbox. The motor makes uses of the Halbach array and the lack of any ferromagnetic materials in the system to ensure the internal magnetic field is sinusoidal. The motor takes advantage of this fact to generate a torque that is independent of both the rotor position and current distribution. Whereas most PM motors rely on the requirement that the coil windings are sinusoidally distributed, this motor is able to compensate for not having sinusoidally distributed coil windings by instead adjusting the current phase vector based on rotor positioning.

After characterizing the motor and through the utilization of data collected during the normal operation of the motor, and several procedures designed to account for discrepancies in the timing of the DAQ board's clocks, the response of the three embedded Hall-effect sensors were mapped to the position of the rotor. Linear sections of this mapping were then used to control the position of the rotor down to an accuracy of just a few hundred thousandths of a degree. Additionally, due to the inherent nature and stability of the system, the only controller required to achieve these results is a PID controller.

ACKNOWLEDGMENTS

I would firstly like to thank my committee chair, Dr. Won-jong Kim, for his guidance during this research. I would also like to acknowledge my committee members, Dr. Sivakumar Rathinam and Dr. Hamid Toliyat.

Special thanks go to Dr. Vu Nguyen, a former student of Dr. Kim, who had the initial idea for this research and without who, this project would not have been possible.

I would also like to thank Sean Zachary Roberson for his excellent LATEX template available through the Texas A&M University Office of Graduate and Professional Studies website.

Lastly, I would like to thank my friends and family for their encouragement and support.

CONTRIBUTORS AND FUNDING SOURCES

CONTRIBUTORS

This work was supported by a thesis committee consisting of Professor Won-jong Kim and Professor Sivakumar Rathinam of the Department of Mechanical Engineering and Professor Hamid Toliyat of the Department of Electrical Engineering.

The initial idea for this research comes from Dr. Vu Nguyen who also came up with an initial design and had even begun printing some of the outer parts of the chassis.

All other work conducted for the thesis was completed by the student independently.

FUNDING SOURCES

This work was financed independently.

NOMENCLATURE

AC	Alternating Current
EMF	Electromotive force
PM	Permanent Magnet
PMAC	Permanent Magnet Alternating Current
DC	Direct Current
PMBLDC	Permanent Magnet Brushless Direct Current Motor
NI	National Instruments
DAQ	Data Acquisition
B	Magnetic Flux Density
B_p	Peak Flux Density
T_{em}	Torque Produced
T_f	Friction Torque
k_T	Torque Constant
I	Current
I_p	Peak Current
r	Radius
N_s	Number of Coils Per Phase
N_w	Number of Wraps Per Coil
p	Number of Poles
l	length
ω	Frequency

t	Time
R	Resistance
L	Inductance
rpm	Revolutions Per Minute
rps	Revolutions Per Second

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Motivation

This thesis project's objective is to design and demonstrate the improved characteristics of a two-phase alternating current (AC) Halbach array motor over conventional motor designs. The characteristics that are to be improved are

- Torque profile – due to torque independence of positioning
    - Lack of cogging torque – due to the implementation of the Halbach array and non-ferrous building materials
        - Precision positioning – due to lack of cogging torque
- Torque strength — due to increased magnetic field strength from Halbach array

A motor that meets all of these objectives has potential in a variety of applications. Precision positioning is useful in manufacturing applications or in areas where gearing the shaft to achieve similar positioning precision may not be desirable. Improved torque rating is desirable in any motor application where high motor torque is important. Lastly, the lack of cogging torque means significantly smoother operation of the motor.

1.2 Novelty and Significance

A novel two-phase AC Halbach array motor has been designed and built to demonstrate the superior positioning characteristics obtained by building the motor out of 3-D printed plastic parts and the implementation of a Halbach array in the motor. The plastic parts virtually eliminate the cogging torque caused by the non-uniform attraction of the permanent magnets to the motor's iron stator slots. This is in addition to the prevention of eddy currents from being produced in the motor which cause power loss. These eddy

currents create their own non-uniform magnetic fields which would interfere with the motor's operation by causing torques of their own. Additionally, the Halbach array provides several desirable characteristics that aid in positioning. Though much work has been done on the analysis of the use of Halbach arrays in motors, the first significant contribution of this project is the construction and actual testing of the system. This is to say that while many papers have modeled Halbach array systems, this experiment collects real data. The second novel contribution of this paper is that though the improved torque characteristics of motors utilizing Halbach arrays has been explored, this project seeks to take this a step further by applying these torque benefits towards accurately positioning the motor.

1.3 Halbach Array

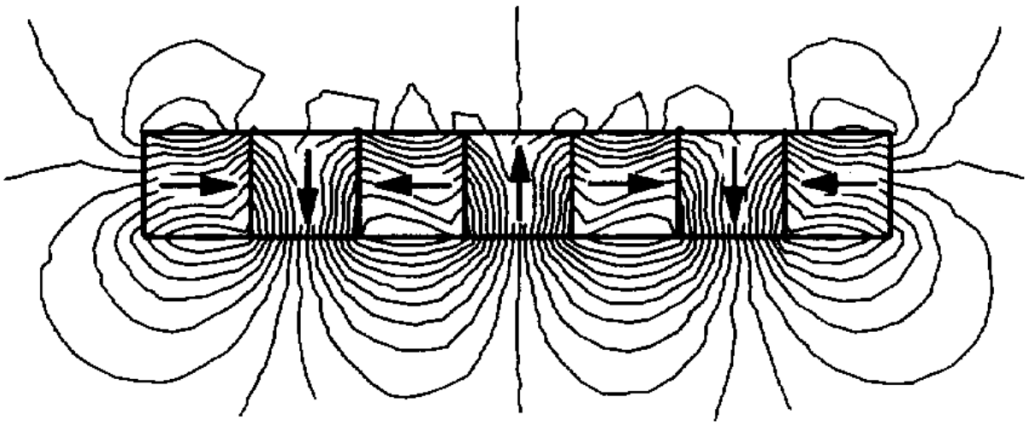


Figure 1.1: Linear Halbach array magnetic field (©1993 IEEE)[1].

In conventional permanent magnet (PM) motors, the permanent magnets are arranged such that a north-south pattern is followed where each magnetization vector in the se-

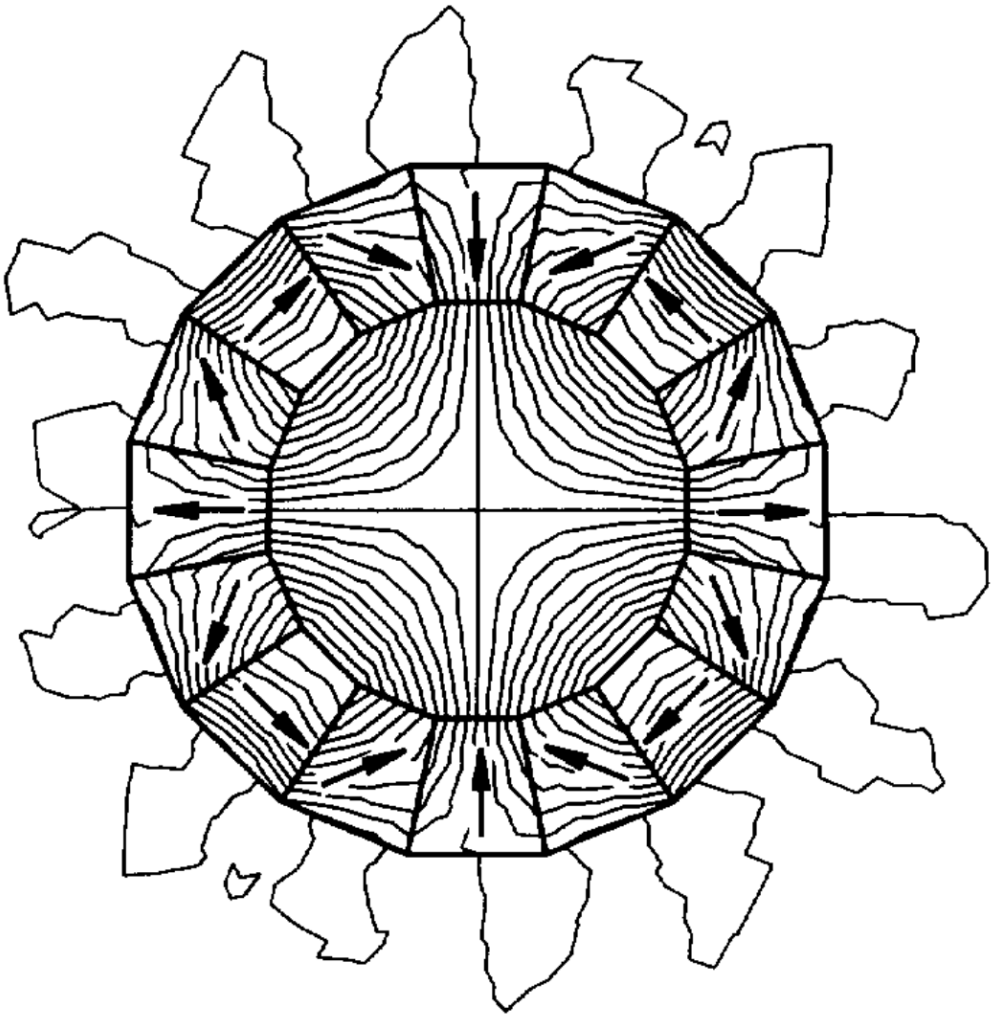


Figure 1.2: Radial Halbach quadrupole magnetic field (©1993 IEEE)[1].

quence is rotated 180°. Through the implementation of a Halbach array, the magnetic field can be concentrated on one side of the array of magnets. This has the effect of increasing the strength of the magnetic field on one side of the array as compared to a conventional magnetic array [3]. This idealized effect can be seen in Figure 1.1 where the magnetic field is represented by the lines with arrows. The concentrated magnetic field lends itself to a greater torque produced in a motor, a more ideally sinusoidal magnetic field, and a

more predictable back electromotive force (back-EMF) [4]. This type of design can also be implemented in a radially oriented Halbach array as can be seen in Figure 1.2.

1.4 Electric Motor

The non-ideal nature of motors results in periodic changes in torque output, also known as torque ripple. This fact implies that the torque output of a motor is dependent on the position of the motor's rotor. Even in synchronous motors, torque ripple remains a prominent effect and is most commonly associated with the cogging torque from the interaction of the PMs and the stator's iron slots [5]. As this torque pulsation decreases motor performance and increases wear on the motor, much work has been done to mitigate these effects [6–9]. While these efforts focus on magnet shaping or skewing the teeth of the motor, the underlying problem remains present [10]. Variations in torque caused by the nonuniform attraction of the PMs to different parts of the motor create torque ripple. This is of particular concern in motors whose operating frequency can align with the mechanical resonance frequency. In an ideal PMAC motor, these torque fluctuations do not exist as the magnetic field is perfectly sinusoidal and is unaffected by the motor. In practice, however, the cogging torque can be expected to be about 2% of the rated torque for a conventional Halbach array motor (compared to about 10% of the rated torque for a conventionally magnetized motor) [3]. It should be noted that cogging torque reductions to about 0.3% of the rated torque have been reported in some PM systems due to skewing of the stator teeth [11]. The motor design presented in this thesis more accurately represents an idealized motor as the Halbach array provides a more idealized sinusoidal magnetic field and the plastic components do not interfere with the magnetic field unlike the metal components of a conventional motor. Additionally, the motor presented here does not rely on an assumed sinusoidal current distribution around the stator.

In order to facilitate the planning of the motor design, the method that Mohan outlines

in his book for calculating the current space vectors of a PMAC motor will be used in conjunction with independent analysis [12]. The full derivation can be found in Section 2.2. To this end, the torque generated by a PMAC motor can be found to be independent of angular positioning and the torque equation can be found to be

$$T_{em}(I) = k_T I \tag{1.1}$$

where

$$k_T = 2N_w N_s l r B_p \tag{1.2}$$

1.5 Prior Work on Precision Halbach Array Motors

Despite the improved torque-fluctuation performance that comes with the implementation of the Halbach array, additional work has been done to further smooth the torque in rotary Halbach array systems. This has been done by reducing the cogging torque through magnet skewing and also by developing a slotless PM brushless DC (PMBLDC) motor [13, 14]. Though this second case specifically points out the use in precise positioning applications, no data or calculations regarding relative positioning accuracy are offered. In other words, despite the wide range of applications, this area of Halbach motor development has been neglected.

Halbach arrays have been used successfully in linear motors to achieve higher force-density motors [15–17]. Additionally, the use of the Halbach array means the field will cause less interference and thus require less shielding in the system [18]. This in turn means the system has less mass and will have a better dynamic performance. These systems, which can have positioning accuracies within 0.5 nm (rms), are still limited to a fairly small stroke range and are thus relegated to fairly niche applications [19].

1.6 Applications of Halbach Array Motors

Motors, and PM motors in particular, are used in industrial applications, residential applications, and everything in between [20]. Given the wide range of application of motors in use today, the need for high-precision positioning systems is undisputed. Though servomotors are traditionally thought of as providing the best positioning characteristics when it comes to rotary motors, even with a rotary encoder of 4000 counts per revolution, a positioning error of 1 count corresponds to 0.09°. Though not common, these kinds of resolutions are possible with dual control systems [21]. Given the uniform torque potential and the lack of cogging torque, the system designed here could be used to provide even better angular precision. This is aided by the fact that the entire motor structure can be 3D printed to allow for reduced manufacturing costs. Combined with the decreased torque ripple of the system design and the system developed in this work has the potential for manufacturing applications that require high-precision.

Additionally, given the use of the Halbach array and the increased magnetic field strength brought about by this, the system could find use in electric-vehicle engines. PM motors are attractive in general to electric vehicle applications due to their high power density and efficiency [22]. Additionally, the fact that the torque is independent of both speed and positioning and therefore can immediately achieve a high starting torque is very desirable. The lack of torque ripple and constant torque potential would also be of particular interest in this application.

1.7 Summary of Work

This body of work explores the design and analysis of an external-rotor two-phase PM AC motor that makes use of a Halbach array in order to create a uniform torque potential that is independent of both the rotor's position and velocity. After this analysis, the construction of the motor and all supporting equipment will be discussed. This discussion will

include both the mechanical and electrical components. The motor will function through the use of MATLAB programs operating a National Instruments data acquisition board that controls transconductance amplifiers. After this, a series of experiments designed to characterize the motor will be presented. During this, the embedded Hall-effect sensors will be used to map the measured magnetic field to specific angular positions of the rotor. This series of experiments will support the explanation of algorithms developed to compensate for deficiencies in the hardware as well as develop a linearized model of this voltage to position mapping. Finally, a control scheme for precision positioning of the system will be developed and implemented. The results of which will be explored and used to tune the controller further.

2. DESIGN AND FINITE-ELEMENT ANALYSES

2.1 Initial Design

As the initial setup for this set of experiments was come up with by Dr. Nguyen, he specified the initial size of some of the mechanical parts. Its approximate size was kept because the dimensions proved to be of such a range that the motor would be ideal for testing and modification. That is, it would not be too large as to be cumbersome and could be easily moved while also being large enough that construction and working with the internal components would not be too difficult. Additionally, the initial idea of making the rotor an external rotor motor was kept so as to increase the radius the motor's force acts at, thereby increasing the torque of the motor.

The utilization of nine sets of Halbach arrays (for a grand total of thirty-six magnets) around the rotor was selected as 180° is divided nicely by nine. Not only this, but the magnet sizes and shapes that are easily available fit nicely into the dimensions of the motor. That is, for a smaller number of repeating Halbach arrays, the magnets would have had to be much larger. In the case of six Halbach arrays (the next smallest number that divides 180° evenly), there would have been twenty-four magnets around the rotor. This would have meant the magnets would have had to have been 50% larger in order to create a similar magnetic field. A decreasing number of magnets means that there is also a greater error between the circular profile from the magnets and an idealized circular magnetic array. On the other hand, increasing the number of Halbach arrays used means that a greater number of magnets are required. Given that the nine Halbach array magnets already cover about 72% of the rotor's inner circumference ($\frac{0.251 \cdot 36}{4\pi} = 0.719$), increasing the number of magnets for the 4-in inner diameter would require the reduction of the size of the magnets in order to allow space for both the magnets and the supporting plastic

material used to separate them.

With the number of Halbach arrays determined and the approximate sizing of the rotor, the magnets were next selected. The BX044-N52 magnets were chosen due to their large pole surface area and their residual flux density value of 1.48 T[1]. The magnet's field visualization can be seen in Figure 2.1. Based on this, assuming the air gap is a sufficiently small distance, the minimum magnetic flux density felt by the stator should be between 0.4553 and 0.5495 T (without considering the increased magnetic field from using the magnet in a Halbach array).

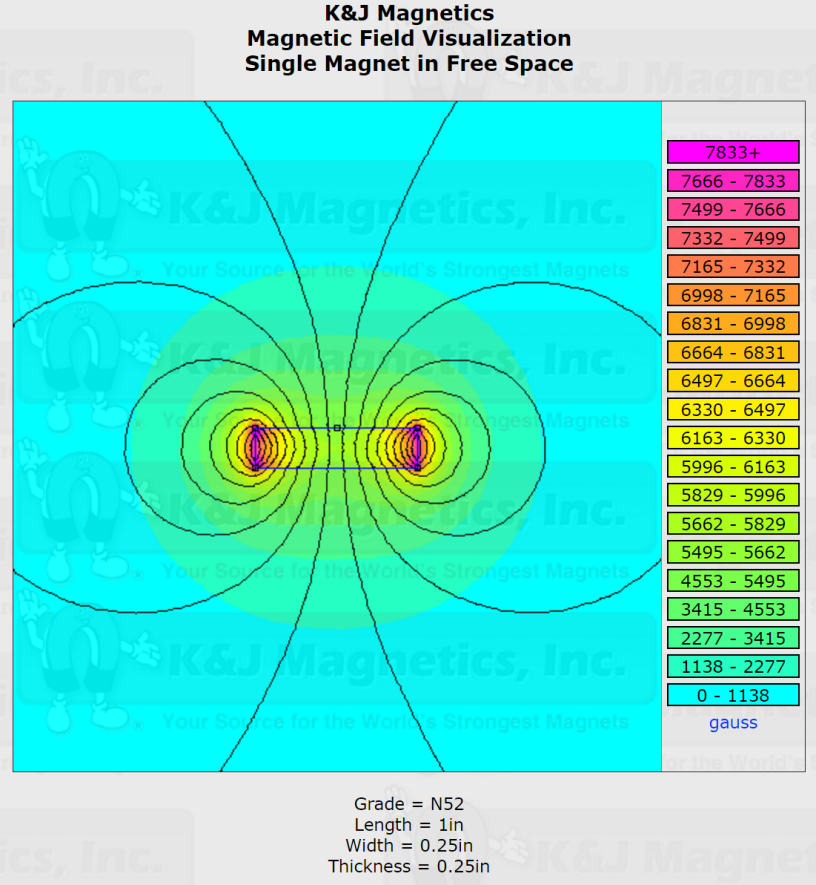


Figure 2.1: BX044-N52 field visualization [2].

[1]K&J Magnetics, Inc.,18 Appletree Ln. Pipersville, PA U.S.A.

Based on this approximate sizing of the stator and rotor, and the characteristics previously discussed, the motor's supporting infrastructure was designed. All of the drawings of the mechanical components of the motor can be found in Appendix B.

With the initial mechanical design planned out, the electrical characteristics next had to be determined. Based on the availability of resources, the stator's coils were hand-wound using 20 AWG NEMA MW136-C magnet wire [23]. The wire coating allows bundles of these wires to be easily fused together which facilitates coil wrapping. Based on the sizing of the wire and the designed gaps in the stator, three layers of eight coils apiece will be wrapped in each of the twelve slots of the stator. As the wire is 20 gauge, it is rated up to approximately 6 A [24]. As the stator contains twelve coils, the motor could theoretically have a number of phases equal to any factor of 12. For this series of tests however, the motor will be wired as a two-phase motor but the ends of the coils will be fed through the shaft to outside the motor where the ends of the coils could be rearranged so as to reconfigure the number of phases of the motor at a later date.

2.2 Torque Analysis

With the given design of the rotor, the magnetic flux density can be described according to the position around the rotor. That is

$$B(\theta_m) = B_p \cos(\frac{p}{2}\theta_m) \tag{2.1}$$

where p is the number of poles of the motor (18 in this case), the subscript p denotes the peak magnetic field, and the subscript m denotes the mechanical angle. As the motor is synchronous, the mechanical speed is directly related to the frequency of the signal by

$$\omega_e = \frac{p}{2}\omega_m \tag{2.2}$$

where the subscript e denotes the electrical frequency. In other words, rather than tracking the system based on the position around the ring, any arbitrary position on the rotor can be described as a function of time given some frequency. As

$$\theta_m = \omega_m t \tag{2.3}$$

this yields

$$B(t) = B_p \cos(\frac{p}{2}\omega_m t) \tag{2.4}$$

$$= B_p \cos(\omega_e t) \tag{2.5}$$

The stator current space vector is defined as

$$i_{sv}(t) = i_1(t) \angle 0° + i_2(t) \angle 90° \tag{2.6}$$

where the subscript *sv* denotes a space vector. Each current is given by

$$i_1(t) = I_p \cos(\omega_e t) \tag{2.7}$$

$$i_2(t) = I_p \cos(\omega_e t - \frac{\pi}{2}\frac{p}{2}) \tag{2.8}$$

With these equations developed, the expected torque generation of the motor can now be calculated, and the Lorentz force equation for a current-carrying wire will be used. That is,

$$\boldsymbol{F(t)} = i(t)\boldsymbol{l} \times \boldsymbol{B(t)} \tag{2.9}$$

As only the components of the current that are axially-directed along the outside of the stator will be considered, the current will always be perpendicular to the magnetic field. This means the force will also always be directed tangential to the motor and the equation simplifies to

$$F(t) = i(t)lB(t) \tag{2.10}$$

The l in this case is the length of the wires crossing the radial face of the stator for each phase. This means that each coil effectively counts as two wires along this face. Substituting in the equations found above yields

$$F_1(t) = I_p\cos(\omega_e t)l\mathrm{B}_p\cos(\omega_e t) \tag{2.11}$$

$$= I_p l B_p \cos^2(\omega_e t) \tag{2.12}$$

Recognizing that the offset between the two phases of the motor corresponds to $\frac{\pi}{2}\frac{p}{2}$ (just as in (2.8)) and the temporal offset for the magnetic field is the same, the equation for the force from the second phase is

$$F_2(t) = I_p l B_p \cos^2(\omega_e t + \frac{\pi}{2}\frac{\mathrm{p}}{p}) \tag{2.13}$$

Putting both forces back in terms of the mechanical speed yields

$$F_1(t) = I_p l B_p \cos^2(\frac{p}{2}\omega_m t) \tag{2.14}$$

$$F_2(t) = I_p l B_p \cos^2(\frac{p}{2}(\omega_m t + \frac{\pi}{2})) \tag{2.15}$$

As this shows the constant phase offset of the second phase, it can be rewritten as

$$F_2(t) = I_p l B_p \sin^2(\frac{p}{2}\omega_m t) \tag{2.16}$$

Adding the forces from both phases results in

$$F_{tot}(t) = I_p l B_p[\cos^2(\frac{p}{2}\omega_m t) + \sin^2(\frac{p}{2}\omega_m t)] \tag{2.17}$$

$$= I_p l B_p \tag{2.18}$$

Given that there are six coils per phase with each coil being made of twenty-four wraps that each have a side length of 2.54 cm (1 in), the final force equation becomes

$$F_{tot} = 7.3152 I_p B_p \tag{2.19}$$

Assuming the air gap is negligible and the outer radius of the stator and the inner radius of the rotor are equivalent yields a torque equation of

$$T_{em} = 7.3152 r I_p B_p \tag{2.20}$$

This yields a torque independent of both position and time so long as the current vector is rotated accordingly. It is significant to note that unlike most PMAC motors no assumption was needed regarding the distribution of current around the stator. That is, where most derivations require a sinusoidally distributed current around the stator, the current distribution of this two-phase motor is unimportant so long as the current is regulated according to the speed of the motor [12]. In this case, an approximately uniform distribution of current results in an approximately uniform distribution of force on the rotor but other than this, the distribution is unimportant. Though this Lorentz force is calculated as acting on

the stator, this force is equivalent to the one felt by the rotor. As the stator is fixed, this results in the rotation of the rotor.

2.3 Magnetostatic Analyses

Through the use of the software Maxwell SV, magnetostatic simulations were performed in order to verify the characteristics of the Halbach array implementation as well as to calculate the expected strength of the internal magnetic fields within the motor. These simulations were performed using N35 NdFeB magnets as opposed to the N52 NdFeB magnets that are actually used in the motor. This grade differentiation is a reflection of the maximum energy product of the magnet that is a good indicator of the strength of the magnetic field. Though the grades are similar in that they share a similar magnetic coercivity value of about 877 kA/m, the distinction is important because N52 magnets have a residual flux density that is about 20% higher than N35 magnets [25]. This means that the simulations will represent a lower bound for the expected system performance but can still be used as an indicator for expected conditions.

In Figure 2.2 the simulation is performed on a linear magnet array where every magnet in sequence has their pole flipped by 180°. That is that every other magnet down the array has their north pole upwards and the magnets in between these have their south pole placed upwards. The magnetic field near the array can be more clearly seen in Figure 2.3. In Figure 2.4 a linear Halbach array is modeled and Figure 2.5 shows the corresponding near-field view. Figure 2.6 is the simulation results from a radially configured array of magnets utilizing the same configuration as from Figure 2.2 and overlayed into the magnet positions in the motor's rotor. In other words, Figure 2.6 represents the configuration from Figure 2.2 but wrapped into a circle where every magnet was placed in a proper location for the motor being analyzed. A closer perspective of this configuration can bee seen in Figure 2.7. Figure 2.8 is the results of a simulation in which a Halbach array was used to

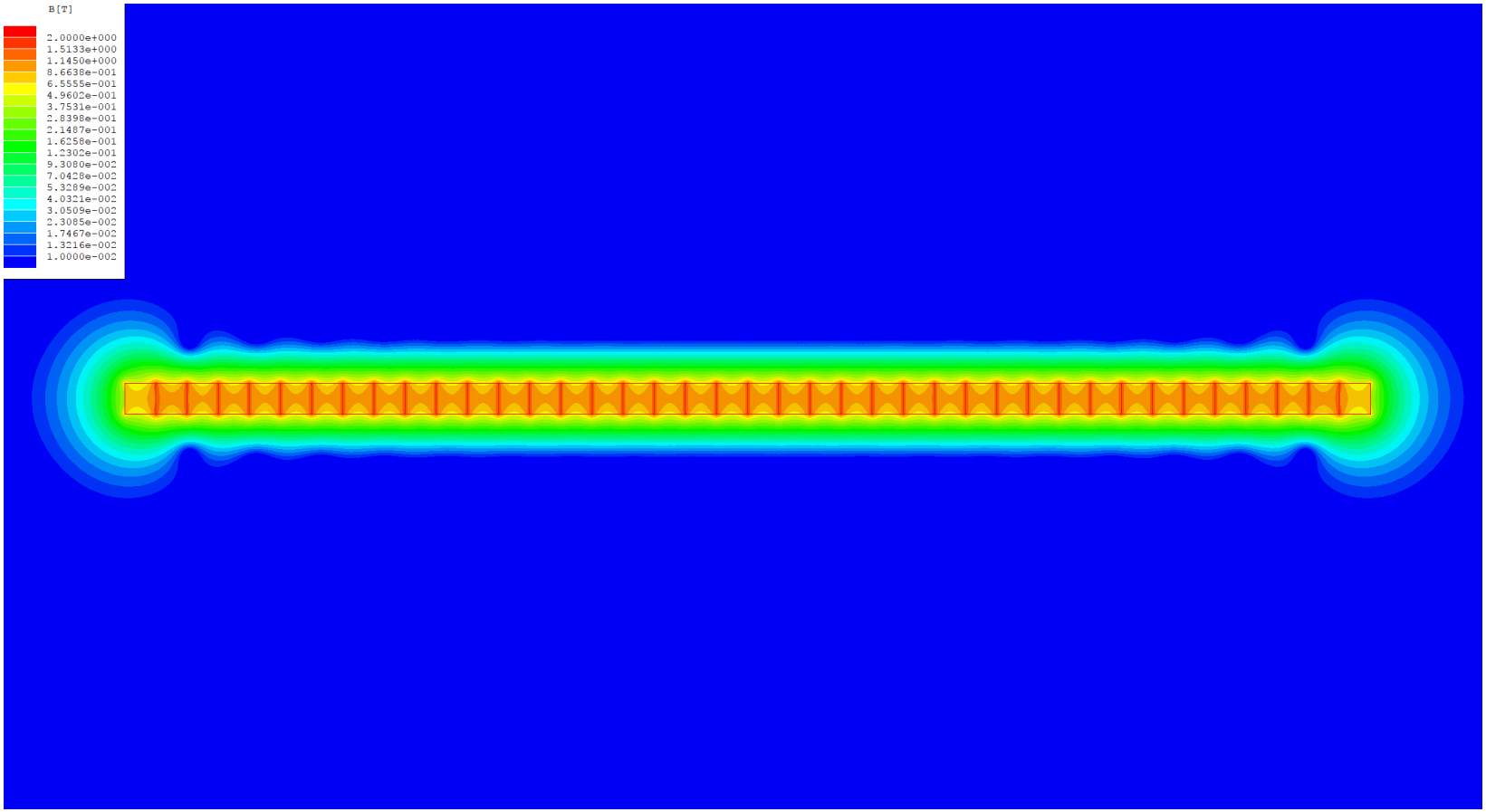


Figure 2.2: Linear magnet array with conventional north-south configuration.

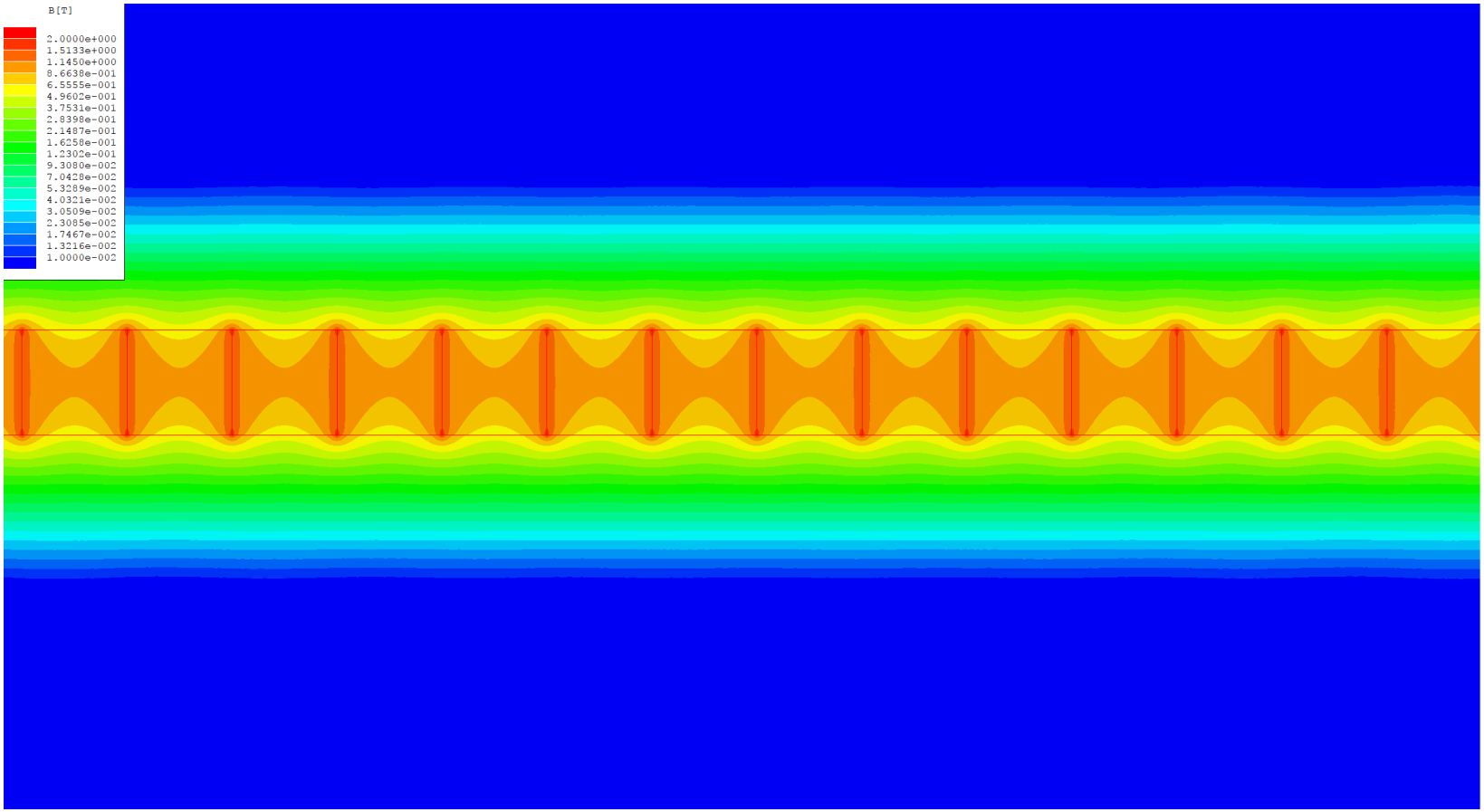


Figure 2.3: Close-up of linear magnet array with conventional north-south configuration.

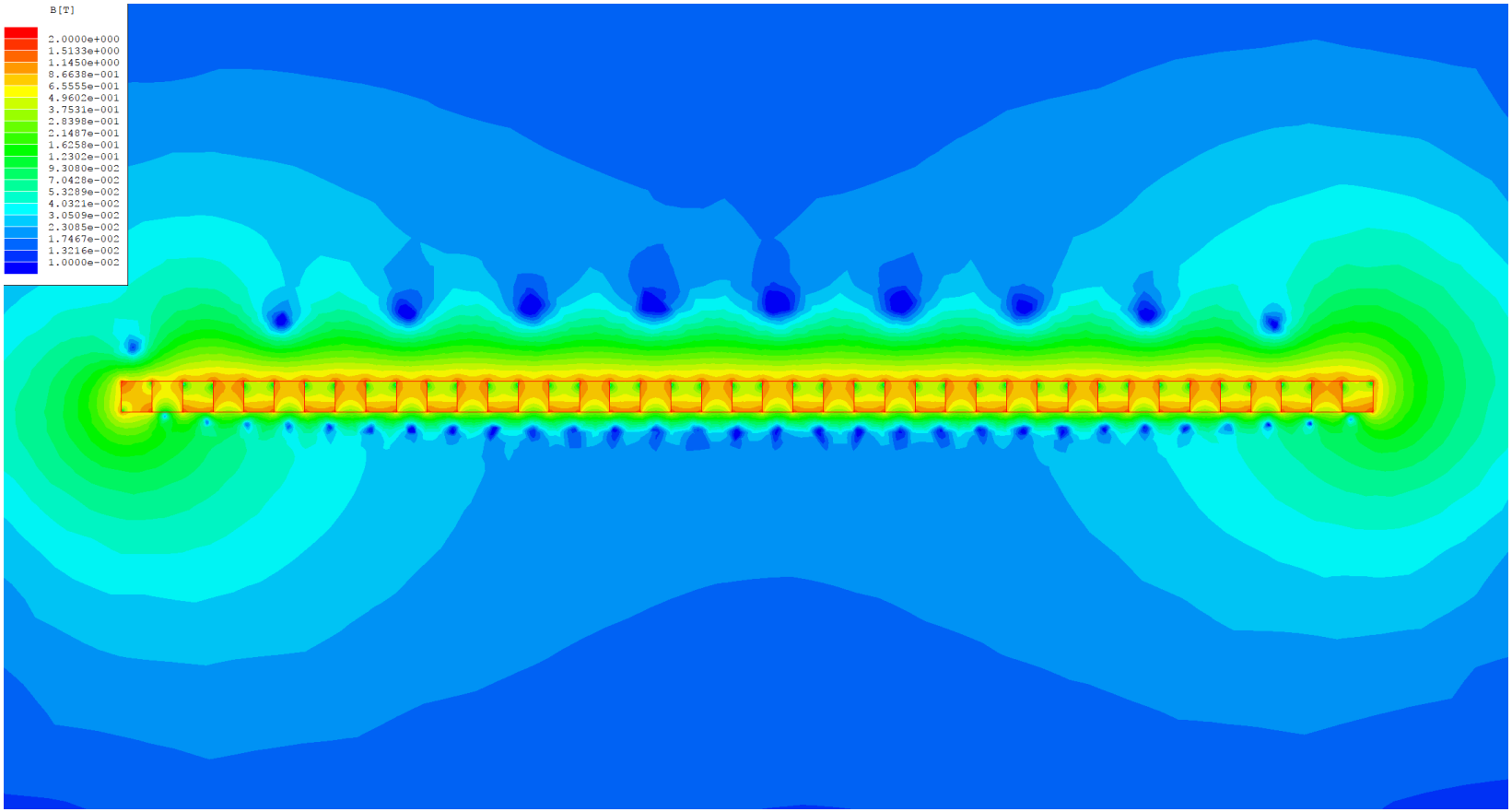


Figure 2.4: Halbach array magnetic field.

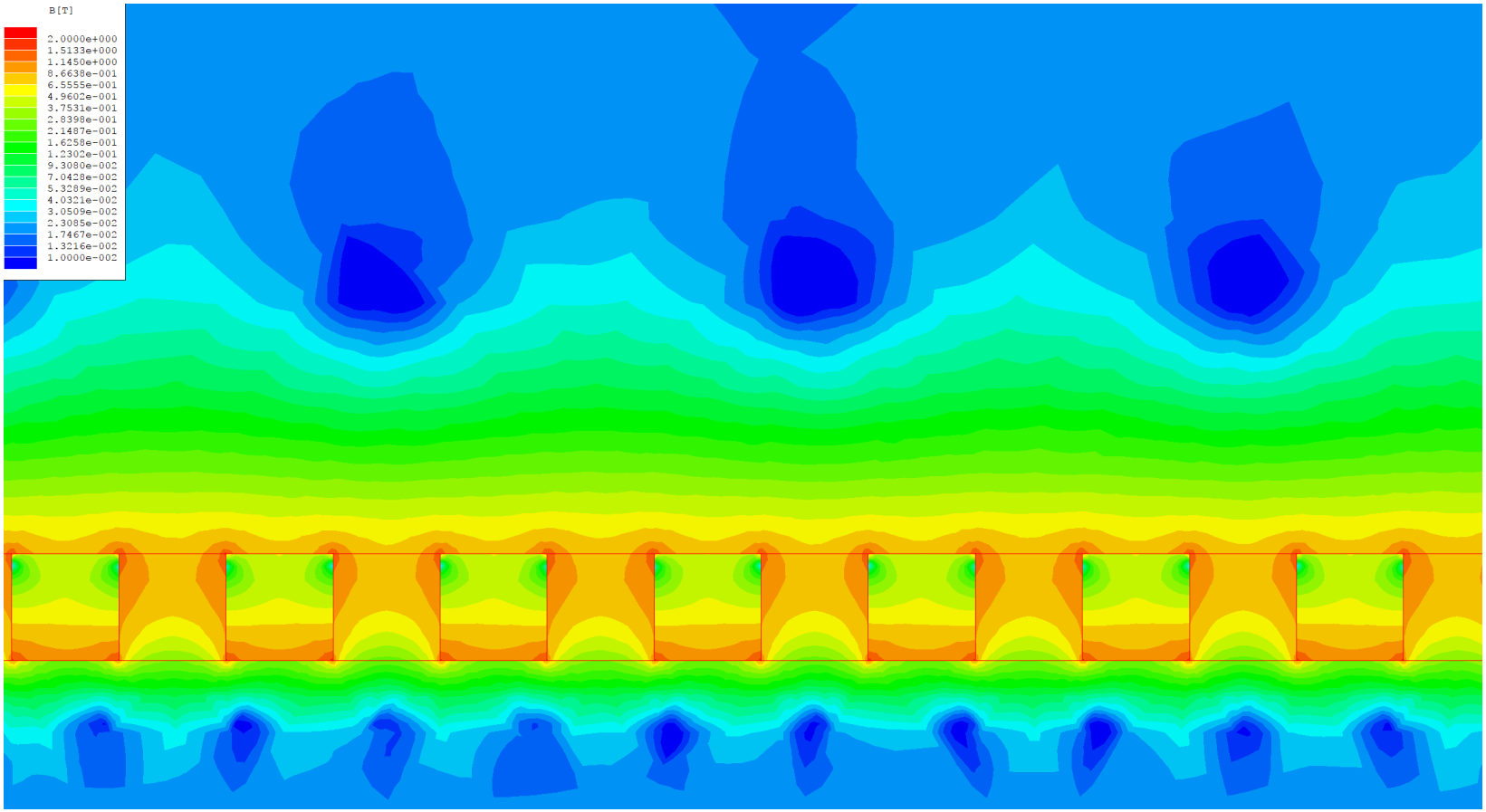


Figure 2.5: Close-up of Halbach array magnetic field.

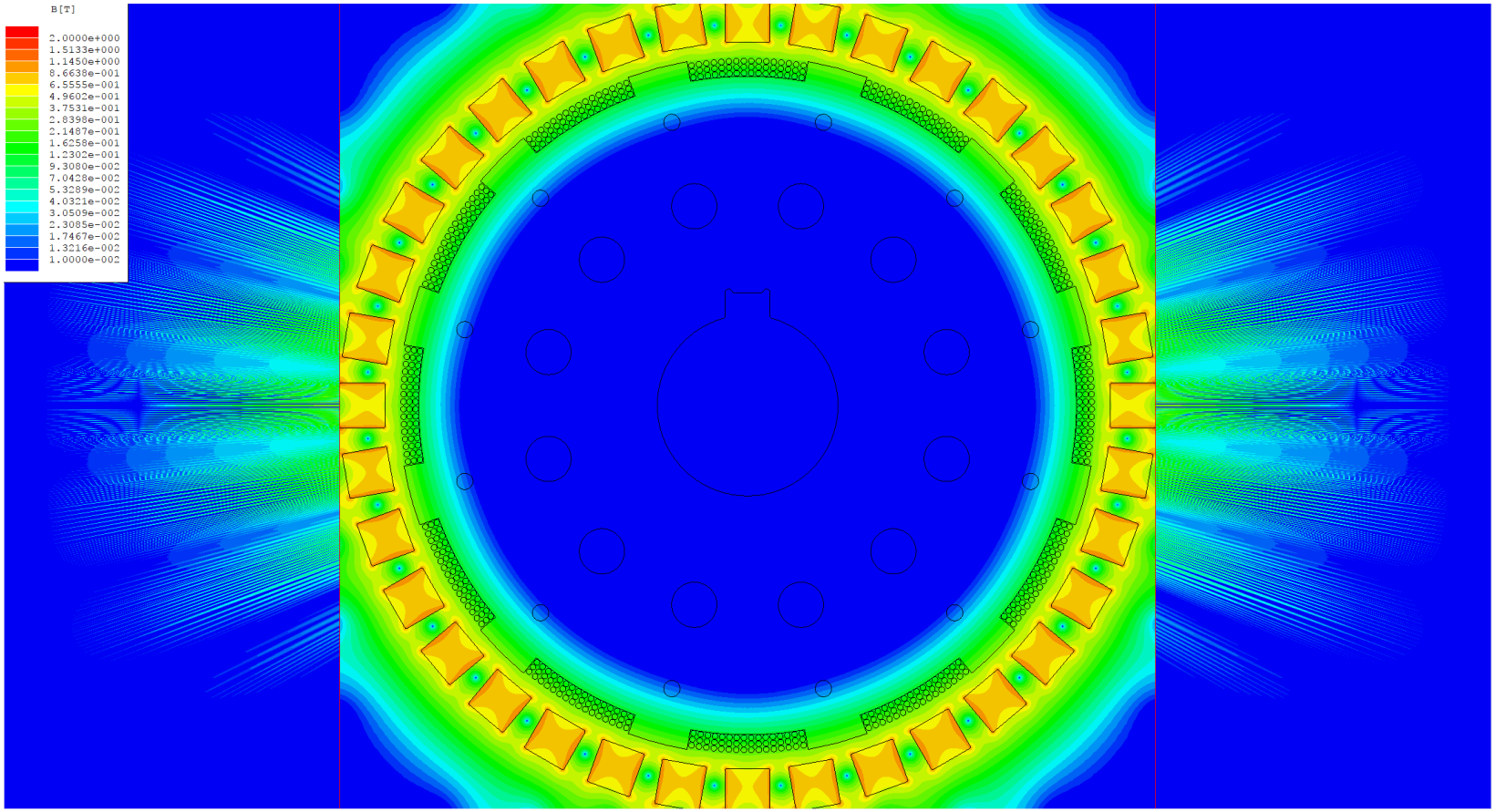


Figure 2.6: Circular magnet array with conventional north-south configuration.

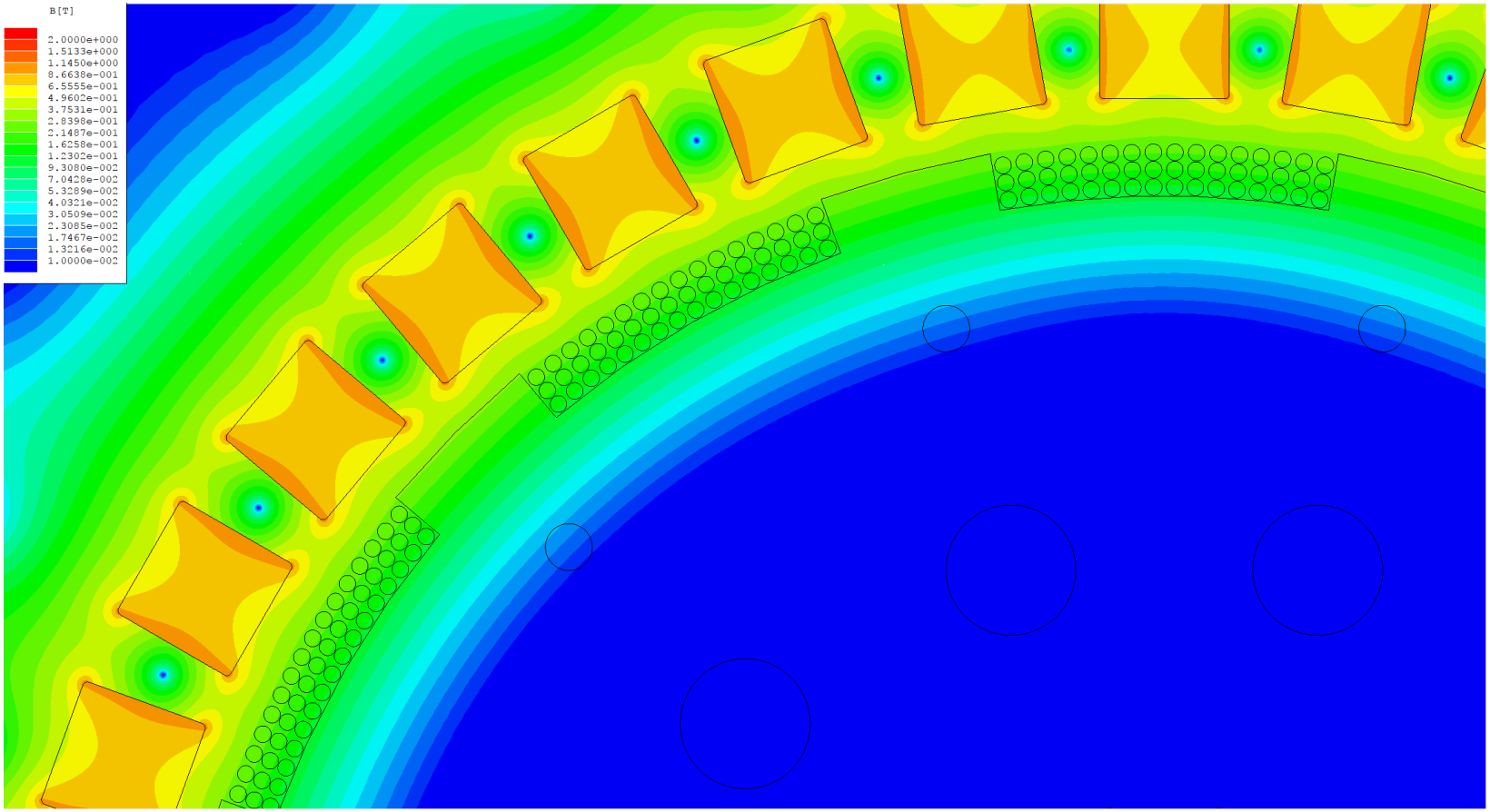


Figure 2.7: Close-up of circular magnet array with conventional north-south configuration.

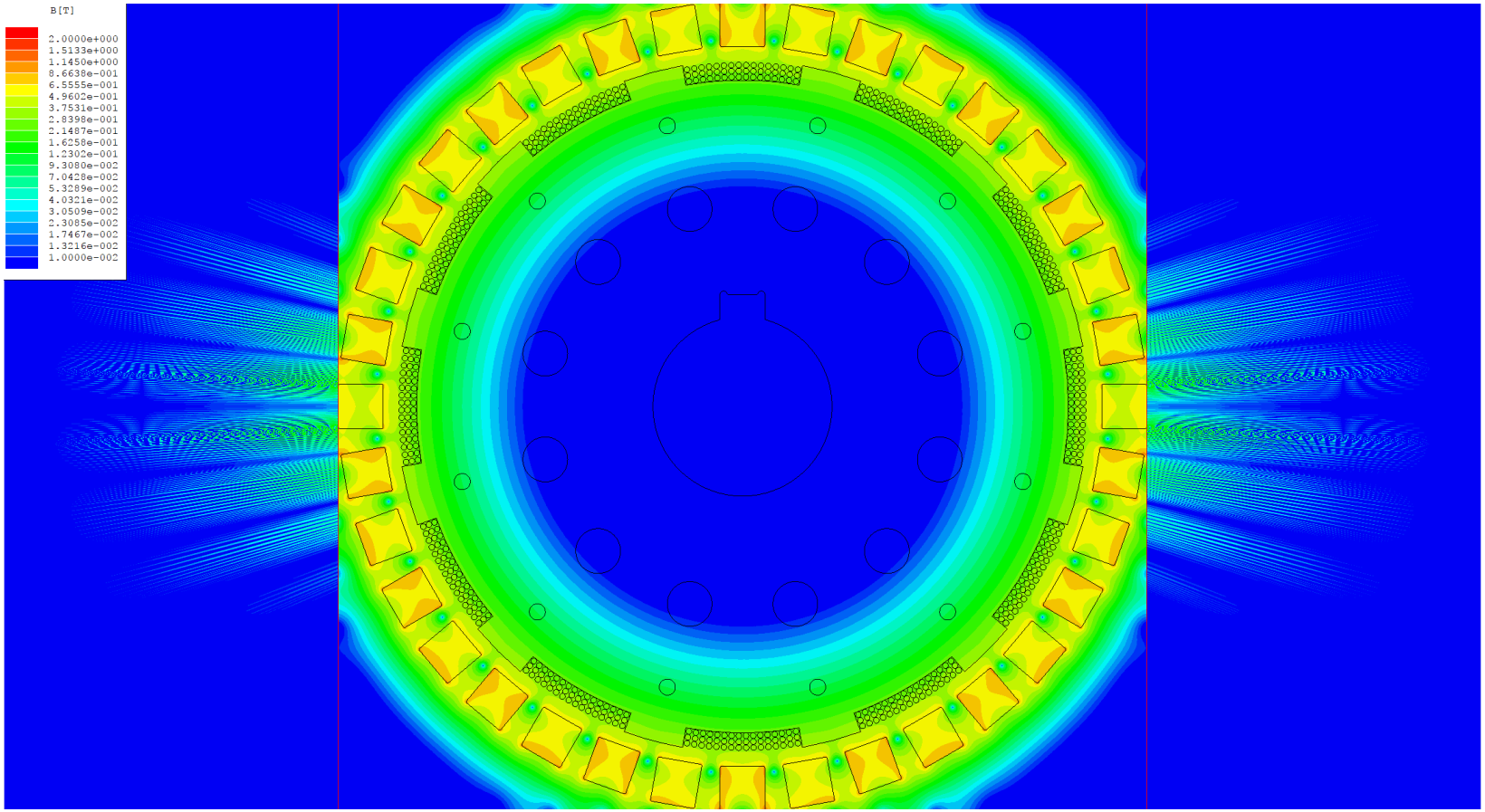


Figure 2.8: Circular Halbach magnet array.

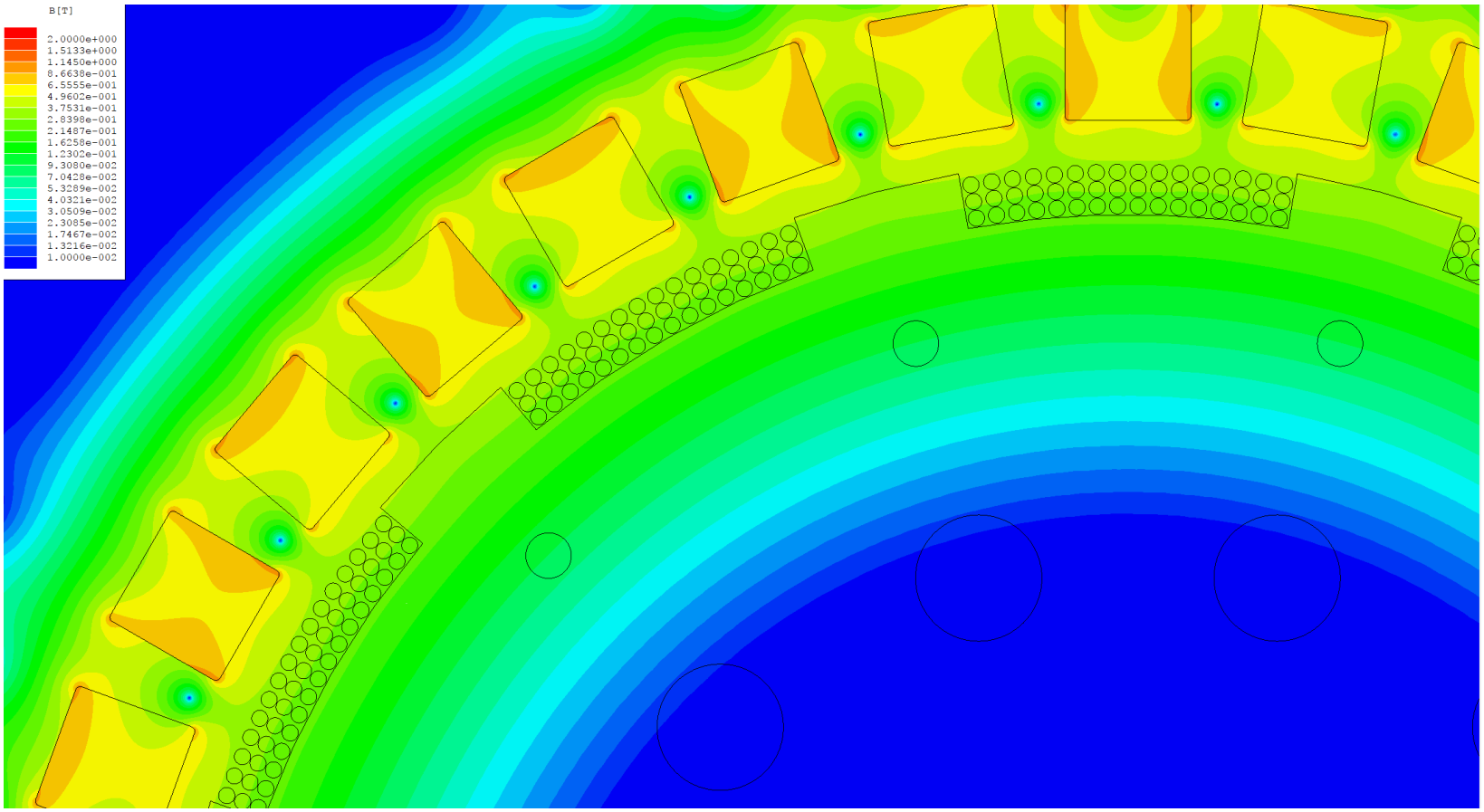


Figure 2.9: Close-up of circular Halbach magnet array.

fill in the slots on the motor's rotor and Figure 2.9 is the enhanced view of this setup. It is important to keep in mind for Figures 2.6 and 2.8 that the red lines mark the edge of the simulation and the results from outside of this area should not be considered accurate.

These simulation results confirm the expectations that a Halbach magnet array serves to concentrate the magnetic field on a particular side of its configuration. In this way, the strength of the magnetic field from the magnets can be amplified. This will serve to increase the torque generated by the motor when current is passed through the coils. Based on these results, the initial design of the motor was finalized.

2.4 Designed Torque Calculation

Referring to (1.1), the torque can be written as a function of the current and a torque constant. Given the specifications of N_s and N_w, an approximate radius of 0.1 m, an approximate wire length of 0.0254 m and a conservative magnetic field peak of 0.2 T on the stator's coils, the torque constant (from (1.2) can be found to be

$$k_T = 2 \cdot 24 \cdot 6 \cdot 0.0254 \cdot 0.05 \cdot 0.2 \,\mathrm{N \cdot m/A} \tag{2.21}$$

$$= 0.073152 \,\mathrm{N \cdot m/A} \tag{2.22}$$

2.5 Design Summary

Given the preceding design description, the motor has the added benefit of being able to have its rotor and stator be 3D printed (other than the coils and magnets). This allows cheap and fast production as well as ensures the magnetic fields generated internally should be minimally interfered with. By contrast, electrical steel has a relative permeability of about 5000 [26]. With the design of the motor completed, the construction can begin.

3.1 Motor Infrastructure

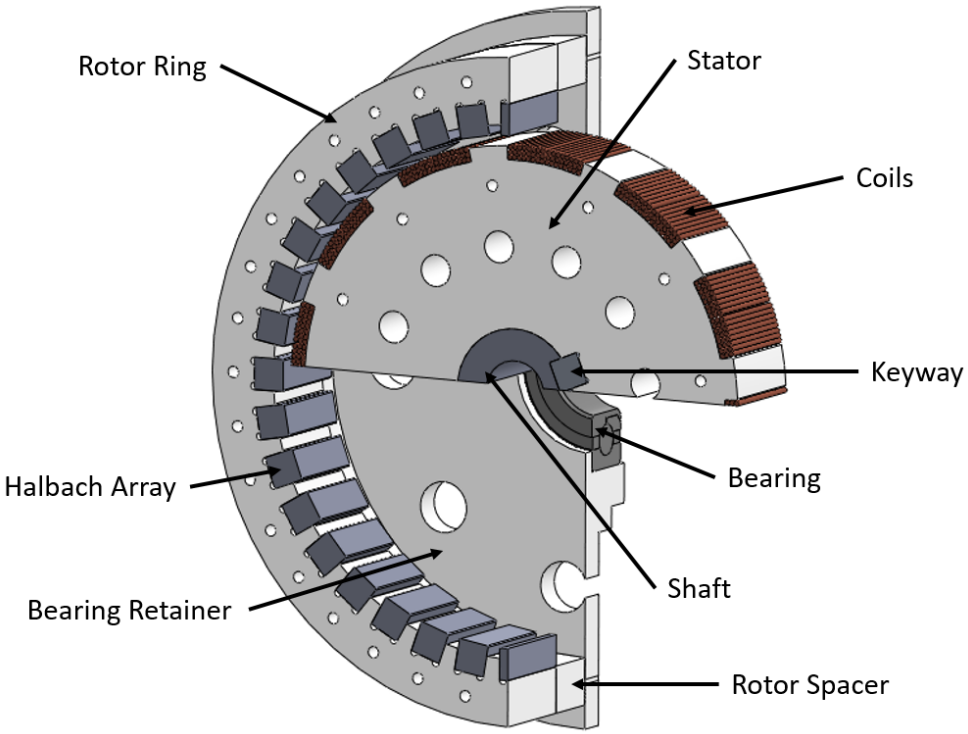


Figure 3.1: Double cutaway view of motor internal structure.

In an effort to accomplish the objectives set forth in this thesis, a motor has been built that incorporates a Halbach magnet array in a PMAC motor. This motor makes use of two phases and is an external-rotor motor. The motor contains nine sets of Halbach magnet arrays lining the rotor (making for a total of thirty-six magnets) and twelve coils between the two phases. The designed internals of the motor can be seen in Figure 3.1, while the fully assembled motor can be seen in Figure 3.2. Additionally, a high-level system

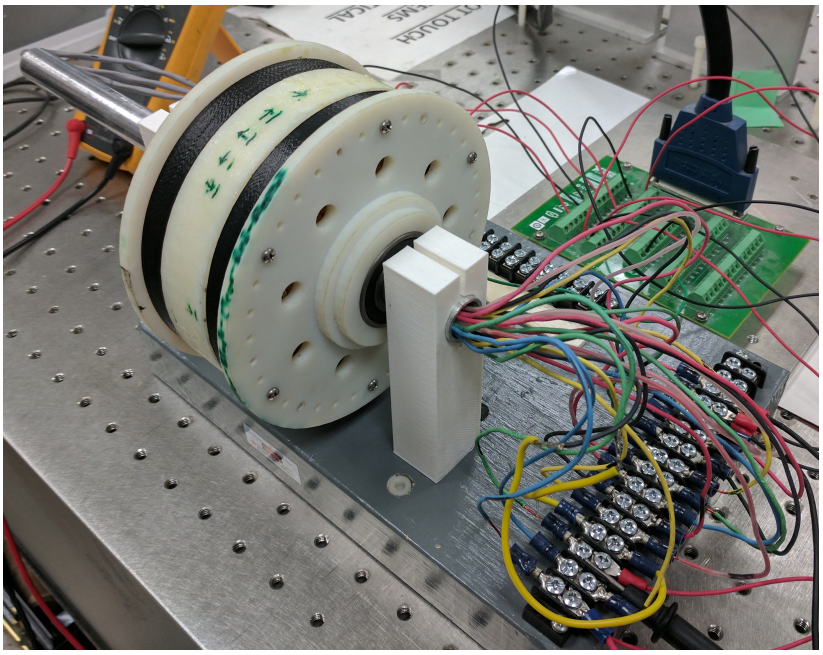


Figure 3.2: Assembled Halbach array motor.

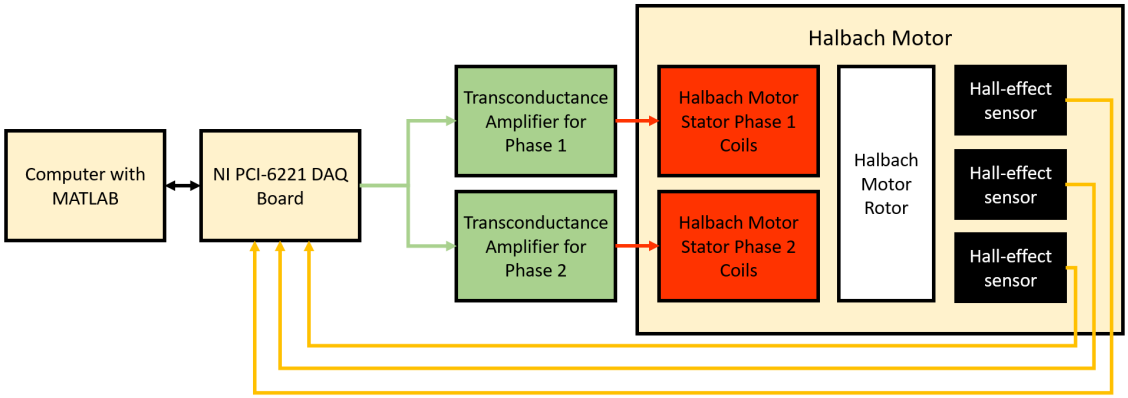


Figure 3.3: High-level system block diagram.

architecture of the motor is given in Figure 3.3.

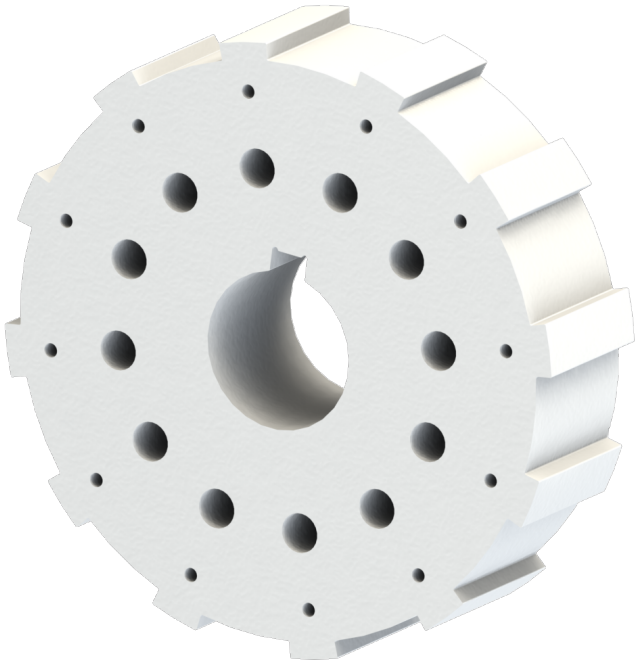


Figure 3.4: Stator of motor.

3.1.1 Stator

The stator can be seen in Figure 3.4. It consists of a cog-like structure made of 3D printed PLA with an outer diameter of 9.8044 cm (3.86 in) and an inner diameter of 9.2456 cm (3.64 in) for the twelve spokes evenly-spaced around the structure.

These spokes were then wrapped with 20 AWG NEMA MW136-c wire such that twelve distinct coils were placed on the radial surface of the stator. Each of these coils consisted of three layers of wire with each layer being eight layers of wire wide. This resulted in each of the twelve coils consisting of twenty-four wraps of the wire. A representation of the coil wrappings can be seen in Figure 3.5 where the small gap in the middle of the sets of wires is the location where two adjacent coils abut. The final implementation of the stator is given in Figure 3.6. The ends of the coils run through a hole in the shaft where they are connected to a block of screw terminals in such a way that every other coil

Figure 3.5: Design of stator of motor including the coils.

is connected together in series. This produces two sets of six coils which give the motor its two-phase property. Each of these phases' properties were measured using an LCR meter and the results are recorded in Table 3.1. Based on these properties, the impedance of each of the phases was calculated with

$$Z = \sqrt{(\omega_e L)^2 + R^2} \tag{3.1}$$

$$= \sqrt{(\frac{\omega_m}{60} 9L)^2 + R^2} \tag{3.2}$$

for various speeds of the motor. The results of this procedure are displayed in Figure 3.7. The ends of each of the phases are the inputs to the motor and are connected to the transconductance amplifiers.

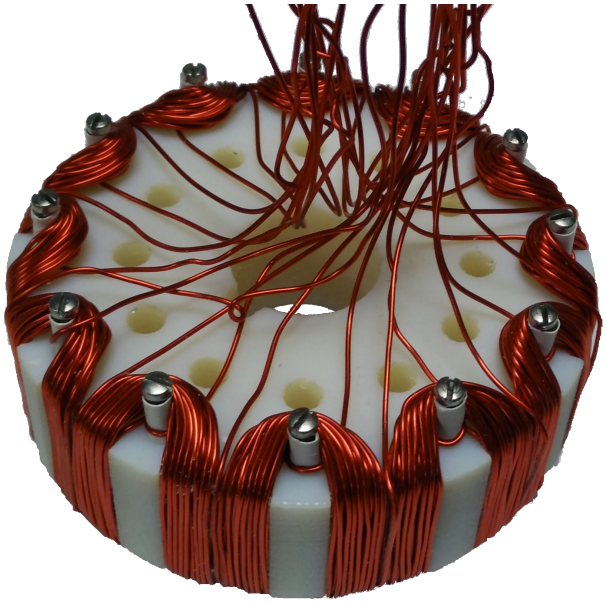


Figure 3.6: Implemented stator of motor.

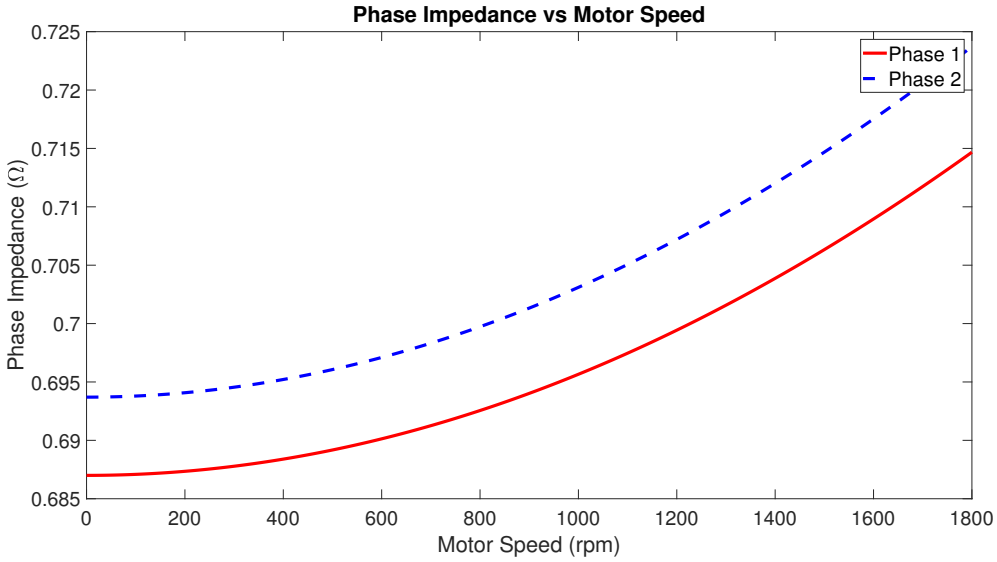


Figure 3.7: Phase impedance vs motor speed.

Table 3.1: Electrical properties of motor's phases.

Phase	Resistance (Ω)	Inductance (μH)
One	0.6870	116.1
Two	0.6937	121.6

3.1.2 Rotor

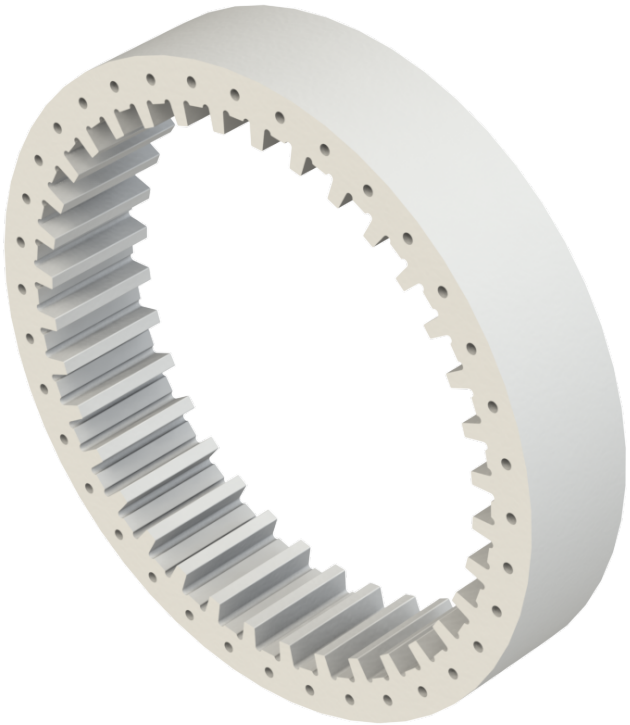


Figure 3.8: Design of rotor frame of motor.

The rotor frame can be seen in Figure 3.8. It consists of a ring of 3D printed plastic with 36 notches meant to house the magnets that were epoxyed in place. These magnets measured 0.638 cm (0.251 in) by 0.638 cm by 2.54 cm (1 in). The ring has an outer

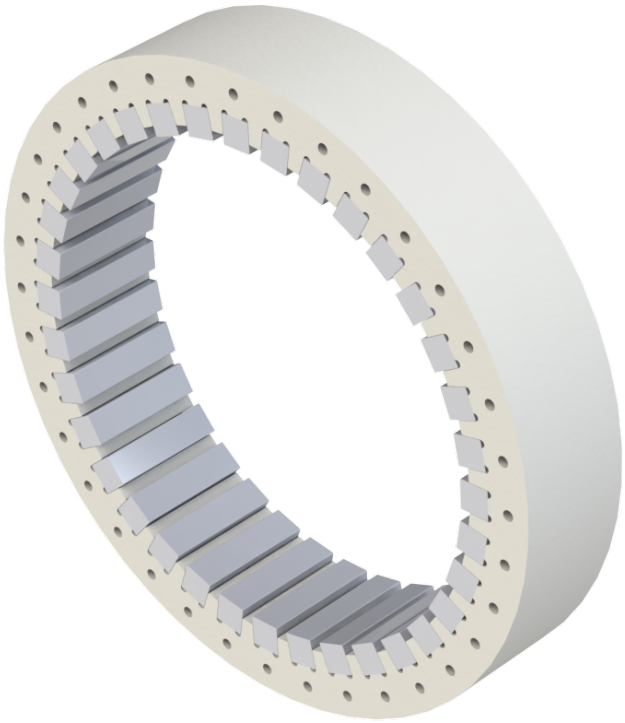


Figure 3.9: Rotor frame with the magnets inserted.

diameter of 13.3 cm (5.24 in) and an approximate inner diameter of 10.2 cm (4.01 in) including the magnets. That is the measurement from the center of the rotor to the middle of the nearest face of any one of the magnets. The rotor with the magnets in place can be seen in Figure 3.9. The magnets were attached to the rotor frame by using an epoxy to glue a few of the magnets in place at a time. While drying, the magnets were clamped in place to prevent any misalignments from the magnets interactions with one another. As the magnets were inset, a Halbach array was created by alternating the direction the north side of the magnet faced. The pattern used to create the array is visualized in Figure 3.10 and the final result can be seen in Figure 3.11.

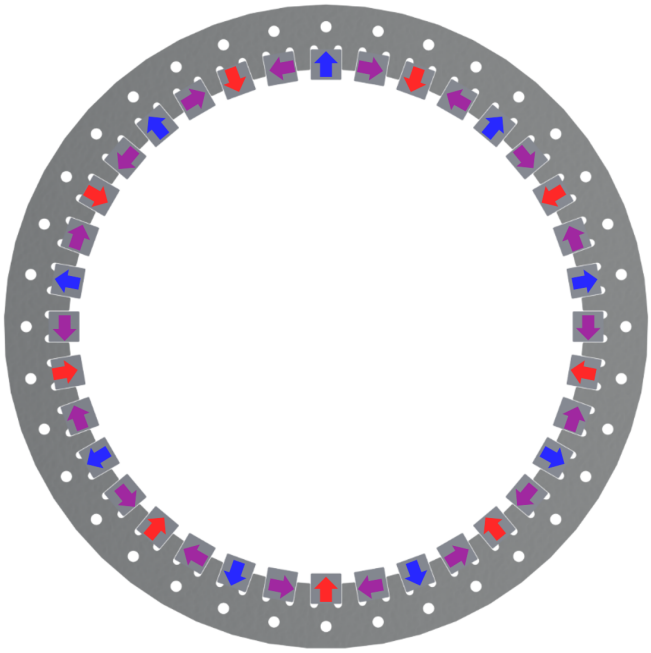


Figure 3.10: Rotor of motor with the magnets' magnetization.

Figure 3.11: Assembled rotor of motor.

3.1.3 Motor Base and Spacer

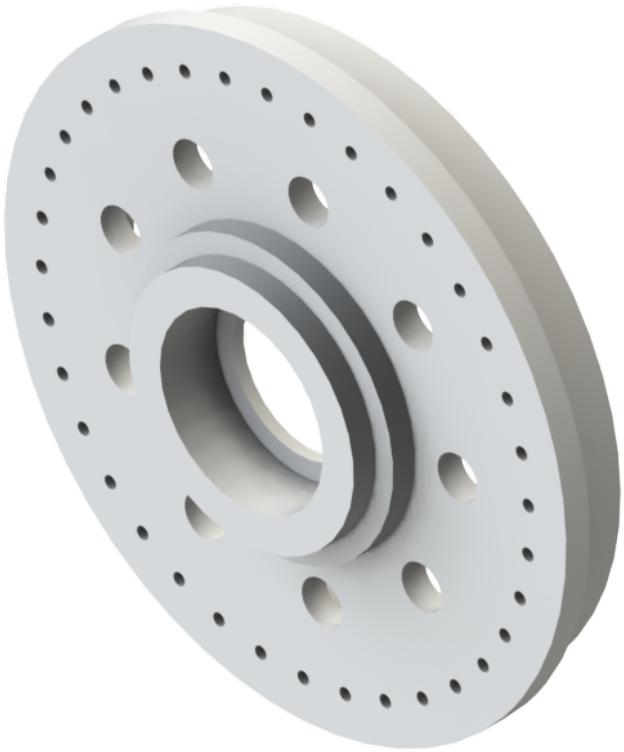


Figure 3.12: Motor spacer.

The 3D printed spacer used to connect the rotor to the shaft can be seen in Figure 3.12. This part was affixed to both sides of the rotor via small inset screws and connected to the shaft via bearings.

The motor base's mechanical components can be seen in Figure 3.13. The base was set atop an optical table and fixed into place. The bearings were affixed to the shaft and inset inside the spacers. The shaft was used to ferry the wires from the stator's coils to the supporting electrical hardware.

The final design can be see in Figure 3.14 and the actual final product can be seen in

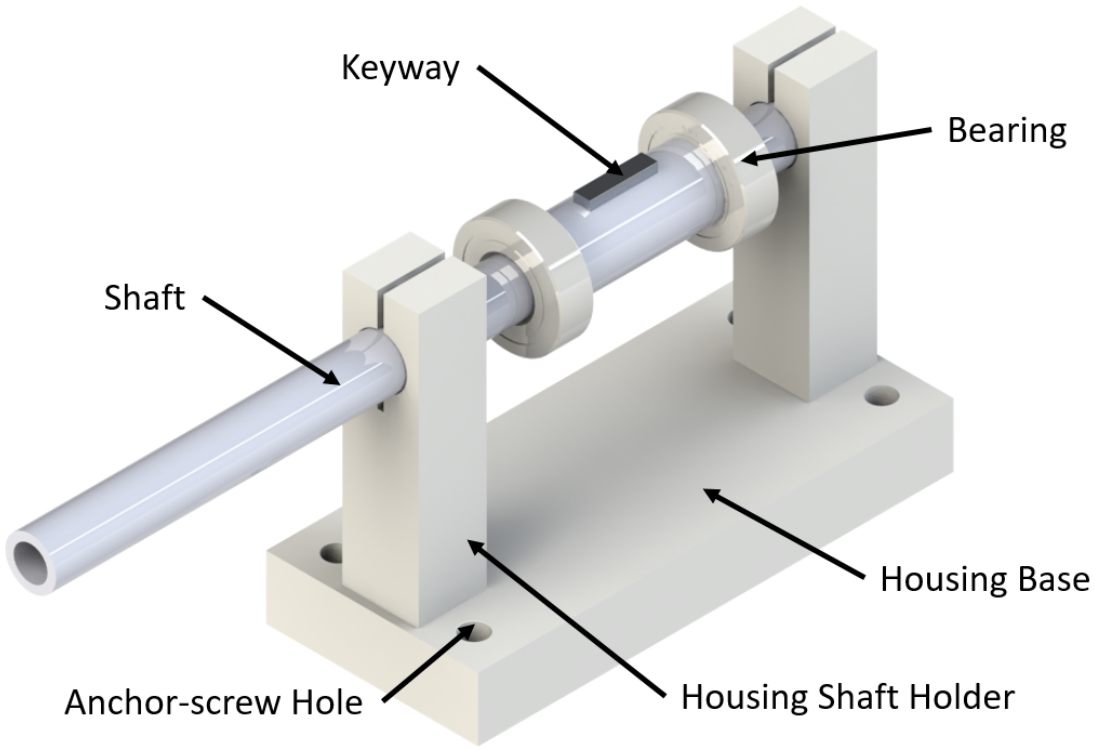


Figure 3.13: Mechanical support components of motor.

Figure 3.2. According to the model, the final weight of all of the rotor's components is 0.482 kg (1.063 lb) and the moment of inertia about the axis of interest is 0.00142 kg·m^2 (4.853 lb·in^2).

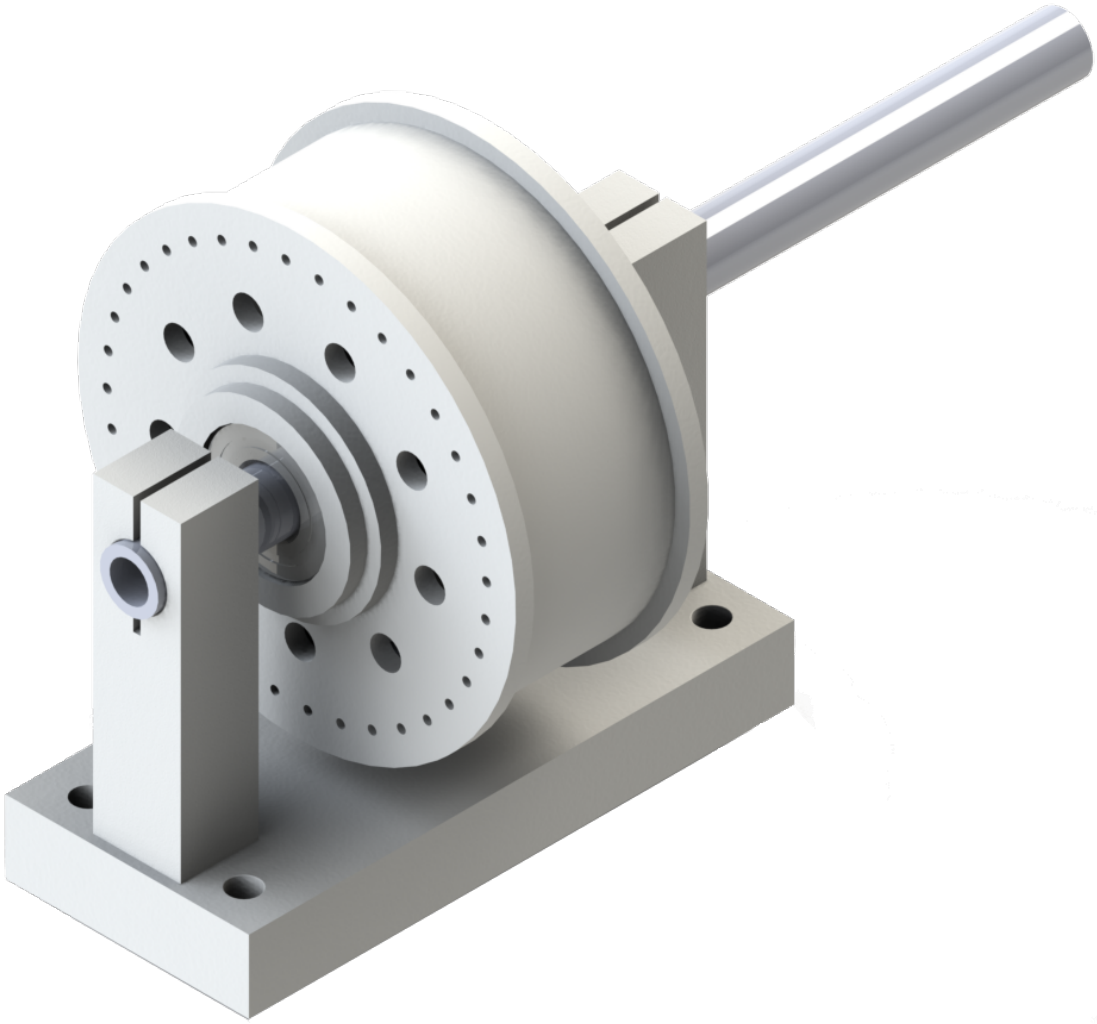


Figure 3.14: Motor assembly final design.

3.2 Motor Support Infrastructure

In order to test the properties of the motor, each phase is driven by a voltage-to-current transconductance amplifier where the power is supplied from three 3-A power supplies wired in parallel, and the control is handled by a NI PCI-6221 board integrated into a computer. This board simultaneously handles both the output voltage to the amplifiers and data input. The data input acquired is all analog. This board reads the voltage from the two

electrical poles (positive and negative) for each of the motor's phases as well as from three Hall-effect sensors embedded in the motor. The output voltage from the NI board is controlled by one of several programs coded in MATLAB that allow for different objectives to be accomplished with regards to testing the motor. These MATLAB programs make use of MathWork's Data Acquisition Toolbox to allow interfacing between the MATLAB code and NI-DAQ hardware.

3.2.1 NI PCI-6221 DAQ Board

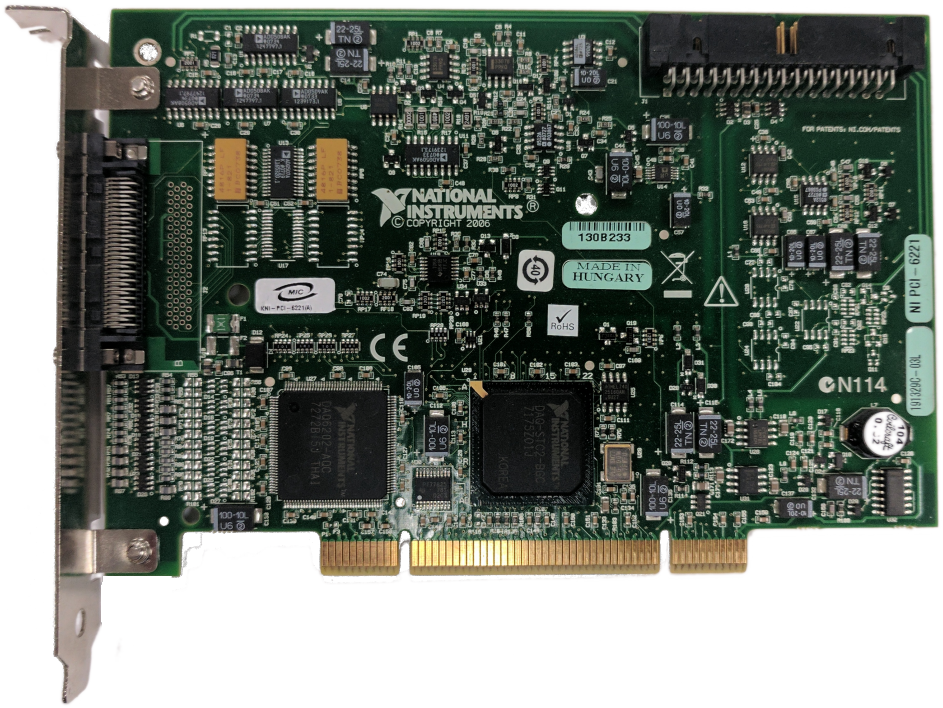


Figure 3.15: NI PCI-6221

The National Instruments[1] PCI-6221 is the connection between a computer and the experimental setup. In terms of output, the board is able to supply the necessary sinusoidal

[1]National Instruments, 11500 N Mopac Expwy Austin, TX U.S.A.

waves as input to the transconductance amplifiers as well as provide the 5-V supply voltage to the Hall-effect sensors. In terms of input, the board acquires the voltage at both ends of the motor's phases so that the differential voltage across each set of coils can be calculated. Additionally, the board reads the voltage from the Hall-effect sensors. This board was integrated into MATLAB via Mathwork's Data Acquisition Toolbox. The code that was used along with the toolbox can be found in Appendix A, and the board itself can be seen in Figure 3.15.

3.2.2 Transconductance Amplifier

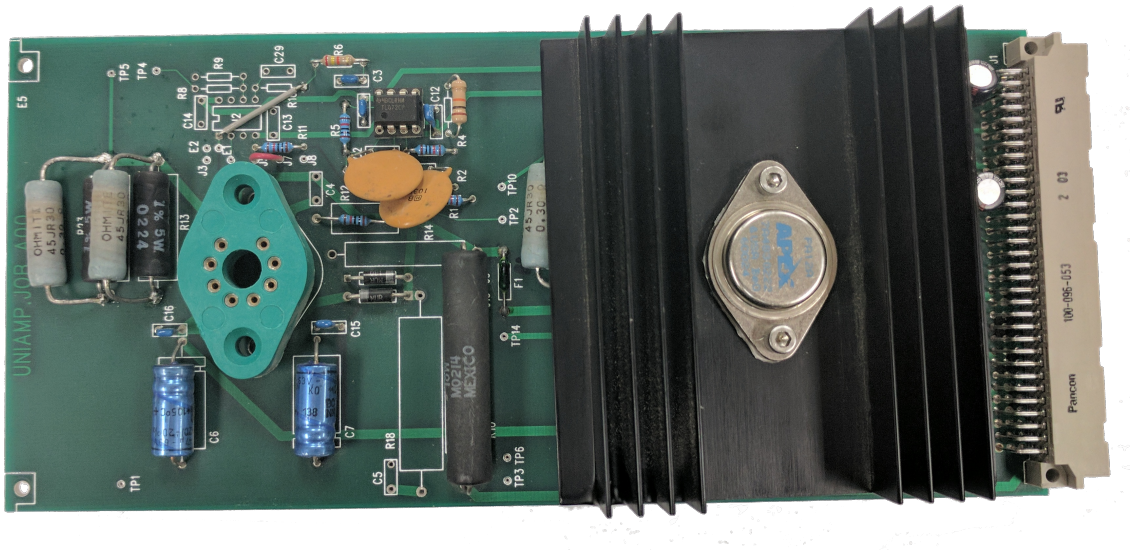


Figure 3.16: Transconductance amplifier with one amplifier and heat-sink removed.

The transconductance amplifier comes on a board with a pair of them as can be seen in Figure 3.16. This amplifier acts as a unity gain voltage-to-current buffer amplifier. That is, no matter the load, if one volt is supplied at input, the amplifier will attempt to output one amp. This amplifier was modified from the work of Kim [27]. Based on the Power

Operational Amplifier PA12A manufactured by Apex Microtechnology[2]. The technical specifications for the PA12A are listed in Table 3.2.

Table 3.2: Technical Specifications of PA12A.

Maximum Supply Voltage	100 V
Maximum Output Current	15 A
Maximum Power Dissipation	125 W
Input Impedance	200 MΩ
Slew Rate	4 V/μs
Settling Time to 0.1% of 2 V Step Input	2 μs

Figure 3.17 gives the circuit for one transconductance amplifier with *LA* and *R*11 representing the load's inductance and resistance. In this case, the *LA* and *R*11 values were obtained by measuring the values for each phase of the motor. These properties are listed in Table 3.1. In the transconductance amplifier, it is easy to see that all of the capacitors are 10 pF (except for two 47 µF capacitors attached to the power supply of the PA12A) and all of the resistors are 10 kΩ except for *R*6 and *R*10 (which are 24 kΩ and 1 Ω respectively). Given these values, the transfer function of the circuit can be calculated to be

$$G = 4 \cdot 10^8 \frac{s^2 + 14167s + 4.1667 \cdot 10^7}{s^4 + 74079s^3 + 1.3482 \cdot 10^9 s^2 + 8.7409 \cdot 10^{12} s + 1.6665 \cdot 10^{16}} \tag{3.3}$$

This yields the Bode plot given in Figure 3.18. As this diagram confirms, the gain of the transconductance amplifier is approximately 1 until 1 kHz where it has dropped to a gain of $1/\sqrt{2}$. In other words, the bandwidth of this transconductance amplifier is 1 kHz.

[2]Apex Microtechnology, 5980 N. Shannon Road Tucson, AZ 85741, U.S.A

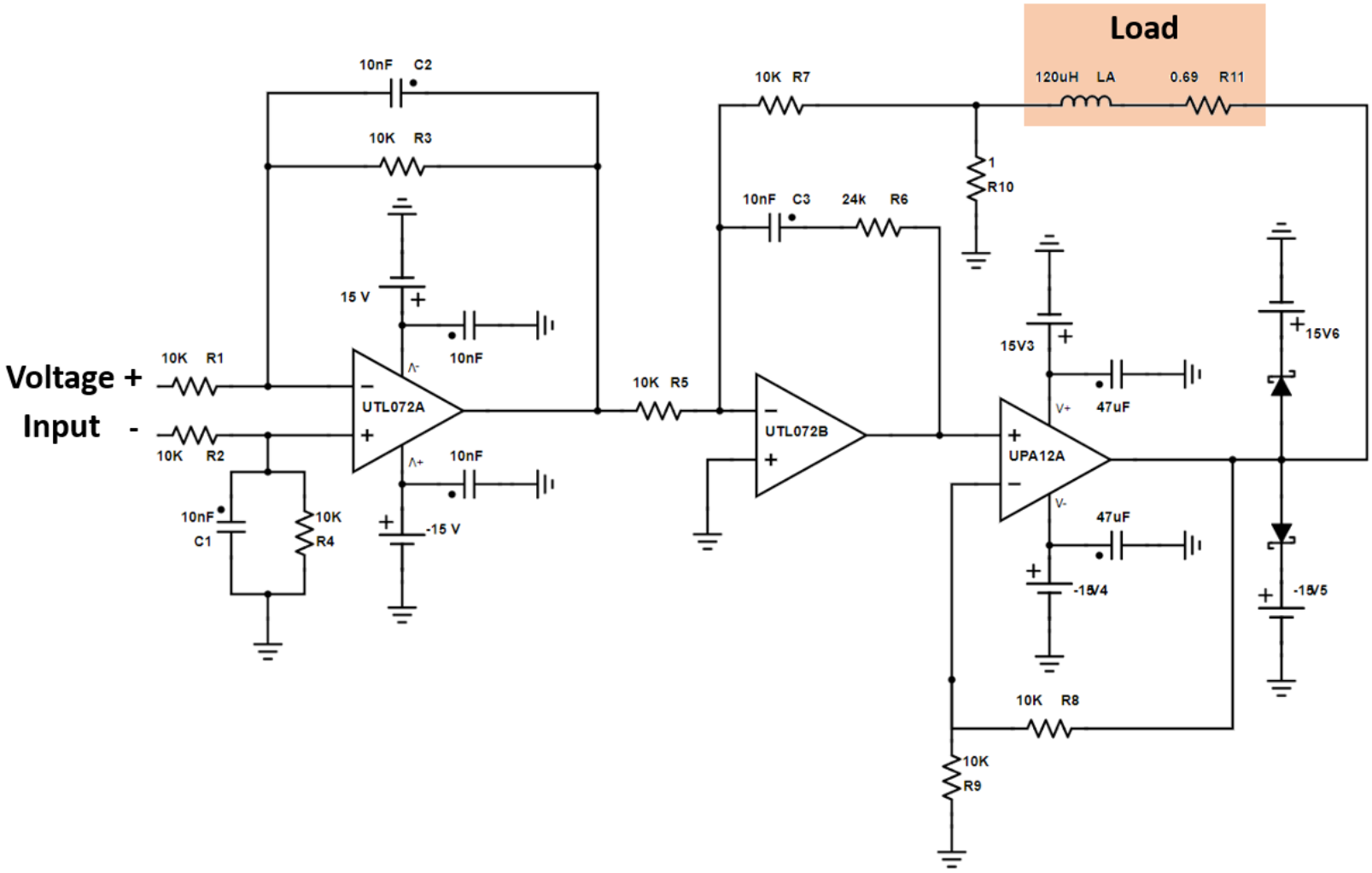


Figure 3.17: Circuit diagram of transconductance amplifier.

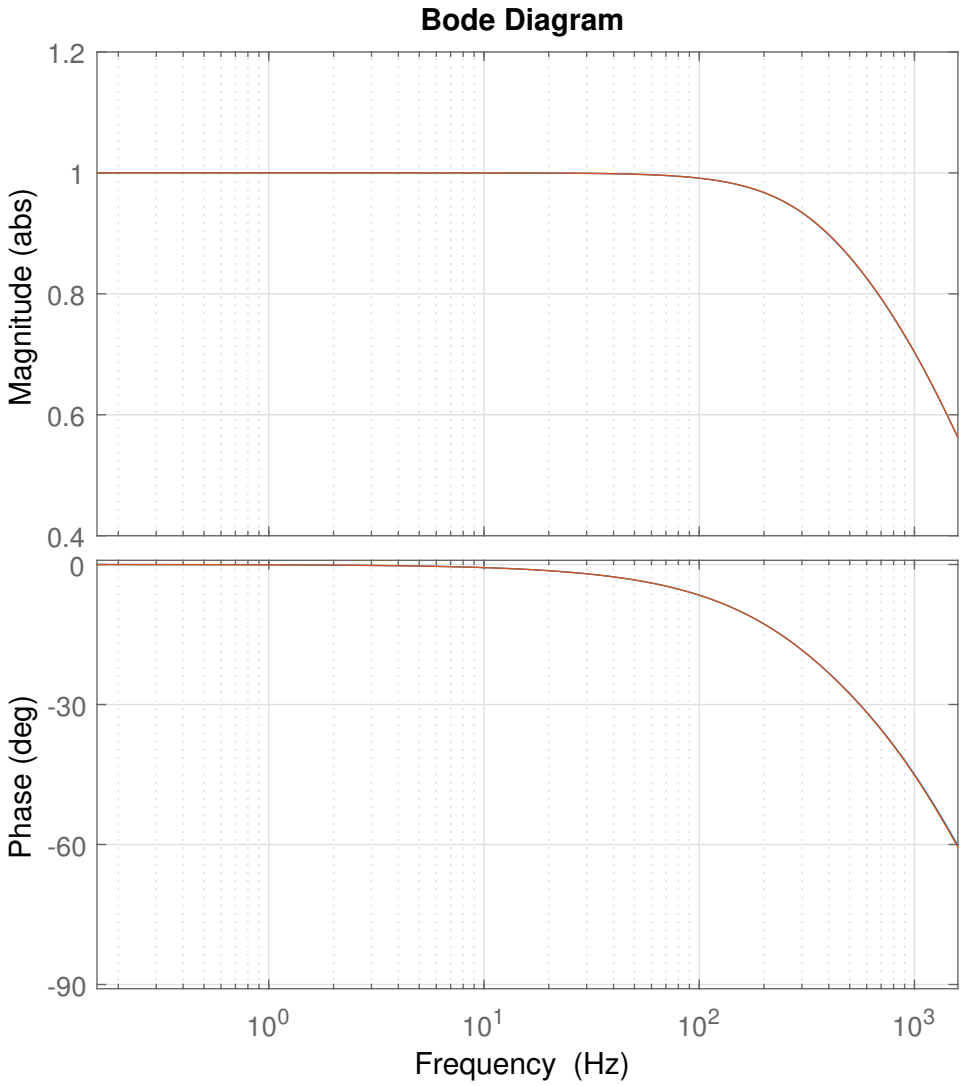


Figure 3.18: Bode plot of transconductance amplifier.

3.2.3 Hall-Effect Sensor

The Hall-effect sensors used for this set of experiments is the A1302K ratiometric linear Hall-effect sensor from Allegero MicroSystems[3]. The technical specifications are listed in Table 3.3.

Table 3.3: Technical Specifications of A1302k.

Maximum Supply Voltage	8 V
No Magnetic Field Output (B = 0 T)	0.5 · Supply Voltage
Output Sensitivity	1.3 mV/G
Output Bandwidth	20 kHz
Linearity	$\pm$2.5%

The Hall-effect sensors were labeled based on the stator cog's they are situated on. That is, 7, 12, and 2. Hall-effect sensor 7 was offset from Hall-effect sensor 12 by five spokes or 150° and Hall-effect sensor 12 was offset from Hall-effect sensor 2 by two spokes or 60° in the same rotational direction. This can be seen in Figure 3.19. These sensors were spaced around the stator such that the measurement plane was oriented radially with the motor in an attempt to capture the magnetic field from the rotor. Specifically, as the stator contained twelve spokes, the Hall effect sensors were placed on the sides of these spokes near the inside of the coil windings. That is, the top of all of the Hall-effect sensors is in the Z-direction while the front of Hall-effect sensors 7 and 12 is faced radially outwards. Hall-effect sensor 2's front face is situated radially inwards.

[3]Allegro MicroSystems, LLC, 115 Northeast Cutoff, Worcester, MA 01606, U.S.A

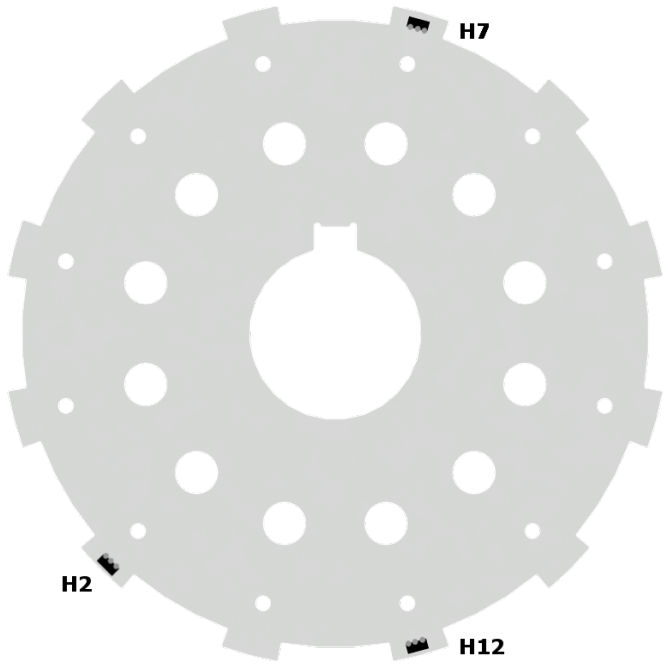


Figure 3.19: Location of Hall-effect sensors.

3.2.4 Power Supply

In order to supply the power necessary to run the motor, three TDK-Lambda[4] power supplies (seen in Figure 3.20) were connected in parallel to the supply terminals of the transconductance amplifier. Each of the power supplies was rated at 3 A and 15 V leading to a maximum supply current of 9 A at 15 V. Given that this is a two-phase system, this yields a maximum per-phase current of $\frac{9\,\mathrm{A}}{\sqrt{2}} = 6.364$ A. Given the measured resistance of the coils listed in Table 3.1, the maximum per-phase power that can be delivered through the power supply becomes

$$P = RI^2 = R(\frac{I_{max}}{\sqrt{2}})^2 = 0.68\frac{9^2}{2} = 27.54\,\mathrm{W} \tag{3.4}$$

[4]TDK-Lambda 3-9-1, Shibaura, Minato-ku, Tokyo 108-0023, JAPAN

Figure 3.20: One of the three power supplies from TDK-Lambda used to power the motor.

4.1 Motor Friction Force

In order to determine the friction force inherent in the motor system, experimental dynamics data was captured on video. The motor was initially accelerated to 10 Hz before input to the motor was removed. Then, the angular position of the motor was recorded over time. This angular positioning was done via visual estimation from video shot at 60 Hz. The experimental data and a curve fit to the data can be seen in Figure 4.1. The R^2 value of this fit is 0.9961.

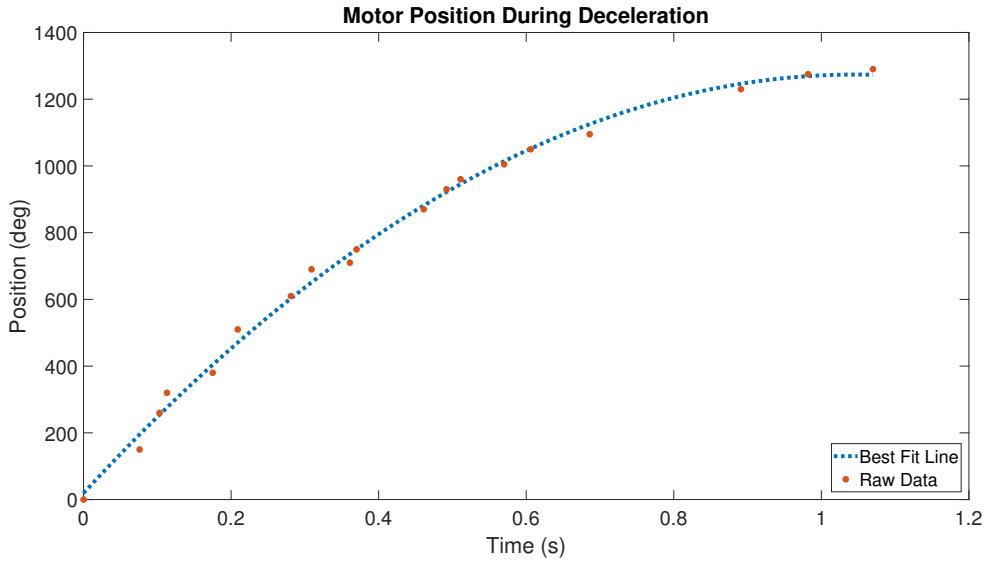


Figure 4.1: Deceleration curve from visual estimation.

This data was then also confirmed by tracking the position of motor utilizing the Hall-effect sensors. By recognizing that the Hall-effect sensors will cross their zero-field value eighteen times during the course of every rotation, the position of the motor can be known

to have advanced 20° every time a single Hall-effect sensor crosses this value. By plotting this positional progression over time, a more complete description of the motor's movement can be developed. This data and the corresponding quadratic curve fit (with an R^2 value of 0.9989) can be seen in Figure 4.2. It is worth noting that the two methods yielded friction forces that were within 5% of each other.

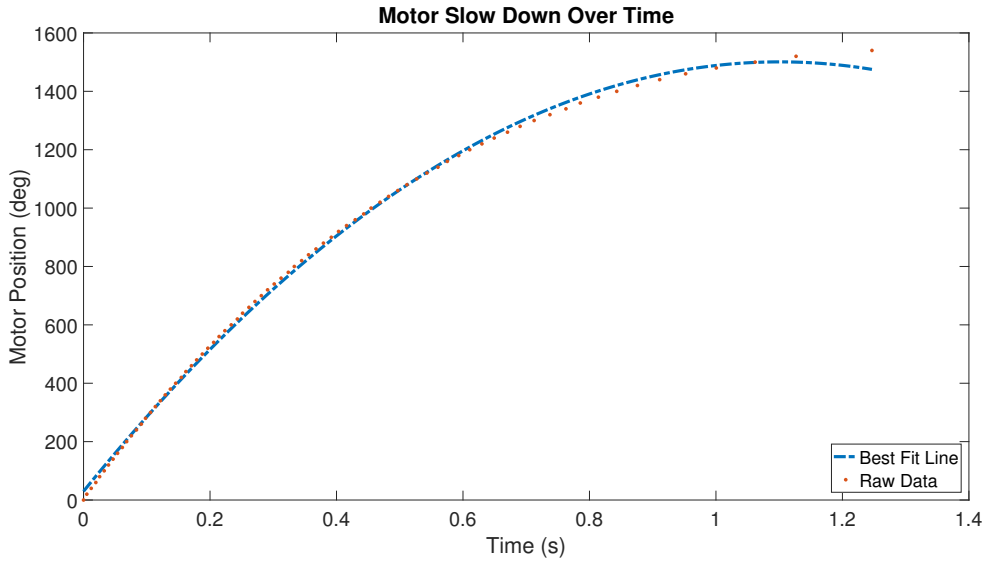


Figure 4.2: Deceleration curve from Hall-effect sensor voltage.

Given that over the time interval of deceleration the position (in degrees) is given by

$$\theta(t) = -1214t^2 + 2672t + 29.9 \tag{4.1}$$

the rotational speed and acceleration can be given respectively by

$$\omega(t) = -2428t + 2672 \tag{4.2}$$

$$\dot{\omega}(t) = -2428 \tag{4.3}$$

This equation can then converted from degrees to radians and the motor dynamics can be modeled as

$$I\dot{\omega}(t) = T_{em} - T_f(t) \tag{4.4}$$

where I is the mass moment of inertia about the axis of rotation and T_f is the torque from friction. Given the deceleration model from Figure 4.1, T_{em} can be set to zero for all *t* and $\dot{\omega}(t)$ is known. I was calculated from the system model to be 0.001414 kg·m^2. Plugging these values in yields an average friction torque (during the course of each motor rotation) of

$$0.001414(-13.489\pi) = -T_f \tag{4.5}$$

$$T_f = 0.0599\ \mathrm{N}\cdot\mathrm{m} \tag{4.6}$$

4.2 Torque Characteristics

As was established previously, the torque should only be dependent on current and will be considered independent of rotor position. Given this, in the absence of an external rotor dynamometer, the stall torque of the motor can be measured to instead determine the motor's torque vs current characteristics. To achieve this, a weight was first attached to the rotor via fishing line and an initially high current was passed through the coils of one phase. This current was then progressively lowered until the torque produced from the motor was no longer large enough to prevent the weight from being pulled downward and causing the rotor to spin. This experiment was repeated for a number of different weights for each of the motor's phases. The basic setup can be seen in Figure 4.3. It should be noted that to prevent fluctuations in friction from differing rotor positions, the rotor was started in the same place for every run. The results of this experiment can be seen in

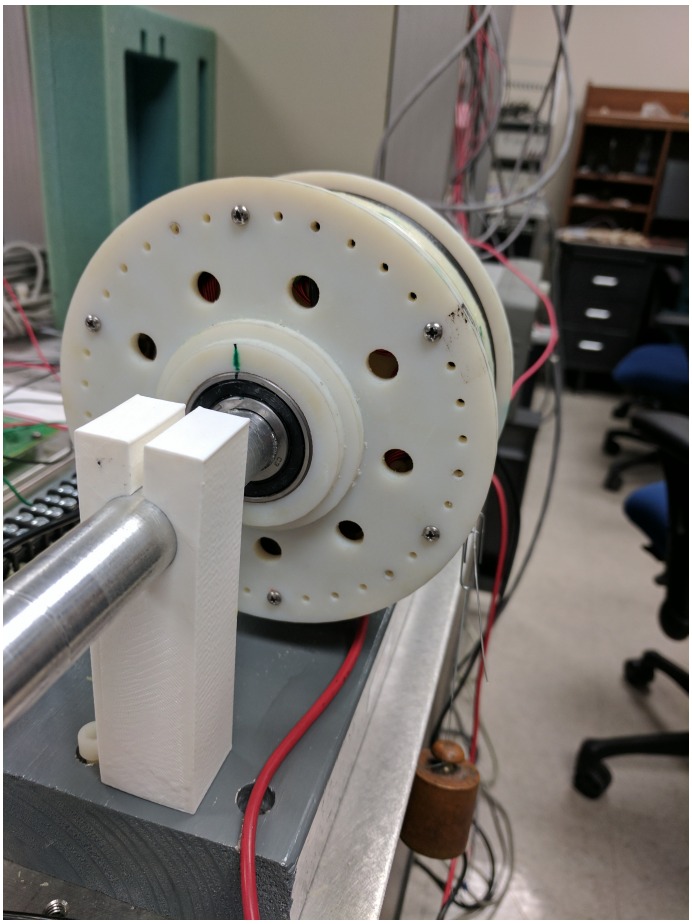


Figure 4.3: Stall torque experimentation setup.

Figure 4.4 and the properties of the linear fit line can be found in Figure 4.1. In this case, the y-intercept represents the friction force at the stable point of each phase (the intercepts are slightly different between the phases due to the positional offset of the phases), and the slope represents the generated torque per ampere for each phase. Though both phases were wound with the same number of turns, the torque disparity between the phases means that in order to achieve a torque that is independent of rotor position, the voltage supplied to the first phase must be regulated down so the phase currents match. That is, the voltage supplied to phase 1 must be $\frac{0.0855}{0.0897}100 = 95.3\%$ of the voltage supplied to phase 1. This

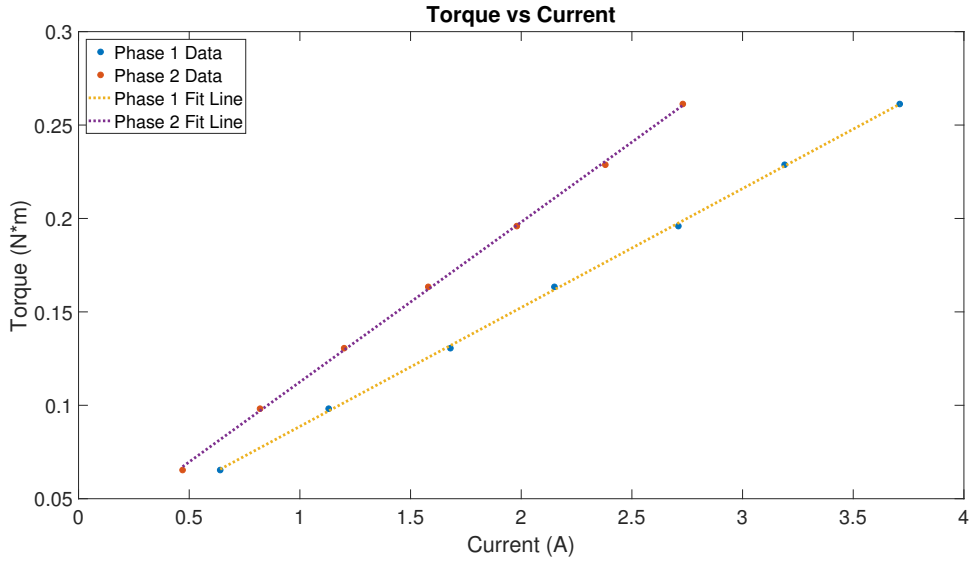


Figure 4.4: Torque vs current.

is most likely It is worth noting that the measured torques per current are quite close to the 0.0732 $\mathrm{N} \cdot \mathrm{m}/\mathrm{A}$ calculated in (2.22). Given that the calculated torque was based on a magnetic field that was 20% weaker than the expected magnetic field as stated in Section 2.3, it is more appropriate to compare the measured torque values to $1.2(0.0732) = 0.0878$, which yields a result even closer to the prior calculations.

Table 4.1: Properties of torque vs current fit lines.

	Slope	Y**-Intercept**	**R Squared**
Phase 1	0.08978	0.2099	0.9997
Phase 2	0.08557	0.02692	0.9996

4.3 Motor Current/Hall-Effect Sensor Interaction

Before the data collected from the Hall-effect sensors can be used to determine the position of the motor, the interaction between the phase currents and the Hall-effect sensors

must be measured and accounted for. As the Hall-effect sensors are located on the stator, the positioning of the rotor does not affect the interaction of the motor current and sensors. In order to determine this interaction, a large current was first passed through phase 1 to align the motor in an arbitrary orientation. This current was then dropped to zero and a base voltage value for the Hall-effect sensors is established. This initial current spike is done to ensure the motor does not turn further once current is applied. The base voltage value represents the voltage response of the Hall-effect sensors in the given stable position of the rotor. After this base voltage is established, the current being passed through phase 1 is increased from 0 to 3 A and then back down to 0. This procedure is repeated for negative currents including the initial current spike. Though ideally the motor should not turn with a negative current (as the motor's rotor should be situated in a position where the forces from every pair of phase 1 coils cancel out regardless of current direction), this was done anyway to ensure thoroughness. As expected, the motor did in fact not change position during the negative current spike. This was reflected in the fact that the base voltage Hall-effect values obtained after the positive and negative current spikes differed by less than 0.02 % for Hall-effect sensors 7 and 2 for example. This entire procedure was then repeated for phase 2 as well. The results from both sets of procedures can be seen in Figures 4.5 and 4.6.

Given the results of the above responses, a relationship was developed between each of the phases and each of the Hall-effect sensors. This was done by first calculating the base voltage as the average of the responses during the period after the initial current spike. The change in voltage from this base voltage value during the current changing procedures was then plotted against the current and a trend was developed. The Hall-effect sensor response and corresponding trend lines can be seen in Figures 4.7 and 4.8. The trend line information is given in Tables 4.2, 4.3, and 4.4. It is readily apparent in these figures that phase 1 affects Hall-effect sensors 12 and 2 quite readily but does not affect Hall-effect

sensor 7. Contrarily, phase 2 only affects Hall-effect sensor 7. This makes sense based on the positioning of the Hall-effect sensors where Hall-effect sensors 12 and 2 are located inside of phase 1 coils while Hall-effect sensor 7 is located inside of one of phase 2's coils. With these results, all future data will be adjusted accordingly. That is, Hall-effect voltages from sensors 2 and 12 will be adjusted based on the current in phase 1 while sensor 7 will be adjusted based on the current in phase 2.

Table 4.2: Hall-effect sensor 7 response to phase currents trend line information.

	Hall Effect Sensor 7		
	Slope	Y-Intercept	R-Squared
Phase 1	8.6057e-4	-1.0890e-4	0.1910
Phase 2	-0.01067	-3.0381-4	0.9885

Table 4.3: Hall-effect sensor 12 response to phase currents trend line information.

	Hall Effect Sensor 12		
	Slope	Y-Intercept	R-Squared
Phase 1	0.01857	0.00160	0.9676
Phase 2	6.2694e-4	-8.8320e-4	0.0270

Table 4.4: Hall-effect sensor 2 response to phase currents trend line information.

	Hall Effect Sensor 2		
	Slope	Y-Intercept	R-Squared
Phase 1	0.01738	-2.0957-5	0.9889
Phase 2	2.4169e-5	-2.7880e-4	0.0494

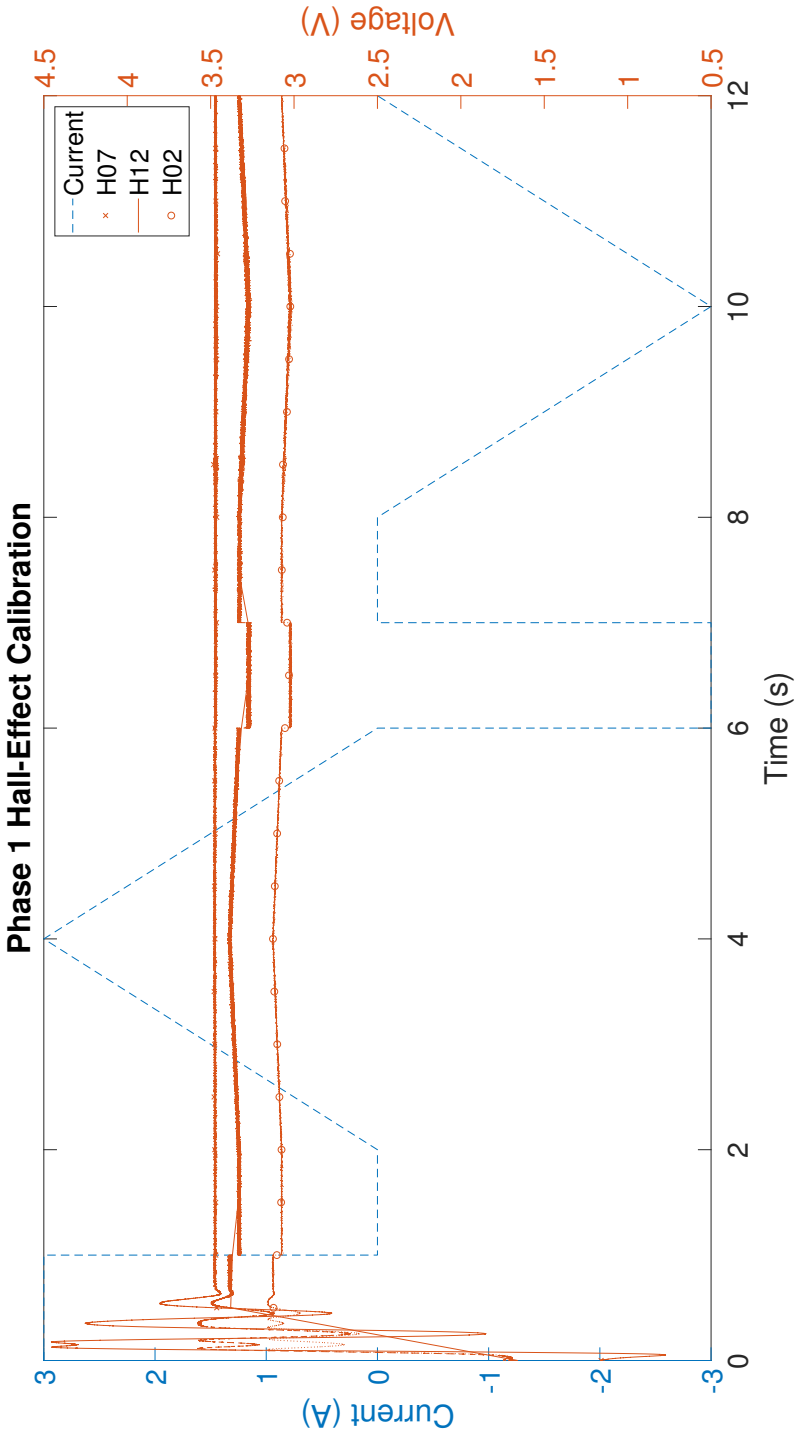


Figure 4.5: Hall-effect sensor data during phase 1 testing

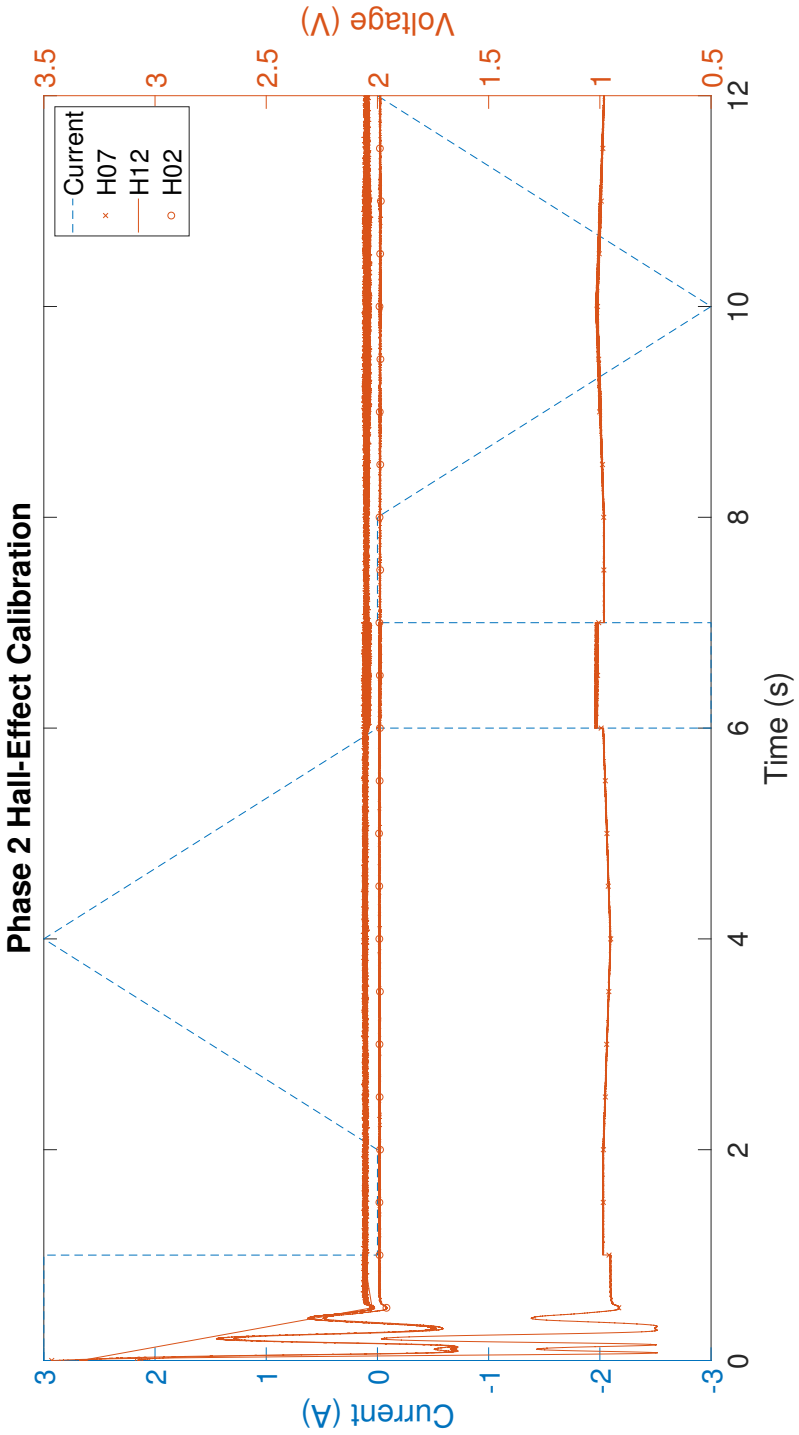


Figure 4.6: Hall-effect sensor data during phase 2 testing

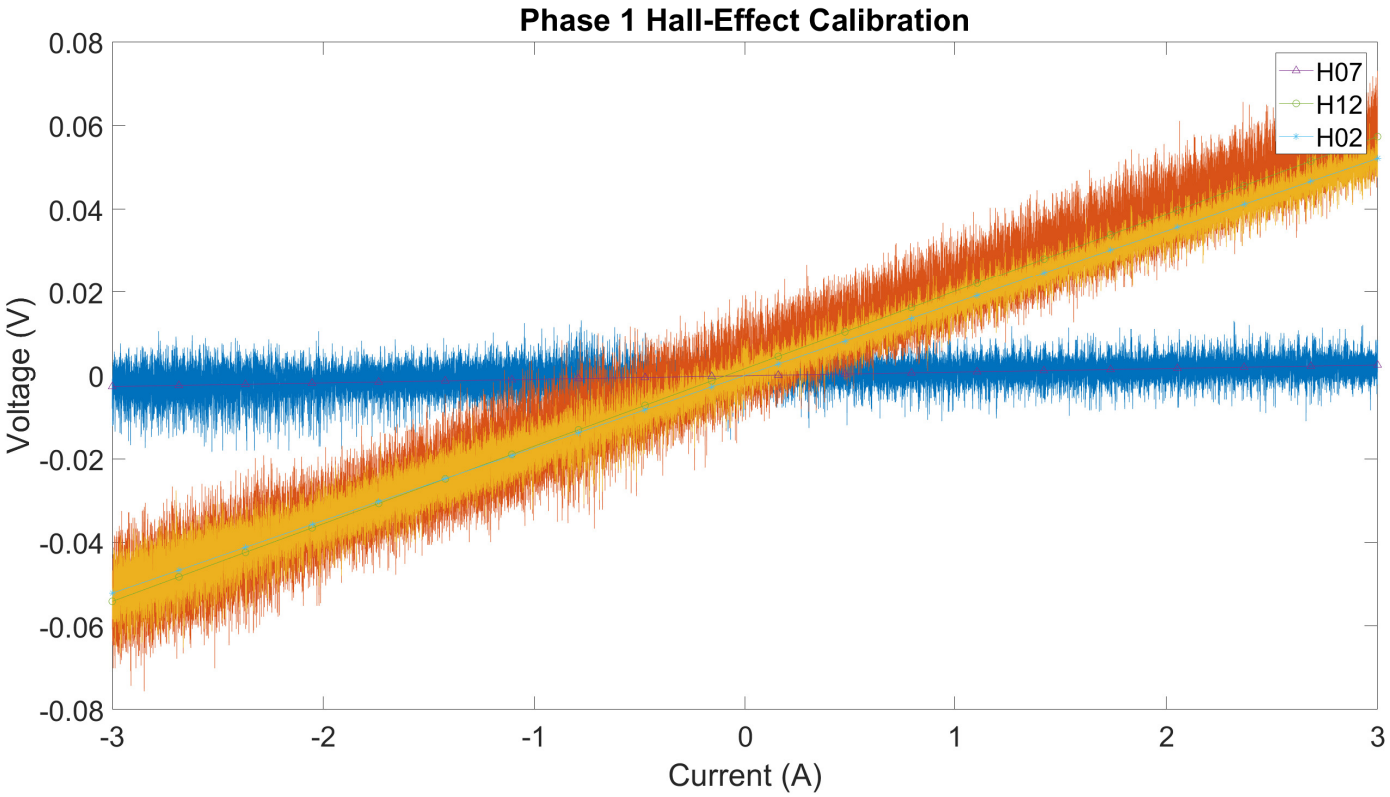


Figure 4.7: Hall-effect sensor response to phase 1 current

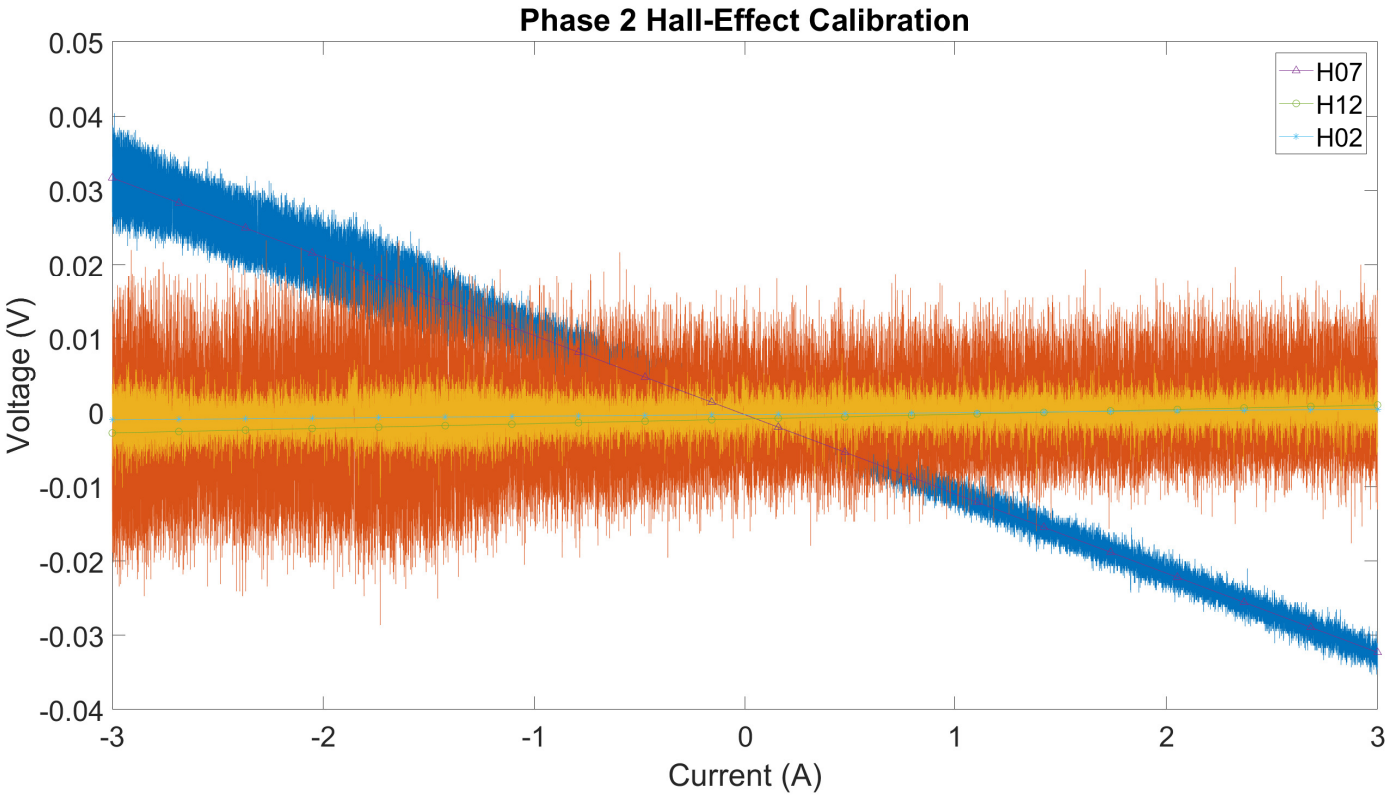


Figure 4.8: Hall-effect sensor response to phase 2 current

4.4 Motor Speed/Hall-Effect Sensor Interaction

To confirm that the speed of the motor was not impacting the Hall-effect sensors, data was collected while the motor was accelerated. To facilitate this, the motor was run with a purposefully long acceleration time to ensure a proper spread of data collection. This also helps ensure that the phase between the rotor angle and current vector stay approximately constant by allowing it to settle. Once the data was collected, the Hall-effect sensor data was plotted against the motor speed while the motor was at the same point during each rotation. To accomplish this, the motor position was assumed to be constant whenever the input signal crossed 0 V. In this way, eighteen distinct positions on the rotor were tracked for changes to the Hall-effect sensor based on motor speed. The corresponding data can be seen in Figure 4.9. Aside from a small bump at 6 rps, near what may be a resonant frequency of the motor, it is clear that speed does not impact the response of the Hall-effect sensor in a meaningful way.

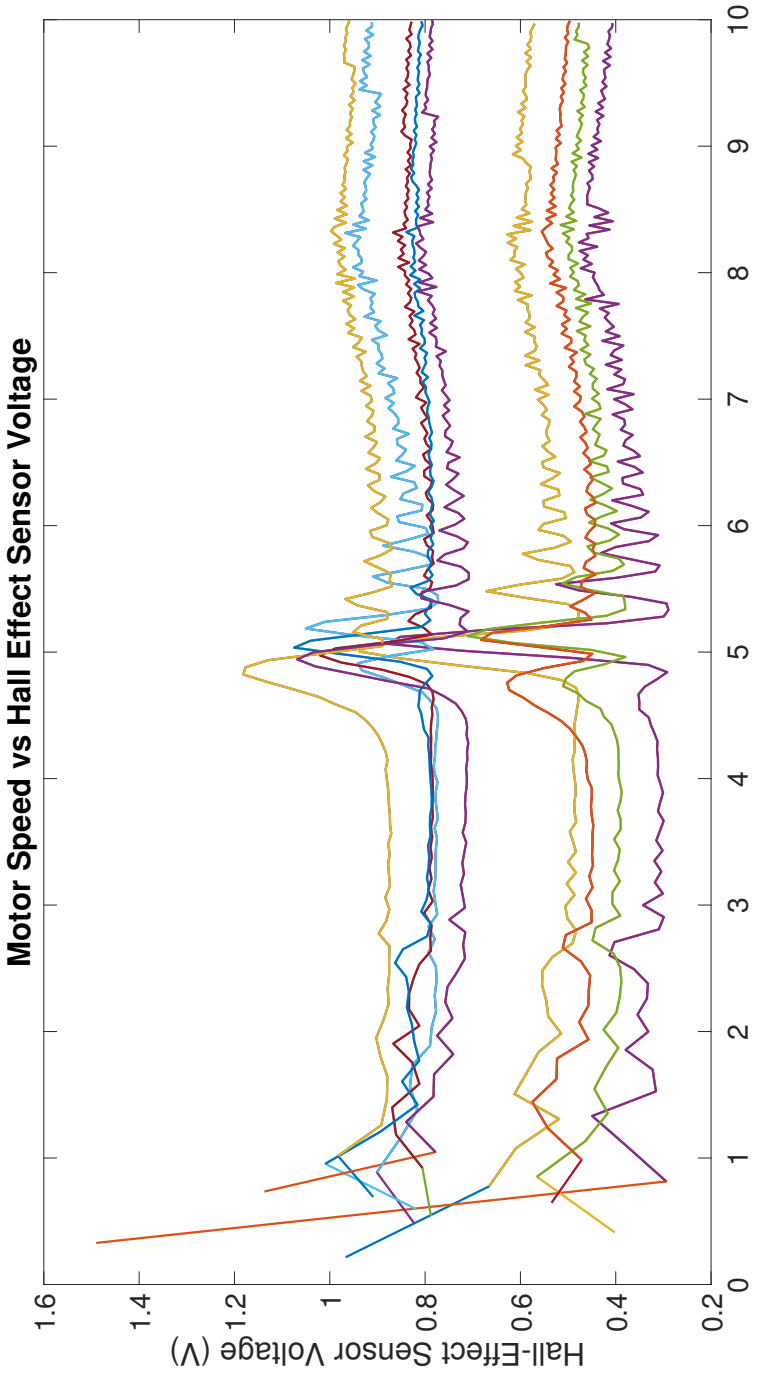


Figure 4.9: Hall-effect sensor response to motor speed

4.5 Motor Back-EMF Measurements

The back-EMF induced in the motor was calculated based on the motor's phase's electrical properties, the specified current, and measured voltage. The current is known due to the transconductance amplifier and the voltage across each phase was measured. Figures 4.10–4.11 show the back-EMF of phase 1 whereas Figures 4.12–4.13 show the back-EMF of phase 2. Additionally, the back-EMF generated during the motor's deceleration after the armature currents to the motor were removed can be seen in Figure 4.14. As is clearly seen in these figures, the back-EMF is directly proportional to the motor speed. By finding the peaks of the back-EMF and comparing it to the known speed at these times, the relationship between the two can be mathematically developed. This can be clearly seen in Figure 4.15 where the linear curve fit coefficients are given in Table 4.5. Though there is a disparity in the strength of the back-EMFs from each phase, this difference reflects the same condition encountered during the torque measurements

Table 4.5: Back-EMF vs motor speed curve fit characteristics

	Slope	**Y-Intercept**	**R-Squared**
Phase 1	0.009115	0.2521	0.9920
Phase 2	0.008774	0.5867	0.9900

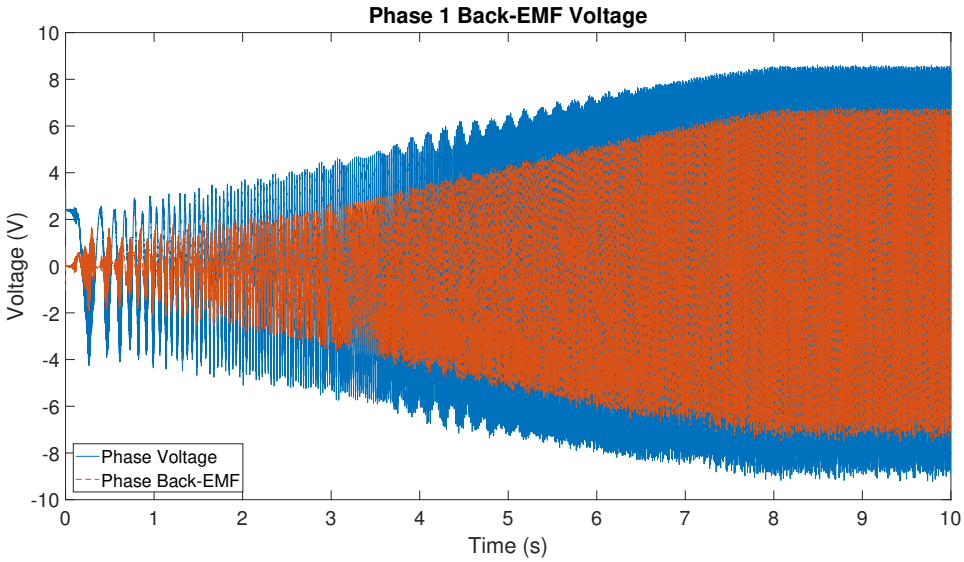


Figure 4.10: Back-EMF of phase 1.

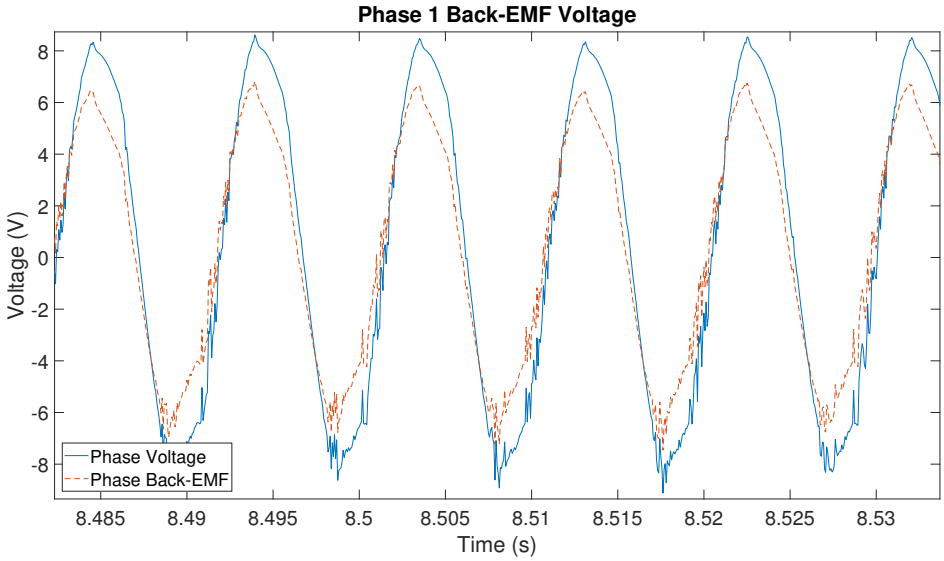


Figure 4.11: Back-EMF of phase 1.

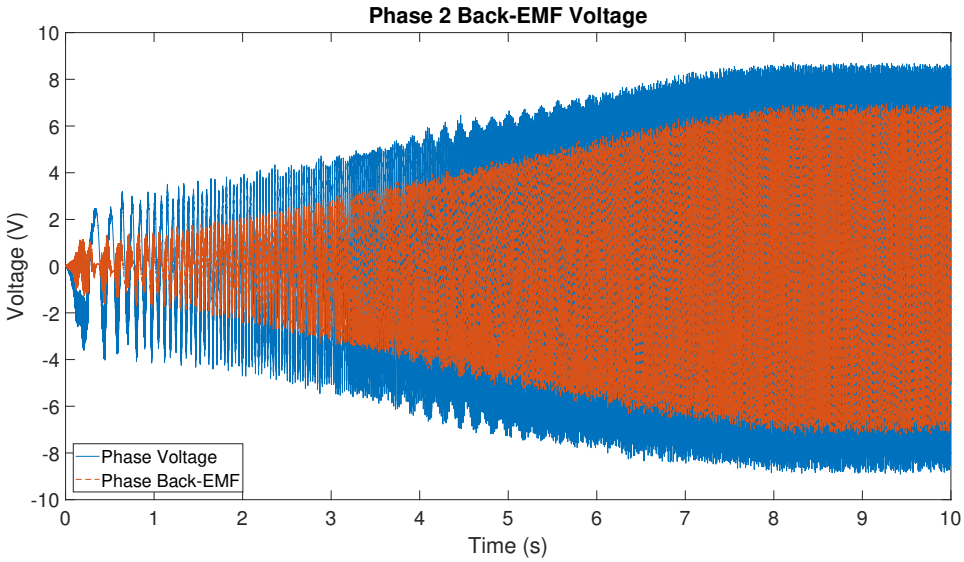


Figure 4.12: Back-EMF of phase 2.

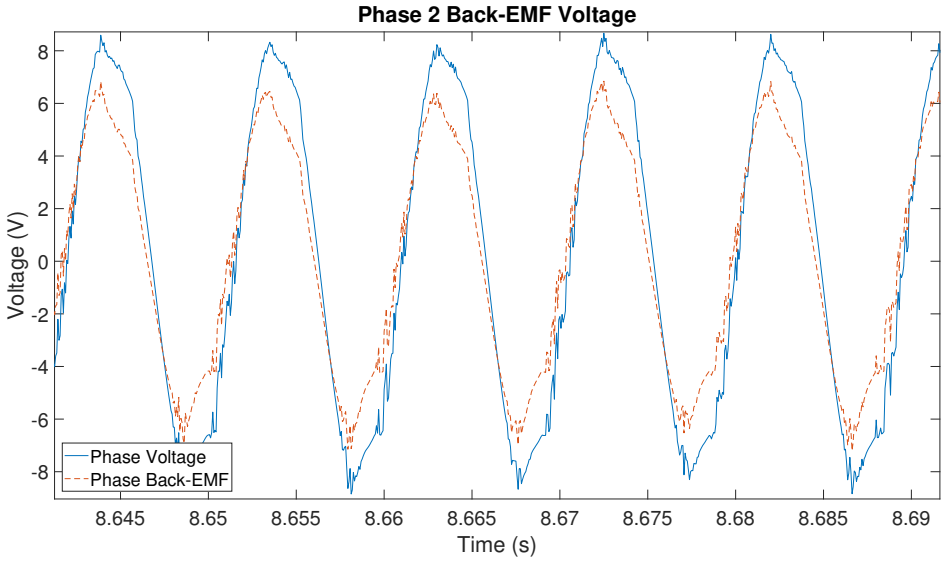


Figure 4.13: Back-EMF of phase 2.

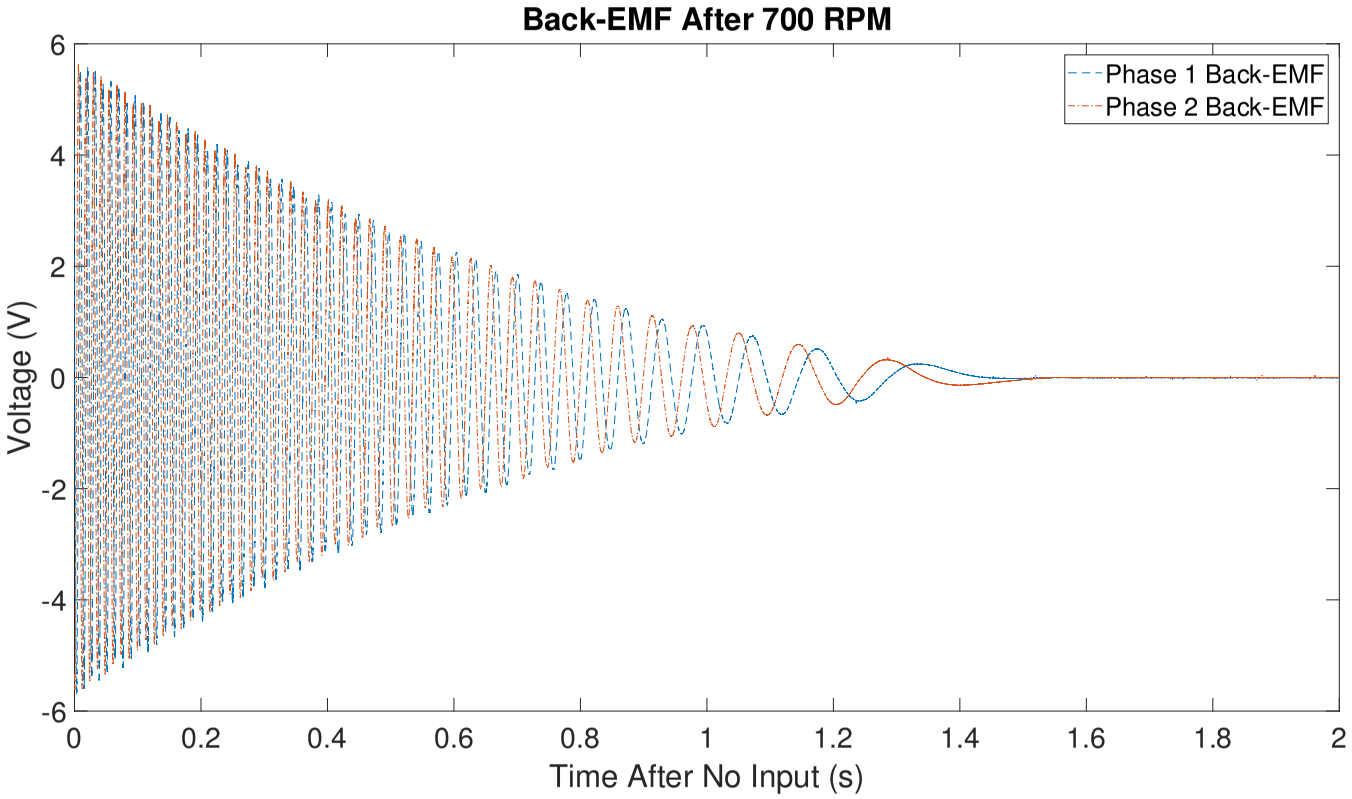


Figure 4.14: Back-EMF after the armature currents were removed.

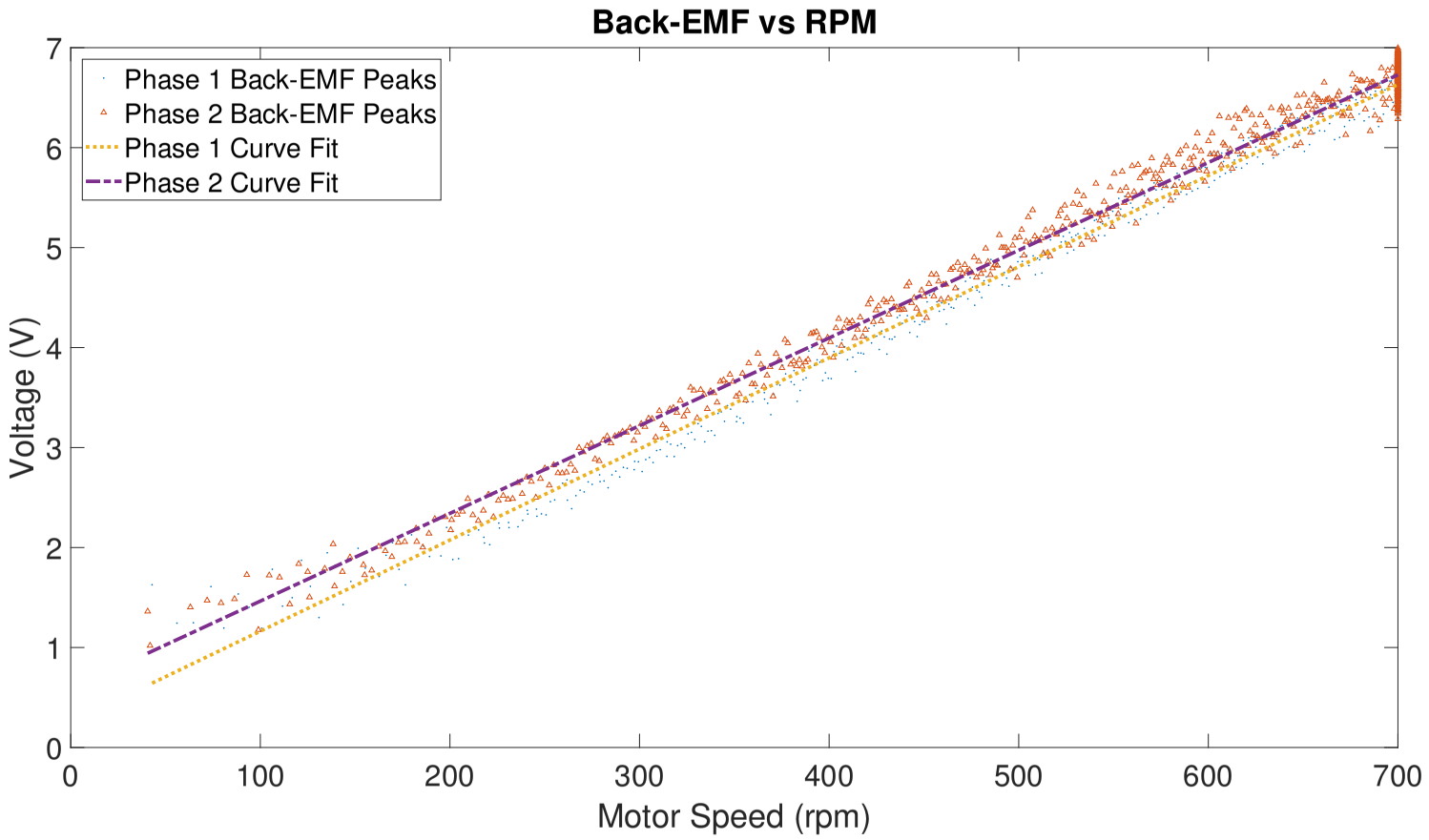


Figure 4.15: Back-EMF vs motor speed.

4.6 Hall-Effect Sensor Positioning

4.6.1 Initial Assumed Velocity Procedure

The next step is to verify the high-precision positioning characteristics. As the motor does not have an encoder, being that it is an external-rotor motor, the use of the system's Hall-effect sensors is critical. The idea is that by determining the voltage from the Hall-effect sensors over the full rotation of the motor, the position of the motor can then be calculated based on a measured voltage from the sensors. As the rotor is embedded with thirty-six magnets, or nine pitches of Halbach arrays, the Hall-effect sensors will experience nine repetitions of magnetic field peaks per revolution of the rotor. This means that for 360° of applied AC signal, the motor will rotate 40°. This is illustrated in Figure 4.16.

The key here is that by taking a large number of samples during the motor's steady-state operation, a distribution of data points can be found that correspond to rotational angles of the motor. In Figure 4.16, the phase signal frequency was specified as 90 Hz which means the motor should be rotating ten revolutions per second (as the motor is synchronous, the speed is known precisely based on the frequency of the input signal), and the sampling rate is 32000 samples/s. This means that for each rotation of the motor 3200 data points should have been collected. This was repeated over fifty-two seconds such that a data set is obtained where the 3200 assumed distinct positions of the motor are measured 520 times via the Hall-effect sensors. As can be shown in Figure 4.17, the distribution of each of these position sets results in a set of data points that are clustered together with few outliers for each set.

During the course of data analysis, it was observed that larger samples of data resulted in a larger spread of data points. This was calculated by finding the sample standard deviation of each of the assumed 3200 positions around the motor. As the standard deviation at each position should get smaller with an increasing data sample (in this case accom-

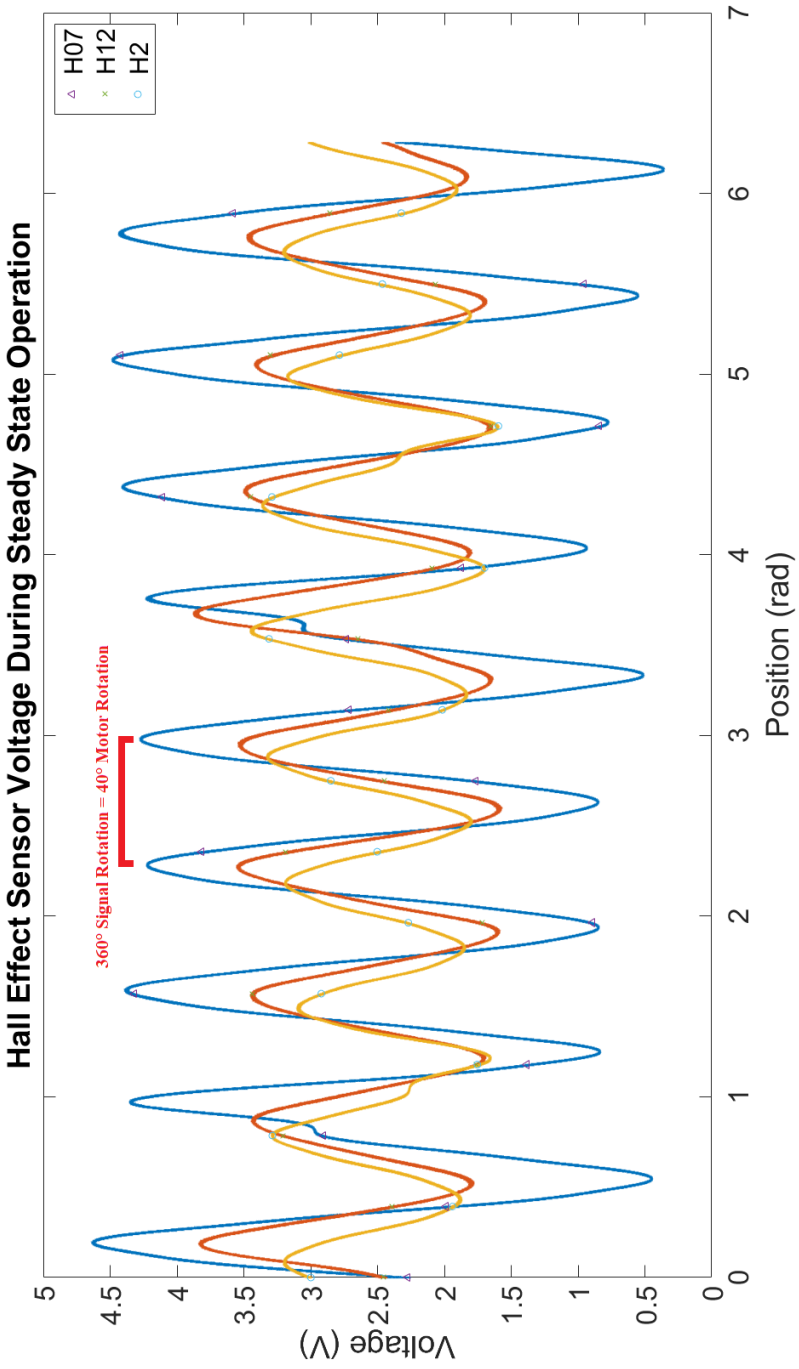


Figure 4.16: Hall-effect sensor data from steady state operation of motor.

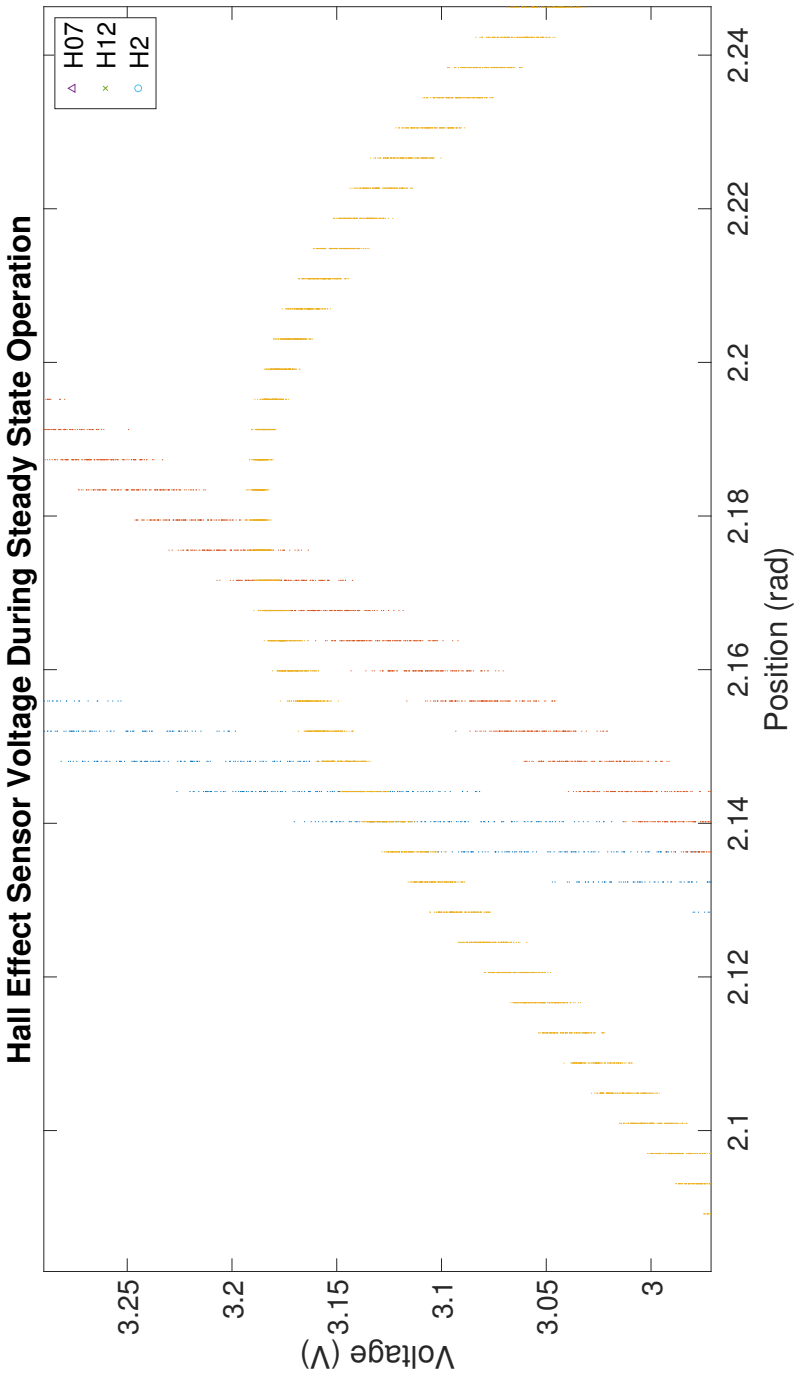


Figure 4.17: Hall-effect sensor data with position sets visible.

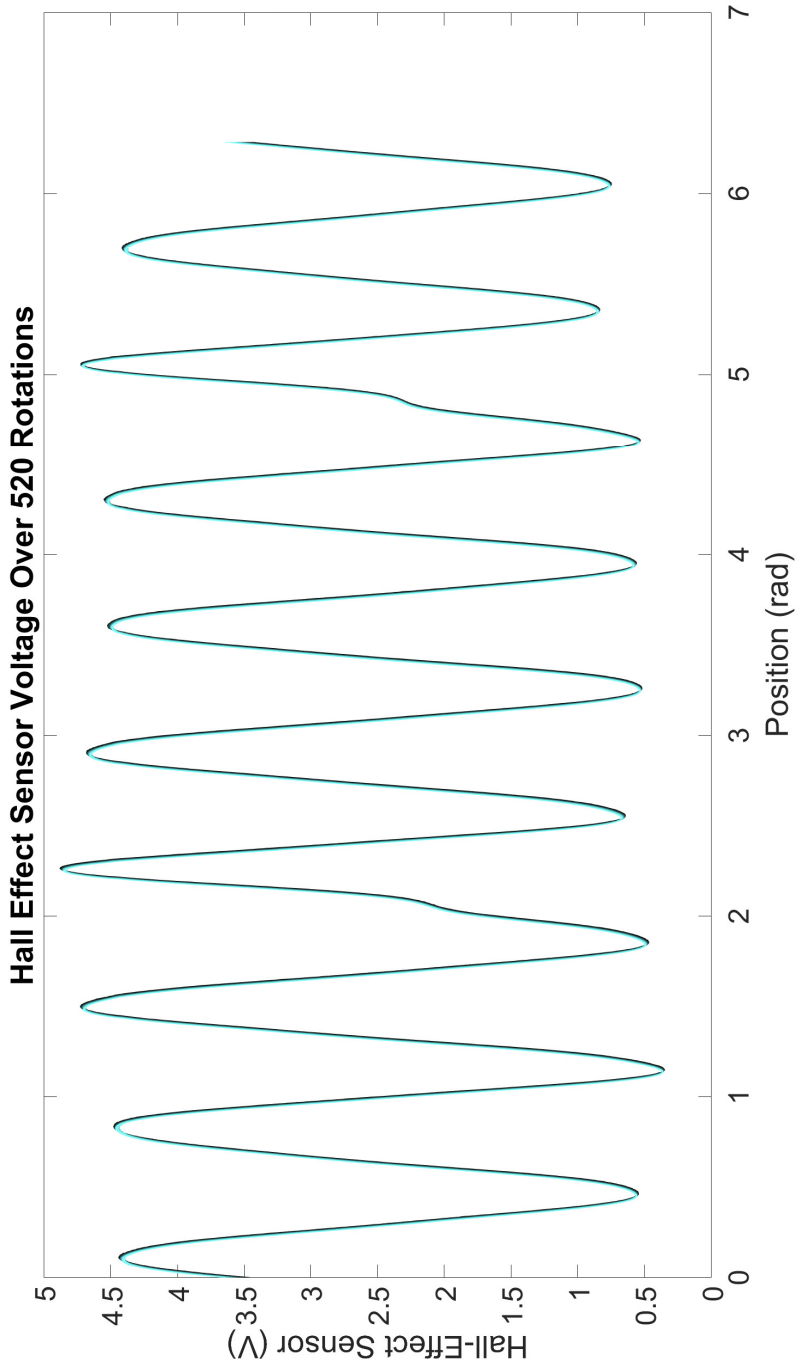


Figure 4.18: Hall-effect sensor data with increasing color intensity over time demonstrating phase shift.

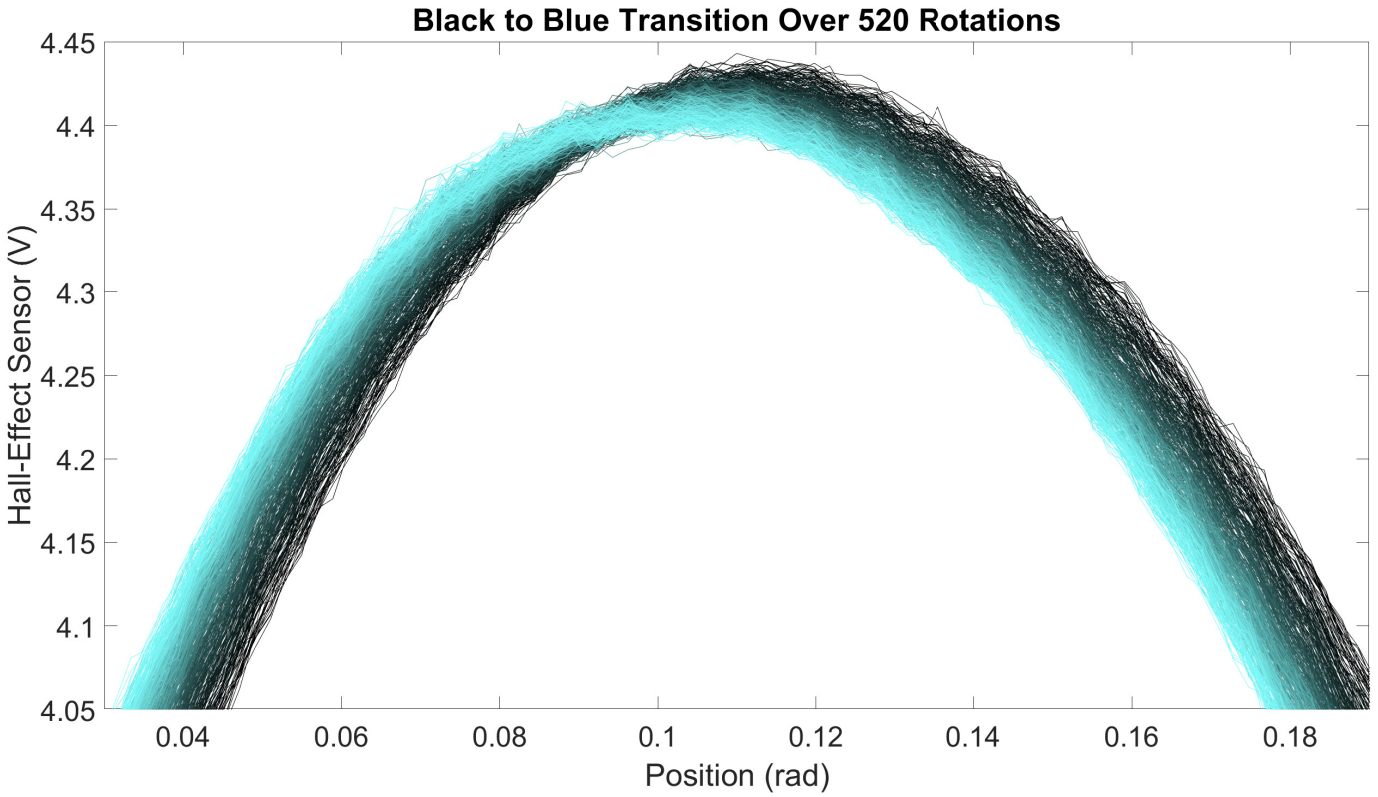


Figure 4.19: Hall-effect sensor data with increasing color intensity over time demonstrating phase shift.

plished by running the motor for longer periods of time), it became apparent that there was a problem. Further inspection of the data yielded a continuous phase shift over time. By plotting a single Hall-effect sensor's data with an increasing color intensity over the course of a number of rotations, this phase shift becomes readily apparent as can be seen in Figure 4.18 and to a much clearer extent in Figure 4.19.

4.6.2 Time Interpolation Compensation

By analyzing the input signals to the coil phases, the actual frequency of the input signals was determined. This was done by finding the times during operation when these signals crossed 0 V and then calculating the time difference between each of these occurrences. As this time represents half of the signal's current period, the speed of the motor can be indirectly calculated. The exact algorithms that were developed can be found in Appendix A. This method was used to determine that the measured DAQ output signal frequency was 89.888 Hz during the motor's steady-state operation. That is, once the motor is up to speed and the DAQ output signal frequency should be 90 Hz, the signal is actually slightly slow. Despite the fact that the signal timestamps used to calculate the signal frequency were of a greater resolution, the signal frequency varied slightly around this value. Though this represents an error of only about 0.125%, it must be corrected for by no longer assuming the prescribed frequencies are correct. It is important to note that though the DAQ board's output and input clocks are slightly out of sync, the input clock that timestamped the measured data will be used to define the motor's speed. In this way, it will be assumed that the output clock is slow and the input clock represents the actual time. This is acceptable, because any deviation the input clock has from a true clock is small and will be applied linearly across all data points. It is also important to emphasize that even if the output and input clocks are not out of sync and the results are caused by some other error, the error manifests itself linearly over time and also affects the input and

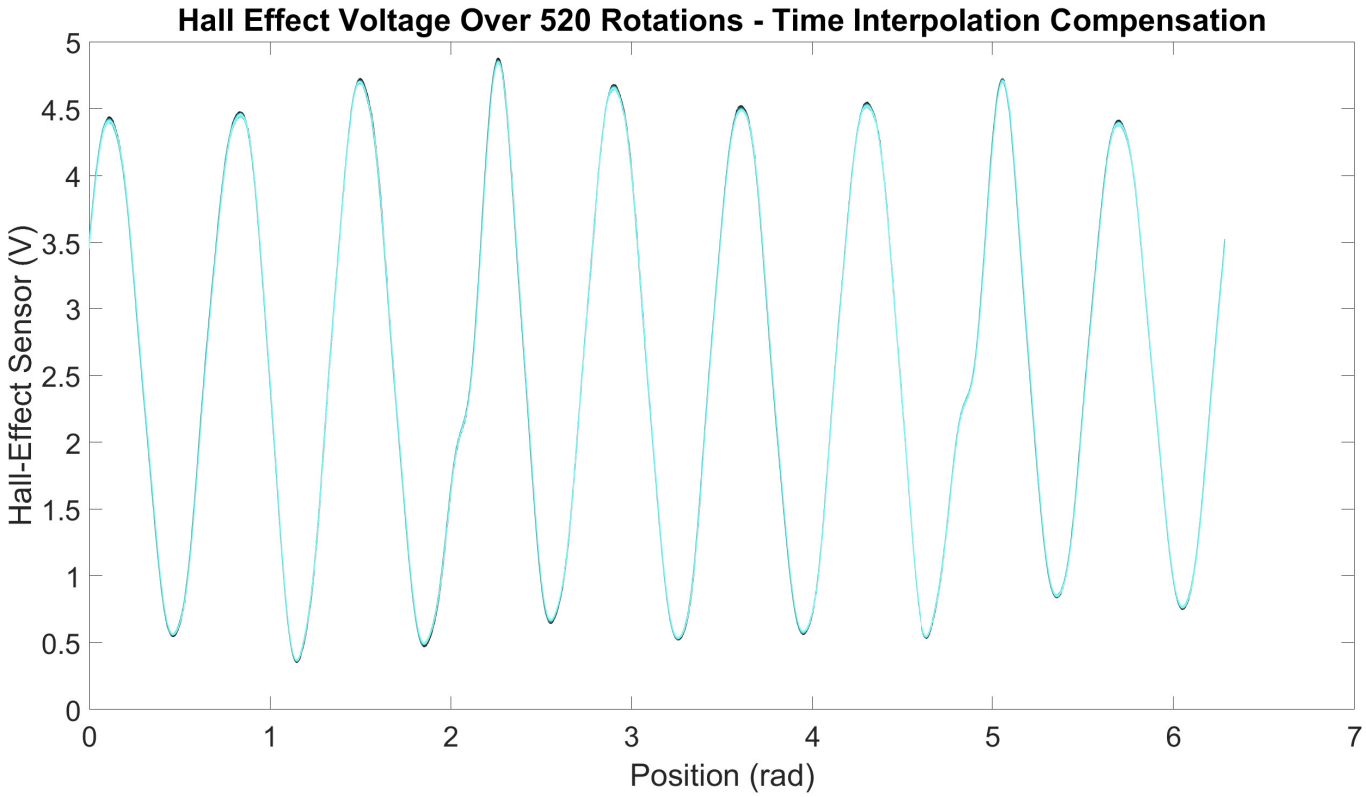


Figure 4.20: Time compensated Hall-effect sensor data with increasing color intensity over time.

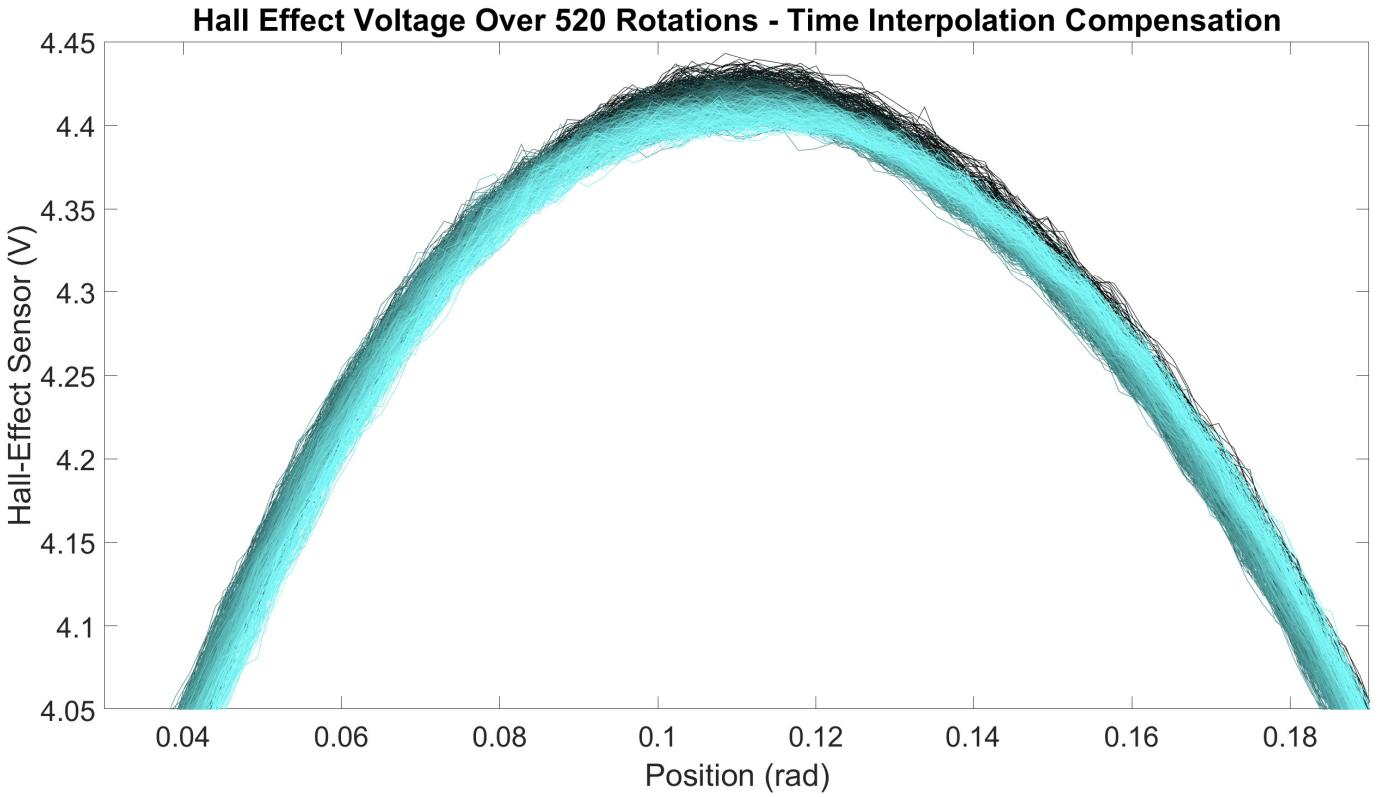


Figure 4.21: Time compensated Hall-effect sensor data with increasing color intensity over time.

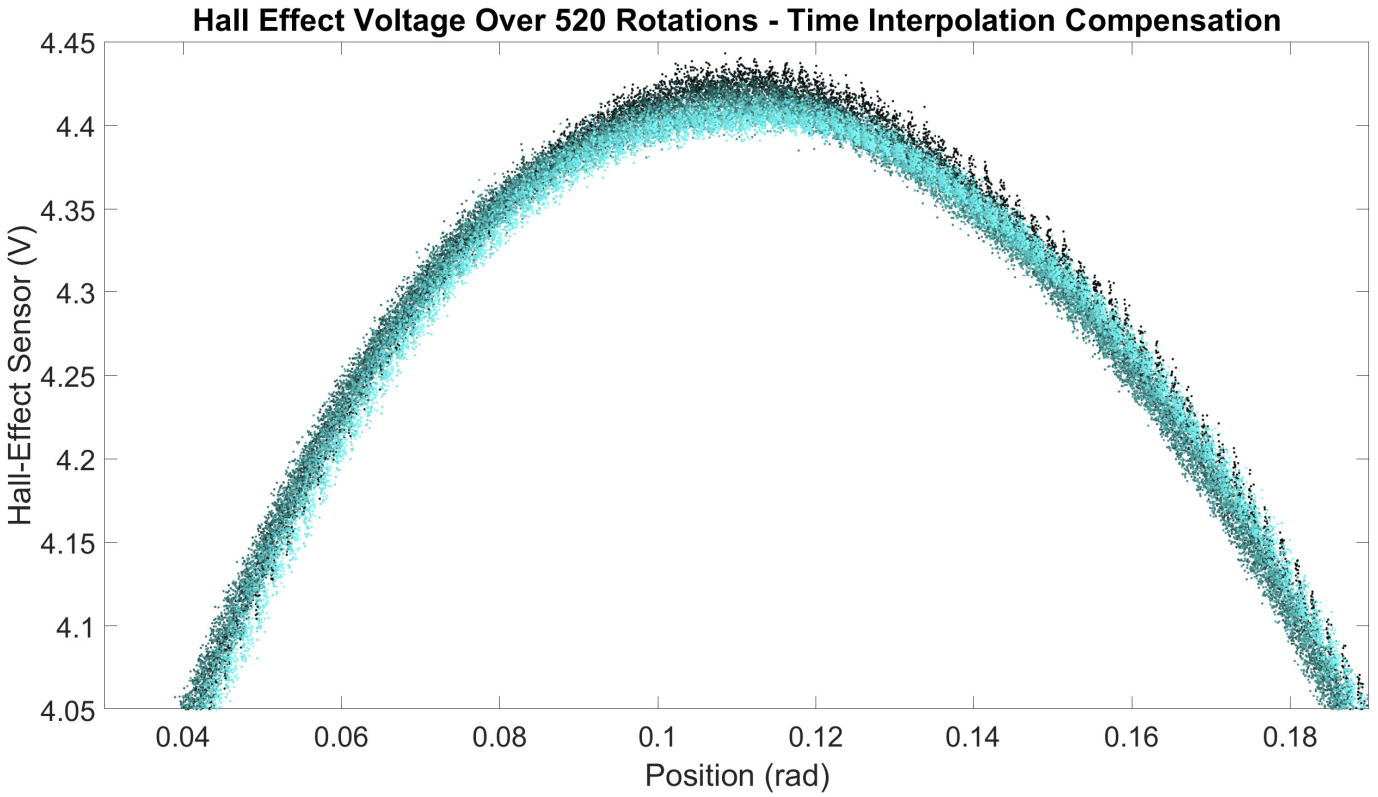


Figure 4.22: Time compensated Hall-effect sensor data with increasing color intensity over time.

output signals unevenly. This means that treating the problem as out of sync clocks should be a valid approach. If this were a simple phase lag problem, the lag would not manifest as it would not increase over time.

With the corrected motor speed, the actual rotational position of the motor can be found during the course of the run. By once again overlaying the Hall-effect voltages with the correct position information, the phase shift can be seen to be eliminated in Figures 4.20 and 4.21. This is further evidenced in Figure 4.22 where the data points are graphed without lines connecting them. In this Figure, it is readily apparent that the clock does not match up perfectly over the course of multiple runs as the plot appears striated. These striations show that the compensation is working as the data points line up well and display a consistent and expected pattern.

4.6.3 Linearization of Data

In order to determine the position of the motor from the Hall-effect sensor data, a linearization procedure was determined to allow for simple position estimation. This was done by setting a voltage range and removing the points outside this region. The results of this can be seen in Figure 4.23. It is important to note that to assist in the calculations from here on out two operations were performed on the data. First, the data was shifted to be centered around zero which will be easily re-shifted at the end of the procedure. Second, the first incomplete section of data was removed. That is, the first rotation of data through the linearized region that did not extend across the full data range. This data can be seen in Figure 4.20 as the data from 0 rad to approximately 0.1 rad during the first rotation. This loss of a minimal fraction of the data was deemed acceptable and results in Figure 4.23 and subsequent figures having a position shift compared to previous plots.

With the linearized regions of the data identified, a linearized model was developed for each region of the plot in Figure 4.23. This results in eighteen different linear equa-

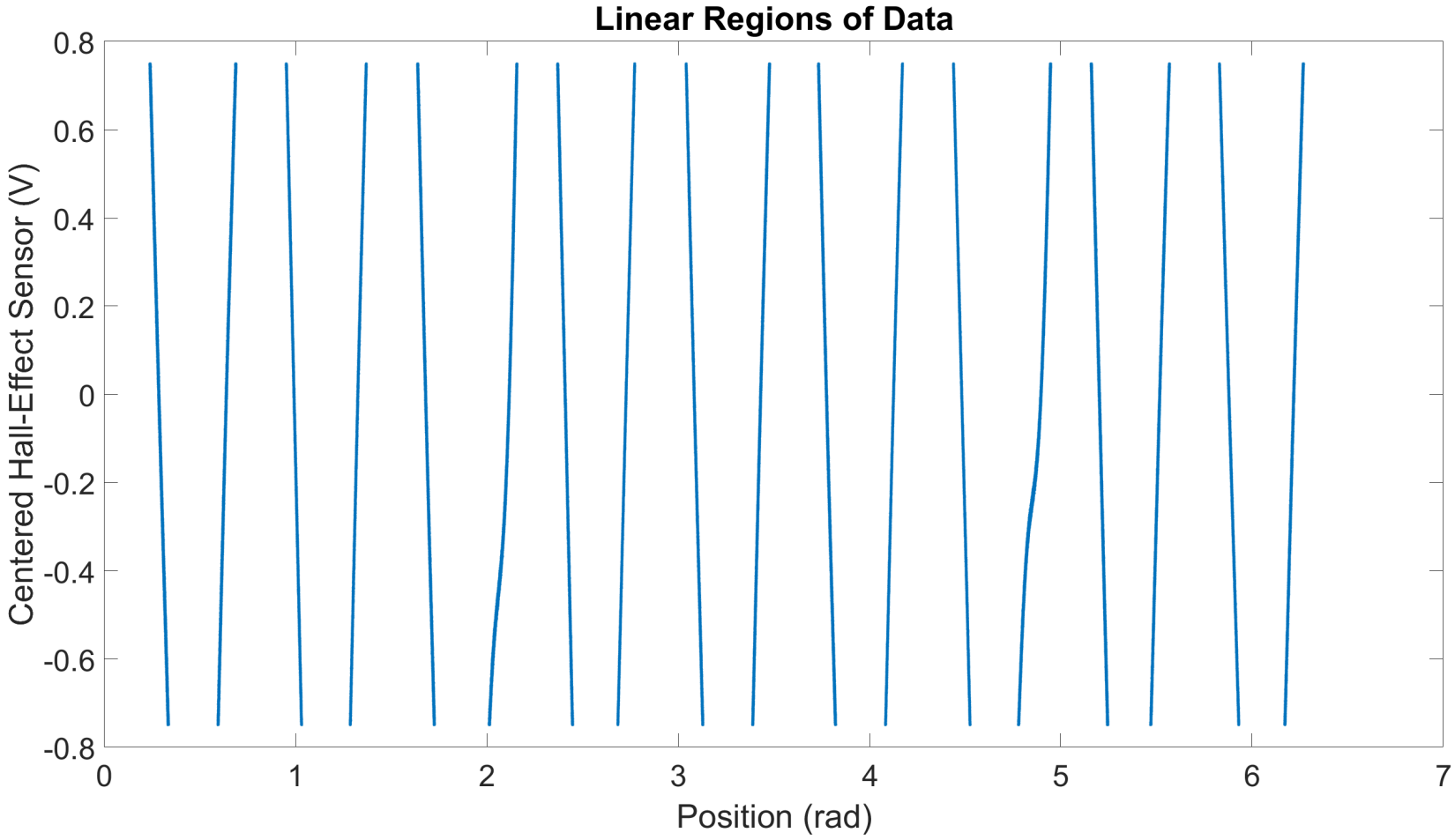


Figure 4.23: Linear regions of time compensated Hall-effect data.

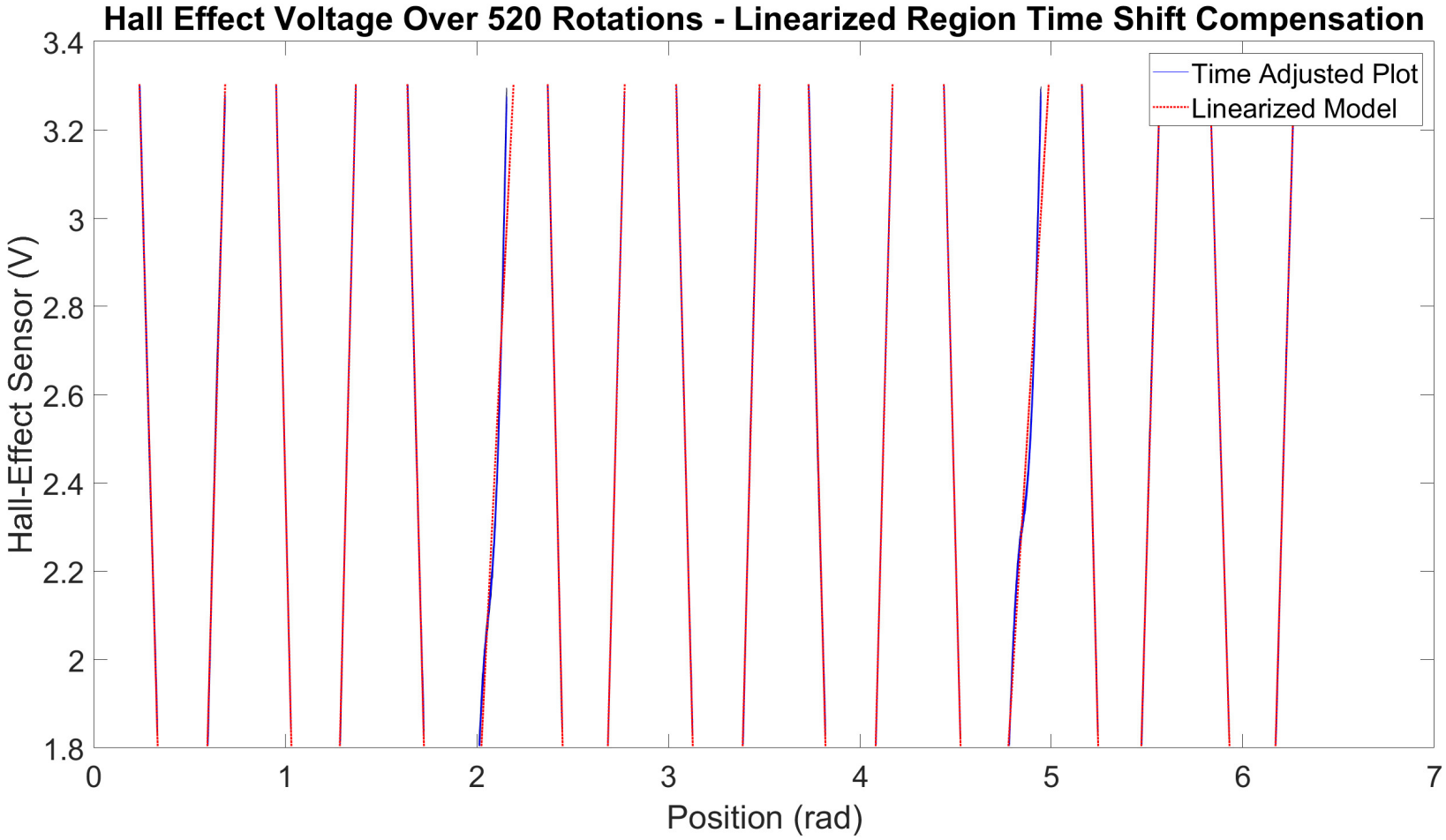


Figure 4.24: Linear model overlaid on time compensated Hall-effect data.

tions. These linear equations were developed to go from a voltage, to a corresponding position for a given segment. In other words, even though voltage is the dependent variable in establishing a positional mapping, in testing, position is the dependent variable. The overlaying of these models on top of the linearized data set can be seen in Figure 4.24.

4.6.4 Linearization of All Data

With the method for developing the linearized correlation between sensor data and position completed, the same method was then applied to all three Hall-effect sensors present in the motor. In the following plots, the data has been re-centered around its original value and is no longer centered around zero. Though only one sensor might be all that is technically necessary, multiple Hall-effect sensors will give rise to greater positional accuracy and redundancy. The linear regions for the entirety of the Hall-effect sensor data are shown in Figure 4.25 and the linear models overlaid on this data is shown in Figure 4.26. Additionally, the full properties of the linearized models are given for Hall-effect sensors 7, 12, and 2 in Tables 4.6, 4.7, and 4.8 respectively.

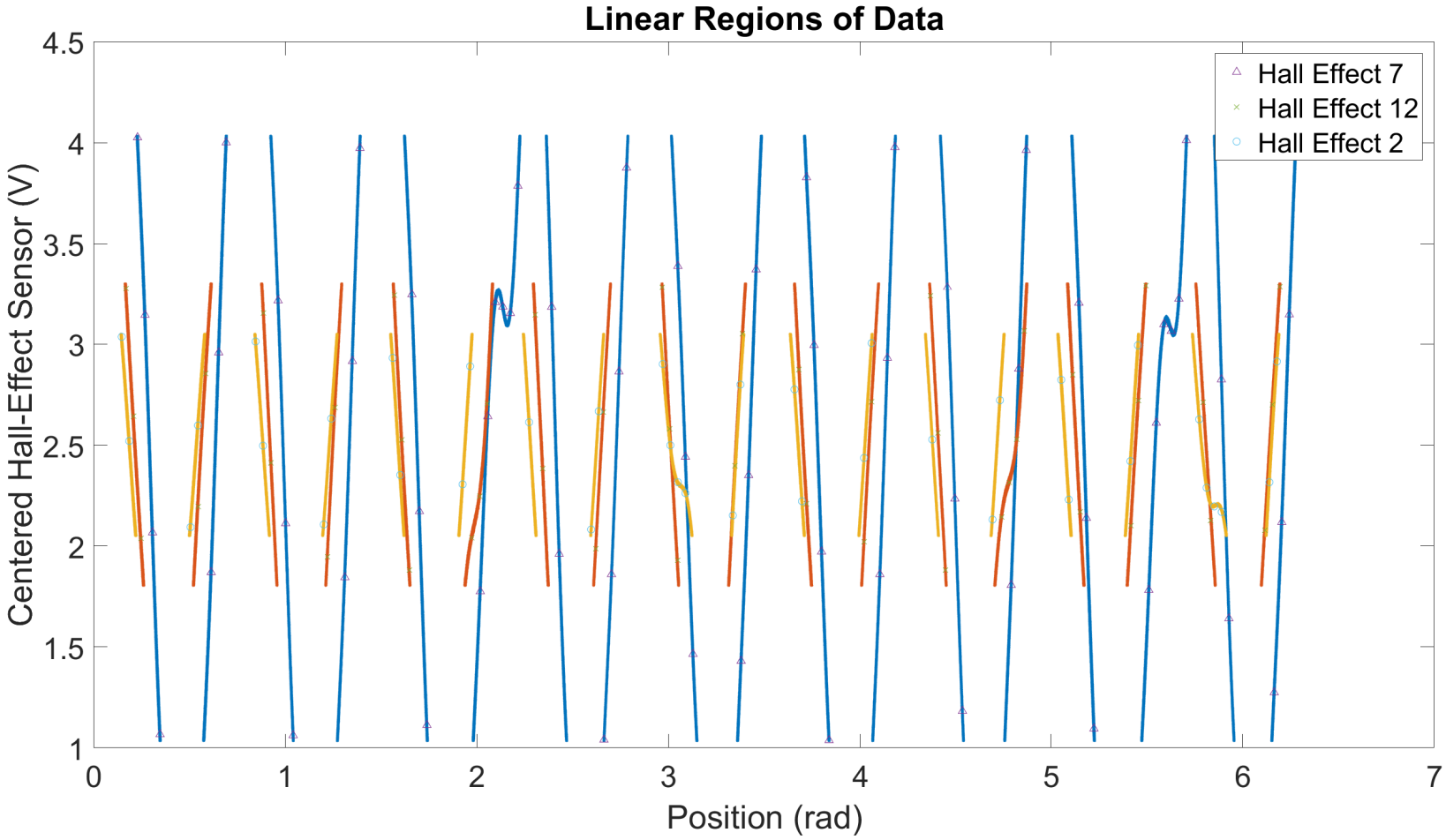


Figure 4.25: Linear regions of all Hall-effect data.

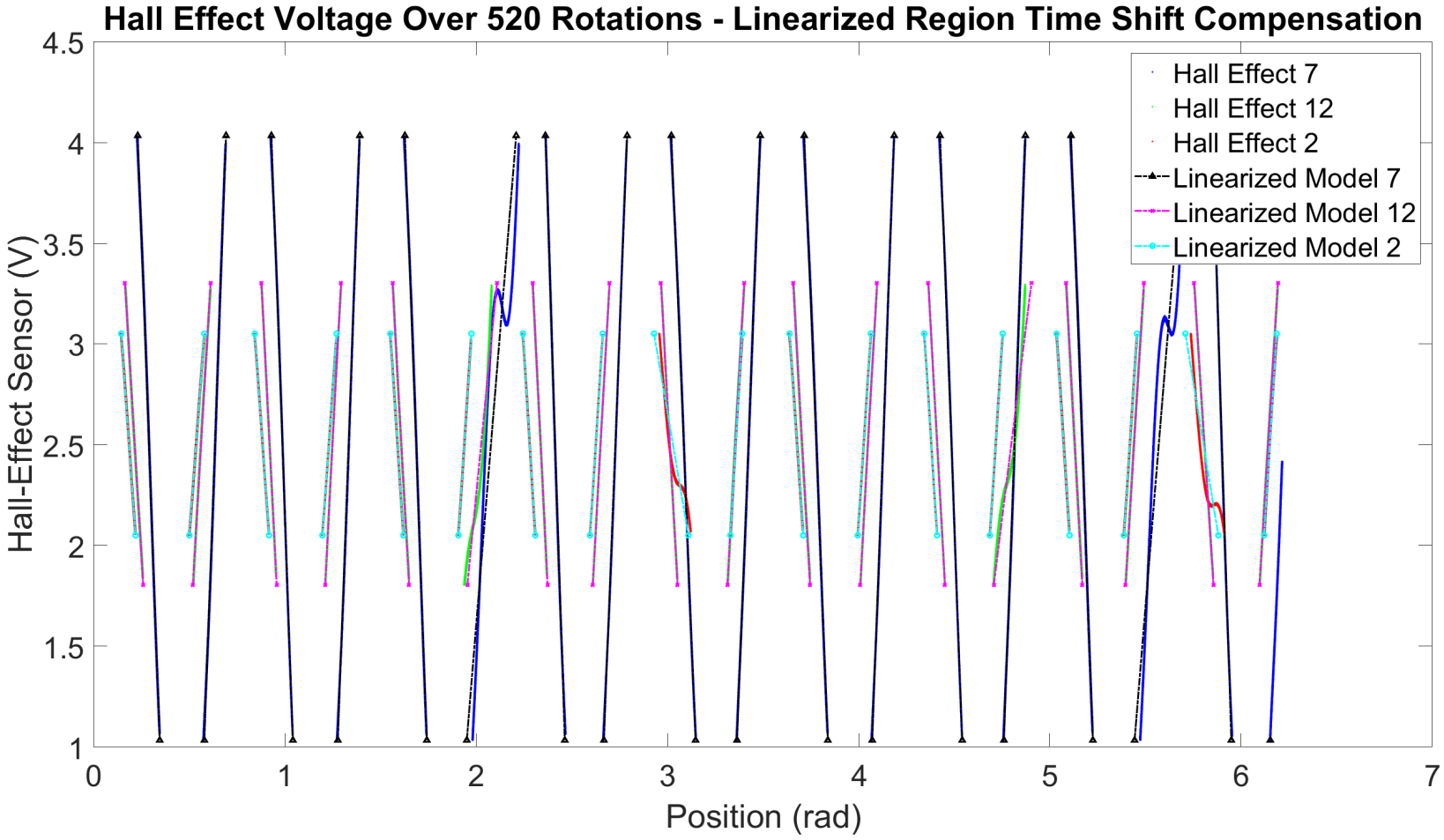


Figure 4.26: Linear models overlaid on all Hall-effect data.

Table 4.6: Linearized model properties for Hall-effect sensor 7.

Region	**Slope**	**Y-Intercept**	**R Squared**	**RMSE**
1	−0.0700	0.4114	0.9950	0.0046
2	0.0710	0.3968	0.9947	0.0049
3	−0.0708	1.1026	0.9945	0.0050
4	0.0702	1.0913	0.9950	0.0047
5	−0.0681	1.7904	0.9953	0.0043
6	0.1147	1.7243	0.9320	0.0237
7	−0.0597	2.5115	0.9866	0.0065
8	0.0682	2.4881	0.9951	0.0044
9	−0.0708	3.2026	0.9945	0.0049
10	0.0649	3.1989	0.9966	0.0035
11	−0.0640	3.8799	0.9963	0.0036
12	0.0598	3.9189	0.9959	0.0035
13	−0.0618	4.5836	0.9914	0.0053
14	0.0564	4.6216	0.9969	0.0028
15	−0.0587	5.2625	0.9970	0.0029
16	0.1085	5.2478	0.9518	0.0189
17	−0.0568	6.0004	0.9864	0.0061
18	0.0755	5.9737	0.9949	0.0021

Table 4.7: Linearized model properties for Hall-effect sensor 12.

Region	**Slope**	**Y-Intercept**	**R Squared**	**RMSE**
1	−0.1269	0.4999	0.9973	0.0029
2	0.1292	0.2004	0.9928	0.0048
3	−0.1053	1.1494	0.9984	0.0018
4	0.1145	0.9213	0.9957	0.0033
5	−0.1211	1.8800	0.9960	0.0033
6	0.1482	1.5800	0.9735	0.0096
7	−0.1154	2.6084	0.9940	0.0040
8	0.1327	2.2732	0.9980	0.0027
9	−0.1253	3.2955	0.9975	0.0028
10	0.1325	2.9854	0.9948	0.0043
11	−0.1319	4.0024	0.9972	0.0031
12	0.1335	3.6745	0.9952	0.0041
13	−0.1320	4.7163	0.9974	0.0030
14	0.1785	4.2737	0.9835	0.0090
15	−0.1186	5.4098	0.9933	0.0043
16	0.1428	5.0436	0.9963	0.0039
17	−0.1382	6.1248	0.9978	0.0028
18	0.1235	5.7923	0.9964	0.0023

Table 4.8: Linearized model properties for Hall-effect sensor 2.

Region	**Slope**	**Y-Intercept**	**R Squared**	**RMSE**
1	−0.1759	0.5608	0.9928	0.0046
2	0.1805	0.0198	0.9947	0.0040
3	−0.1625	1.2298	0.9958	0.0032
4	0.1564	0.7713	0.9943	0.0035
5	−0.1423	1.8855	0.9951	0.0031
6	0.1456	1.5116	0.9890	0.0047
7	−0.1306	2.5485	0.9973	0.0020
8	0.1393	2.2111	0.9965	0.0025
9	−0.2569	3.6113	0.9520	0.0149
10	0.1331	2.9856	0.9882	0.0042
11	−0.1733	4.0529	0.9960	0.0033
12	0.1662	3.5539	0.9966	0.0030
13	−0.1710	4.7582	0.9944	0.0040
14	0.1727	4.2245	0.9929	0.0045
15	−0.1711	5.4482	0.9948	0.0038
16	0.1745	4.9203	0.9930	0.0045
17	−0.2684	6.4054	0.9188	0.0213
18	0.1272	5.7898	0.9904	0.0029

4.6.5 Linearized Model Prediction Error

Given the voltage range of the linearized model of the Hall-effect sensors (about 3 V for Hall-effect sensor 7) and the fact that each 360° of signal rotation corresponds to 40° of the motor's rotation, the degree per voltage can be calculated. Extending this, given that each 180° of signal input corresponds to only 20° of rotation (as each of the eighteen linear regions in Figure 4.23 comes in a pair that has both an increasing and decreasing voltage component), the precision of the motor positioning based on the Hall-effect sensors can be found to be about 7°/V. Pairing this with the data for each position set allows the distribution of error to be calculated for a given linearized model. Excluding the two outlier root-mean-square error (RMSE) values from each of Tables 4.6−4.8, yields average RMSE values of 0.0043, 0.0040, and 0.0036 rad respectively. These RMSE values correspond to approximately a 0.23° spread for a given position. In other words, utilizing only a single sensor yields a rotational uncertainty for a given position of about 0.23° or given the diameter of the rotor, a linear uncertainty of about 200 μm. An equivalent RMSE value of all of the sensors can be found by using the variance of each sensor. That is

$$\sigma_{tot}^2 = (\sigma_1^{-2} + \sigma_2^{-2} + \sigma_3^{-2})^{-1} \tag{4.7}$$

where the system is assumed unbiased and σ is the RMSE of each sensor. This yields and equivalent RMSE of 0.13° or a linear uncertainty of only 150 μm. This allows for very precise closed-loop position (and by extension velocity) control based on the feedback obtained from the Hall-effect sensors.

5. MOTOR POSITIONAL CONTROL

Now that the motor is fully characterized (particularly with regard to positioning), the motor's positioning controller can be implemented. For starters, given the calculated motor friction force of 0.06 N·m as well as the measured torque constant of 0.086 $\mathrm{N \cdot mA}$ means that in order for rotor to move, a current of more than 0.7 A must be applied. Therefore, this stall-current is actually the base current required for all positioning operations. Rather than regulating current, a current amplitude was set and the angle of the electrical signal was controlled. That is, even though the motor should ideally respond perfectly to the electrical input phase signals, this is not always the case. Instead, an initial set of signals is sent to the motor that would ideally get it to the right position. The main purpose of this is get the motor on the correct linearized segment (as defined in Subsection 4.6.4) as well as to get the motor close to the desired position. From this position, the controller is then implemented. An overview of the positioning process is shown in Figure 5.1, and a block diagram of the positioning controller is given in Figure 5.2.

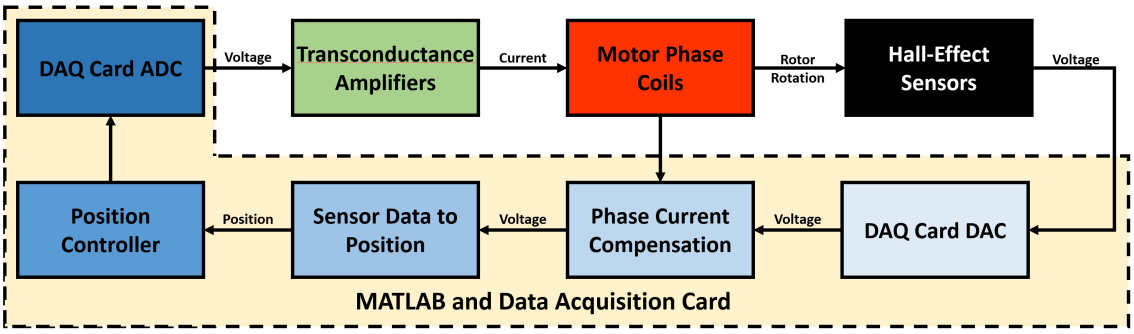


Figure 5.1: High-level controller block diagram.

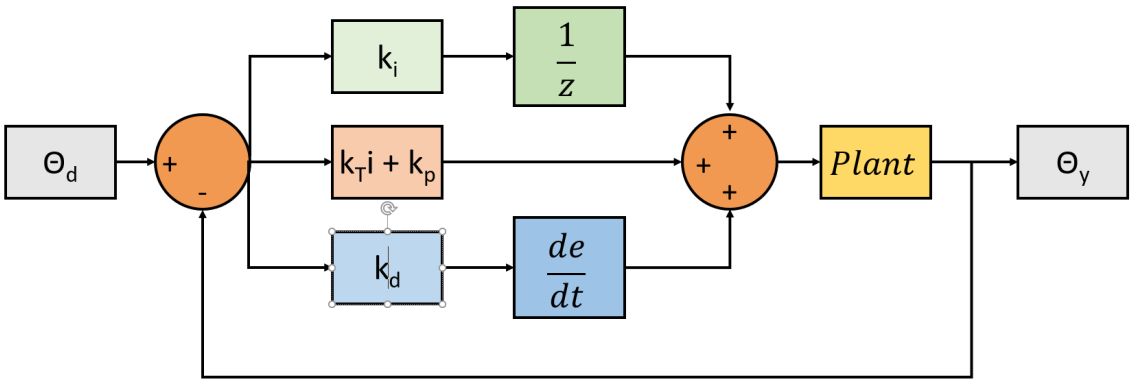


Figure 5.2: Positioning PID controller block diagram.

5.1 Controller Procedure

In order to properly relate the voltage of the Hall-effect sensors and the position of the rotor, the motor must know its absolute position. That is, given the linearization described in the previous section, the system must know what linear segment the rotor is currently on. To achieve this, during the linearization processing, the two points corresponding to the largest and smallest voltages of Hall-effect sensor 7 were identified. Referring to Figure 4.20 it is plain to see that two of the eighteen peaks are consistently and clearly higher and lower (respectively) than the rest of the peaks. By identifying these two peaks and their positional distance from one another, a standard of reference can be developed.

Before the motor can begin controlling its position, it is first rotated around once (in the same direction as it was during the linearization procedure) and the Hall-effect voltages are recorded. During this rotation, each consecutive data point is assigned a position from 0 to 2π. For now, this positional association is arbitrary. Results from this procedure can be seen in Figure 5.3. The maximum and minimum voltages of Hall-effect sensor 7 are then found from this measured data and the distance between these points is compared to the reference distance processed previously. As long as these two distances are sufficiently

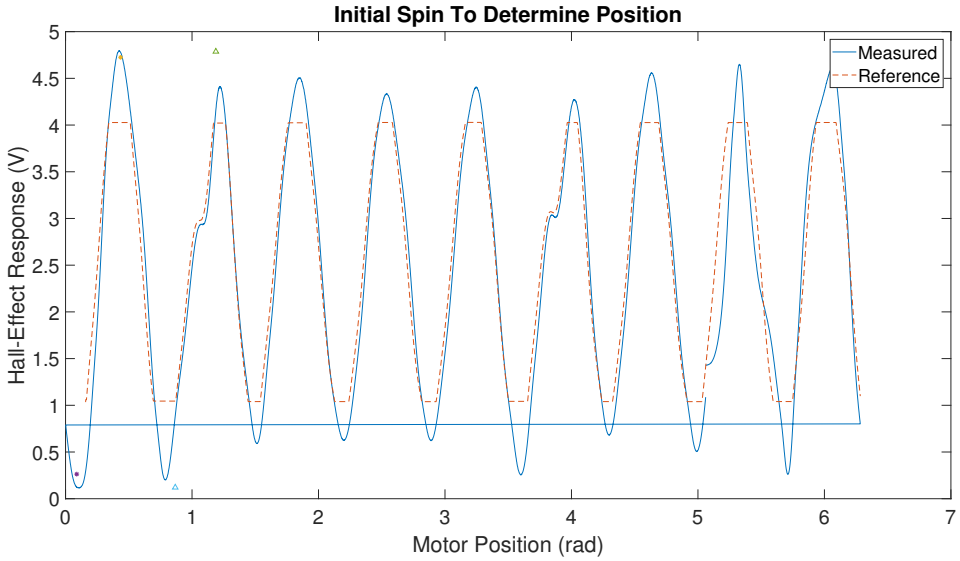


Figure 5.3: Initial spin voltage overlayed on reference voltage.

close, the motor proceeds, otherwise the system stops the procedure. After proceeding, the system then compares the measured voltage positions to the reference voltage positions and uses this calculate its actual position. This procedure and the corresponding results overlayed on top of the linearized position model are shown in Figure 5.4.

Based on a given desired initial position, the motor then moves to this approximate position. Up until this point, it does not matter how the phase angle of the applied signal and the mechanical angle of the rotor relate to one another. Given the system now knows the rotor position as well as the desired position, the system can rotate to this location by simply applying an electrical signal that rotates nine times the difference between the actual and desired position. Once the motor is close to the desired position, the system will establish a relationship between electrical phase angle and mechanical position. Moving forward, any desired change in the angle of the rotor can be made approximately by a change of the electrical phase angle of nine times this value. Following this, the system then repeats the test rotation procedure by comparing peak voltages to confirm its position,

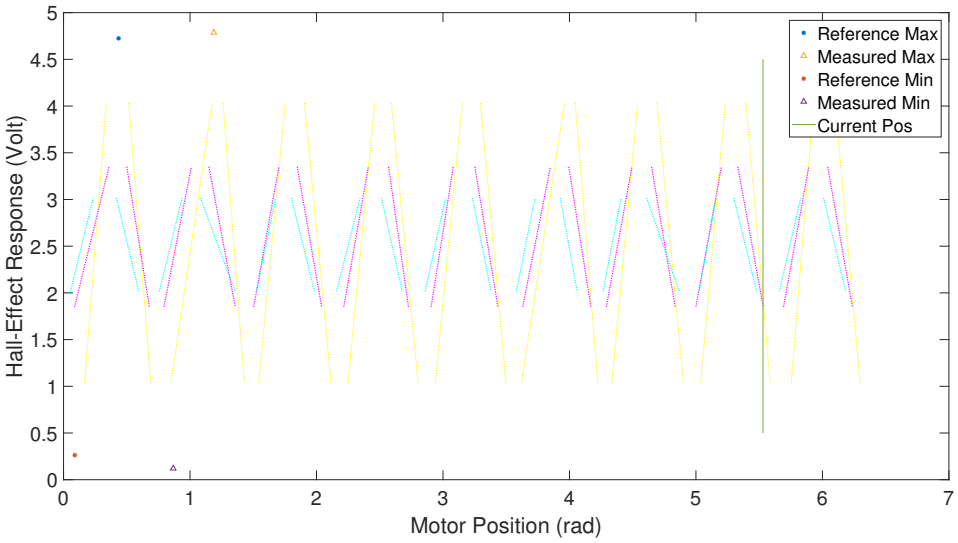


Figure 5.4: Initial spin referenced location over linearized model.

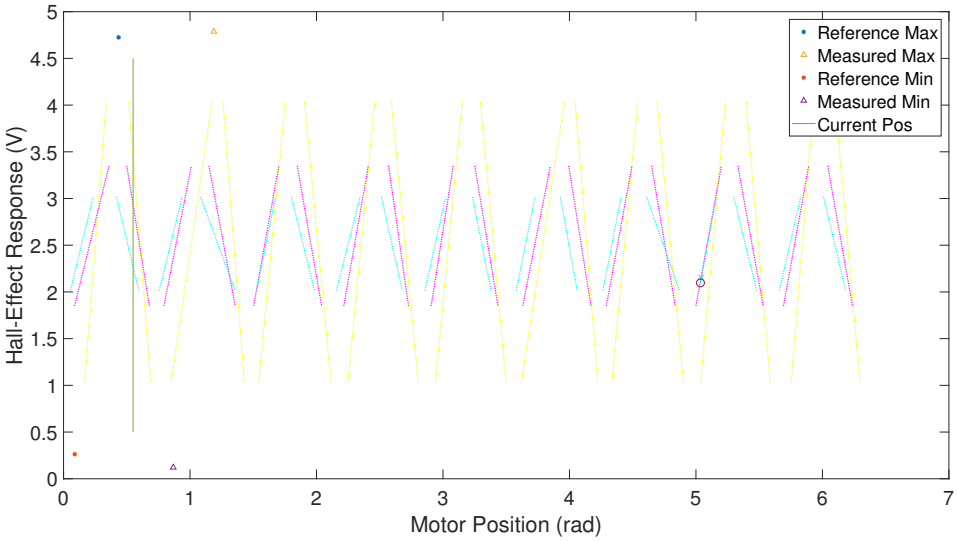


Figure 5.5: Confirmation spin verified location over linearized model.

ensure it is on the correct linear segment, and verify the established rotor angle/phase angle relationship. Figure 5.5 reflects the same information as Figure 5.4 but shows the that the

motor has moved to the desired starting position. It is worth noting that the cross and circle in Figure 5.5 represent the positions determined from Hall-effect sensors 12 and 2 respectively and match up nearly perfectly with Hall-effect sensor 7.

Once this confirmation has occured, the system asks for a user determined position, where this desired position is limited to the linear regions that were determined during the linearization processing. Though the motor can be controlled to operate at any position covered by the linearization procedure, this series of tests will focus only on those positions that are covered by the linearized segments of all three Hall-effect sensors. These positional bounds are listed in Table 5.1. After receiving a desired position, the motor then turns to the correct linear segment and approximate position specified before initiating the positional controller.

Table 5.1: Positions covered by all three Hall-effect sensor linearizations

Lower Bound (rad)	**Upper Bound (rad)**
0.166	0.234
0.515	0.594
0.848	0.938
1.261	1.355
1.543	1.676
1.905	1.984
2.248	2.345
2.592	2.678
2.939	3.021
3.295	3.379
3.630	3.728
4.045	4.061
4.335	4.408
4.692	4.836
5.038	5.161
5.396	5.479
5.753	5.829
6.100	6.184

5.2 Controller Structure

The controller, as implemented through MATLAB and its events system, is broken into three main programs or components. The first program handles all of the initial position determination and user input. Once a desired position has been specified, this program continues running waiting for the next desired position. The second program is an updating event that is triggered whenever data is available from the DAQ board. Due to the nature of the DAQ board and it's integration with MATLAB, data is only collected whenever the board is specified to be outputting signals. This function triggers whenever a certain amount of data is available, stores the collected data, and determines the rotor's position from the Hall-effect sensor voltages. The third program is the controller of the system which uses the position determined by the updating program to implement the controller and adjust the phase signals as necessary.

5.3 Controller Design

In order to design a positional controller for the system, the system dynamics model must be adjusted slightly. As in 4.4 the system model can be given as

$$I\ddot{\theta}(t) = k_T i - T_f \tag{5.1}$$

so long as the current vector is rotated around to correspond with the changing angular position of the rotor. For a constant current vector however, the system dynamics can more accurately be described by

$$I\ddot{\theta}(t) = k_T i \sin(\phi - \theta) - \mathrm{T_f} \tag{5.2}$$

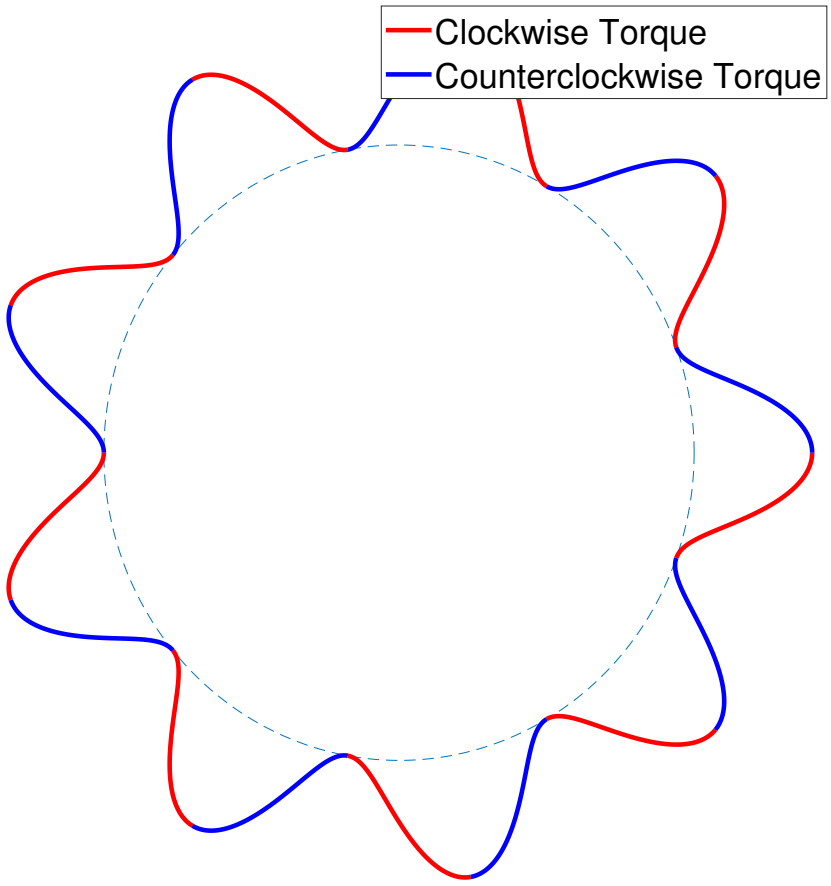


Figure 5.6: Force distribution around rotor for a static current vector.

where ϕ is the angle of the current vector and θ is the position of the rotor. This is due to the sinusoidal distribution of the magnetic field which is illustrated in Figure 5.6. In this diagram, rotating the rotor manually can be thought of as traversing around the curve where a greater radial distance from the base circle corresponds to a greater torque and the nine points tangential to the base circle are where no torque occurs. Similarly, rotating the

current vector would correspond to a rotation of the waveform. Based on this representation, it is plain to see that for a static current vector, there are nine stable points around which the rotor could settle. It is around each of these stable points that the torque can be said to be sinusoidally distributed with respect to angular position.

Based on this, the system can be linearized around each of these points. Continuing from 5.2,

$$I\ddot{\theta} = k_T i(\phi - \theta) - T_f \mathrm{sgn}(\dot{\theta}) \tag{5.3}$$

The angle of the current vector input can then be made to be

$$k_T i\phi = T_f \mathrm{sgn}(\dot{\theta}) + k_{\mathrm{T}}\mathrm{i}\theta_{\mathrm{d}} - k_{\mathrm{d}}\dot{\theta} - k_{\mathrm{p}}(\theta - \theta_{\mathrm{d}}) - k_{\mathrm{i}}\int_0^{\mathrm{t}} (\theta - \theta_{\mathrm{d}})\,\mathrm{d}t \tag{5.4}$$

where θ_d is the desired angular position and k_p, k_i, and k_d are the associated terms for a PID controller. This setup allows the controller to be designed as a PID controller for the system while canceling out the friction force. Defining the error as

$$e = \theta - \theta_d \tag{5.5}$$

$$\dot{e} = \dot{\theta} \tag{5.6}$$

plugging 5.4 into 5.3, and canceling the appropriate terms yields

$$I\ddot{\theta} = -k_T ie - k_p e - k_d \dot{e} - k_i \int_0^t e\,\mathrm{d}t \tag{5.7}$$

Rearranging these terms and taking the Laplace transform results in

$$Is^2\Theta + k_d s\Theta + (k_T i + k_p)\Theta + \frac{k_i\Theta}{s} = (k_T i + k_p)\theta_d + k_i\theta_d \tag{5.8}$$

$$\frac{\Theta}{\theta_d} = \frac{(k_T i + k_p)s + k_i}{Is^3 + k_d s^2 + (k_T i + k_p)s + k_i} \tag{5.9}$$

In order to make this system stable, k_d must be greater than zero. This becomes readily apparent by setting k_i zero. The poles of the system can then be found to be

$$s = -k_d \pm \sqrt{k_d^2 - 4I(k_T i + k_p)} \tag{5.10}$$

Due to the hardware limitations of the system, the update rate of the system is only 9 Hz. This should not be confused with the sampling rate of 1200 Hz. This update rate is how often new input can be fed to the controller to achieve the desired position. Given that this series of test is focused on precision positioning, this should not cause a problem provided that the controller gains do not result in too large of a response too quickly and cause unstable oscillations. However, the system model does need to be converted to a discrete system. due to the slow sampling rate. Using MATLAB, the system was converted to a discrete-time transfer function defined by

$$\frac{\Theta}{\Theta_d} = \frac{0.7203z^2 - 0.614z - 0.06683}{z^3 - 1.192z^2 + 0.2314z - 0.0003867} \tag{5.11}$$

A comparison of the step response of the continuous and discretized systems can be seen in Figure 5.7 and the properties of discrete-time transfer function are given in Table 5.2. Additionally, the root locus of the system is given in Figure 5.8 where the poles were found to be 0.0017, 0.2420, and 0.9479.

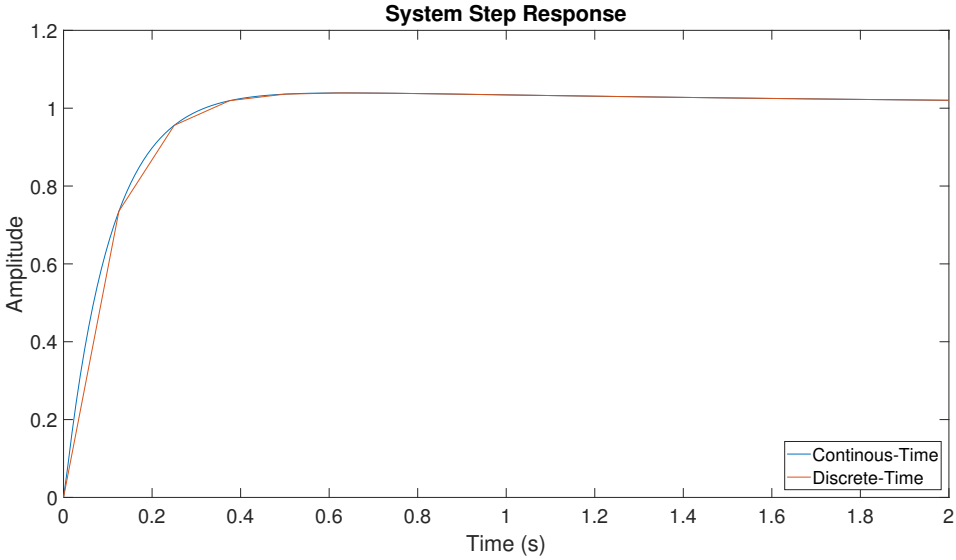


Figure 5.7: System step response for both continuous-time and discrete-time.

Table 5.2: Properties of discrete-time transfer function.

Proptery	**Value**
Kp	1
Ki	0.5
Kd	0.1
Rise Time:	0.1847
Settling Time:	0.2870
Settling Min:	0.9645
Settling Max:	1.0359
Overshoot:	1.7108
Undershoot:	0
Peak:	1.0359
Peak Time:	0.5556

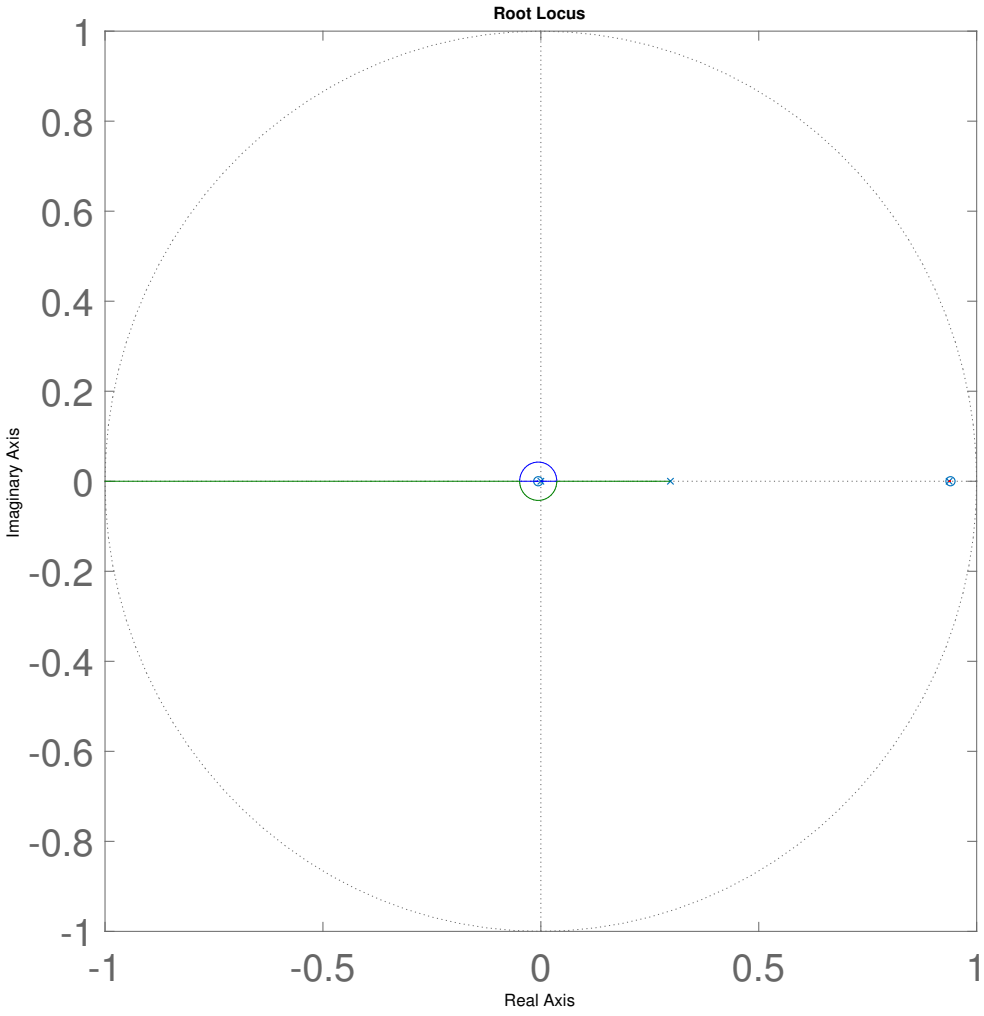


Figure 5.8: Root locus of discrete-time system.

5.4 Controller Implementation

This controller was implemented and the full system response is given in Figure 5.9, the corresponding phase currents and electrical phase angle are shown in Figure 5.10, and the convergence of the RMSE is shown in Figure 5.11. For the sake of consistency and ease of comparison, all system responses to step inputs (unless otherwise specified) will begin at 5.05 radians.

These system responses demonstrate an RMSE convergence value of about 0.016 degrees (or $2.8 \cdot 10^{-4}$ radians). Given that the outer diameter of the rotor is 0.133 meters, this rotational precision corresponds to a circumferential precision of about 19 micrometers. Despite this convergence, there remains a persistent noise in the system caused by the flipping sign of the velocity and its alternating effect on the controller. The effect of this is exacerbated by the slow update rate. The system convergence can therefore be improved by removing this term. The system's new step response results are shown in Figures 5.12–5.14. It is readily apparent from Figure 5.13 that the noise from the oscillating friction term is now gone. This corresponds with the improved RMSE performance which now converges to about 0.012 degrees (or approximately $2.1 \cdot 10^{-4}$. Given this, it doesn't seem there is any need to compensate for removing the friction term from the controller. Though these results don't take into account the error from the linearization procedure (but speak more to the noise within the system), they show that given a mapping of the Hall-effect sensor voltages to a corresponding position, the motor can consistently and accurately move to the specified location.

The response of the system to consecutive step responses within the same linearized segment is shown in Figures 5.15–5.18. The first two steps are rotations of only $5 \cdot 10^{-5}$ ° and though the data is noisy, the steps are clearly visible. This corresponds to an error of only $8.73 \cdot 10^{-7}$ radians and a circumferential error of just over 58 nm.

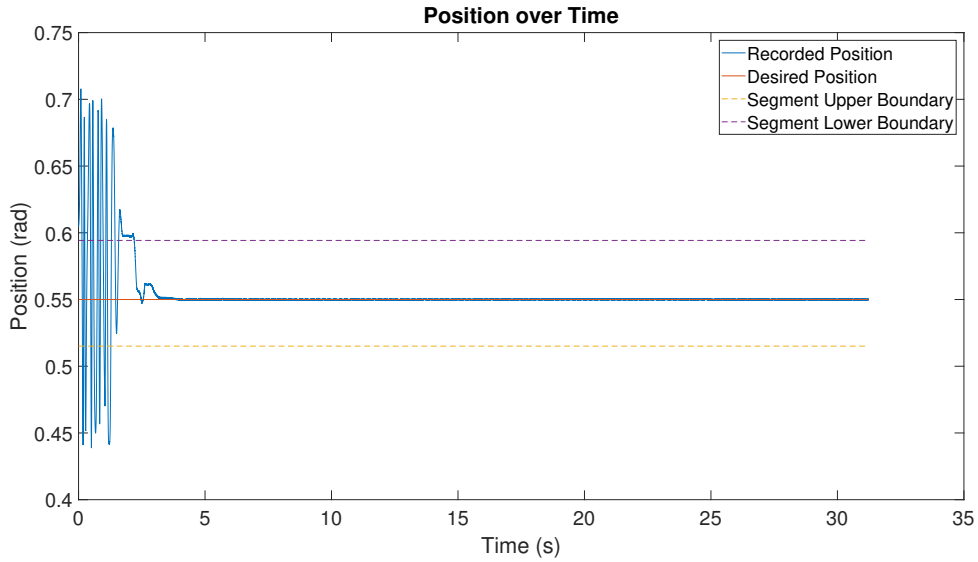


Figure 5.9: Applied controller step response.

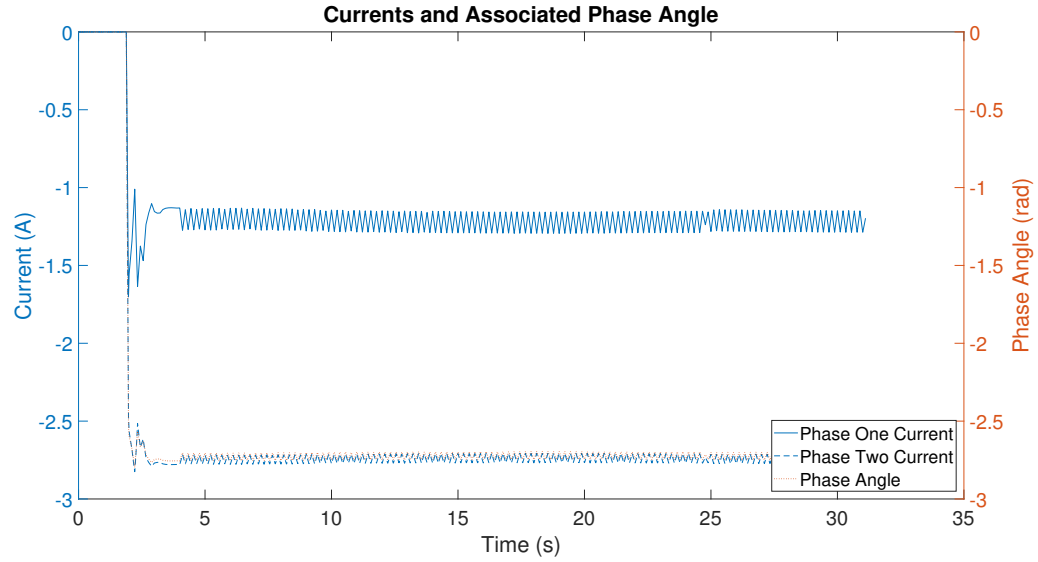


Figure 5.10: Applied controller phases and phase angle.

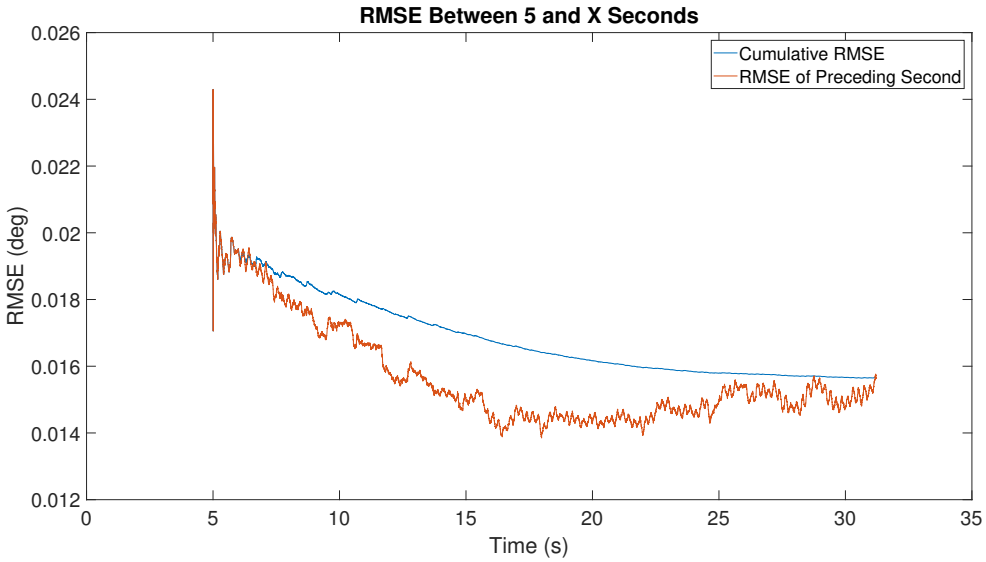


Figure 5.11: Applied controller RMSE convergence over time.

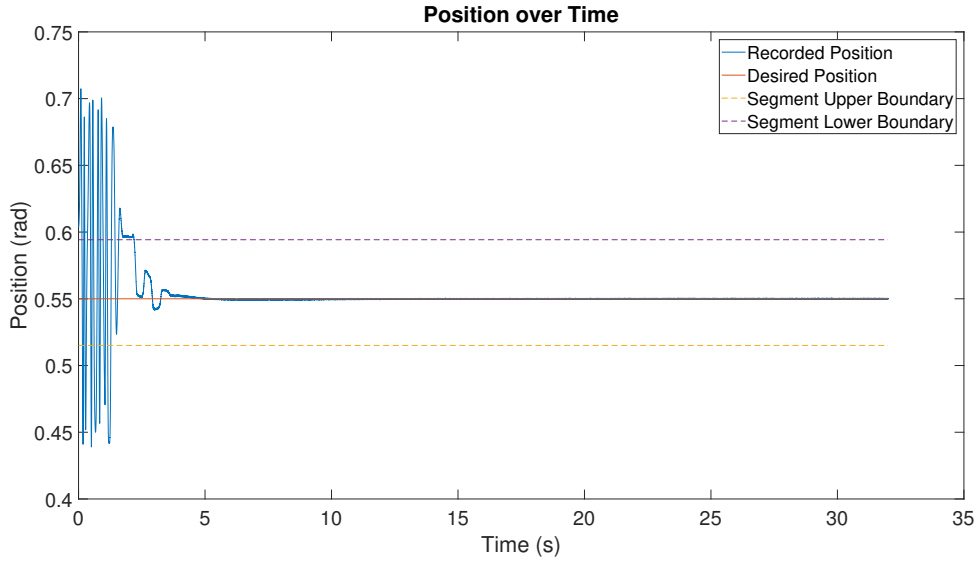


Figure 5.12: Applied controller step response.

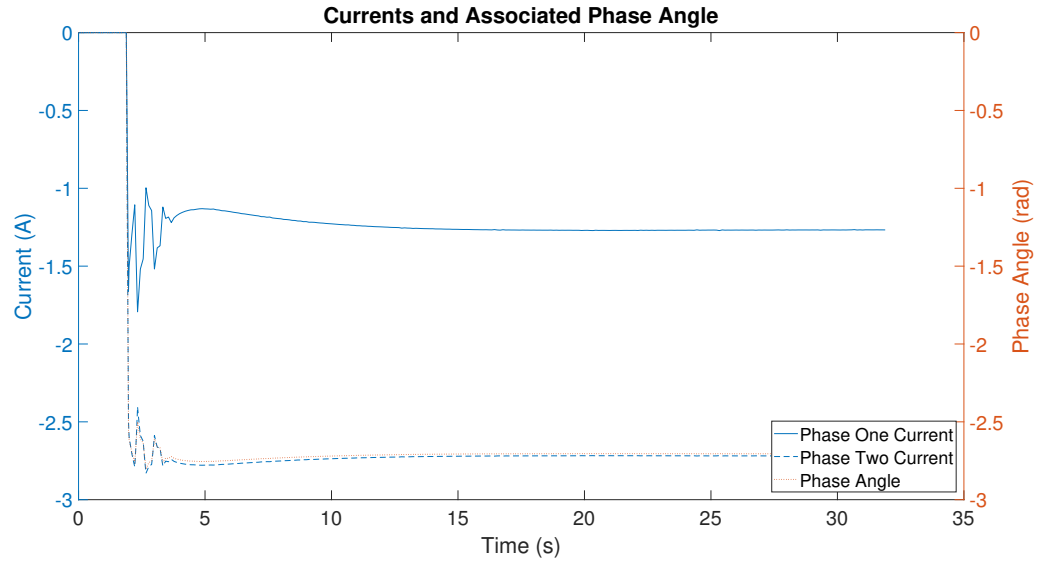


Figure 5.13: Applied controller phases and phase angle.

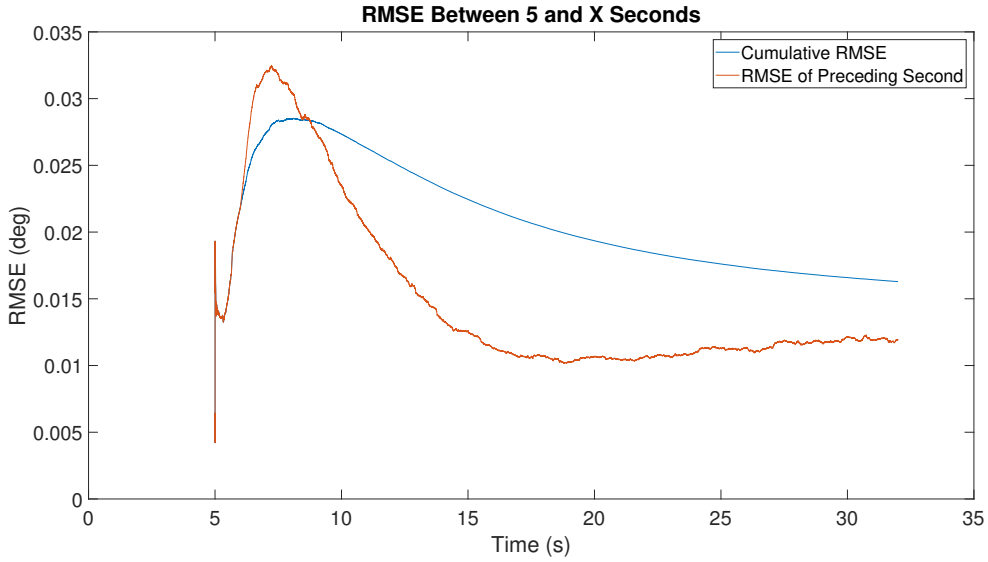


Figure 5.14: Applied controller RMSE convergence over time.

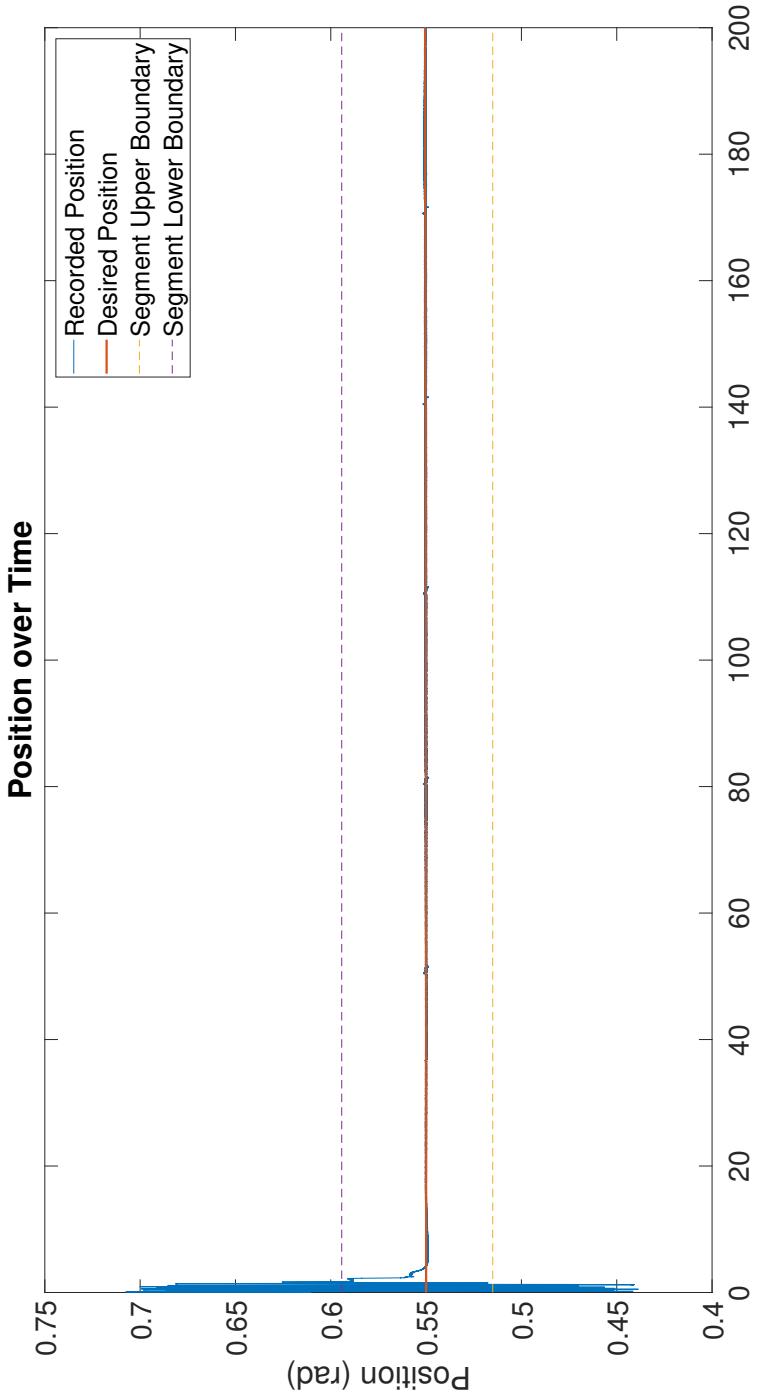


Figure 5.15: Step responses inside same linear segment of linear controller.

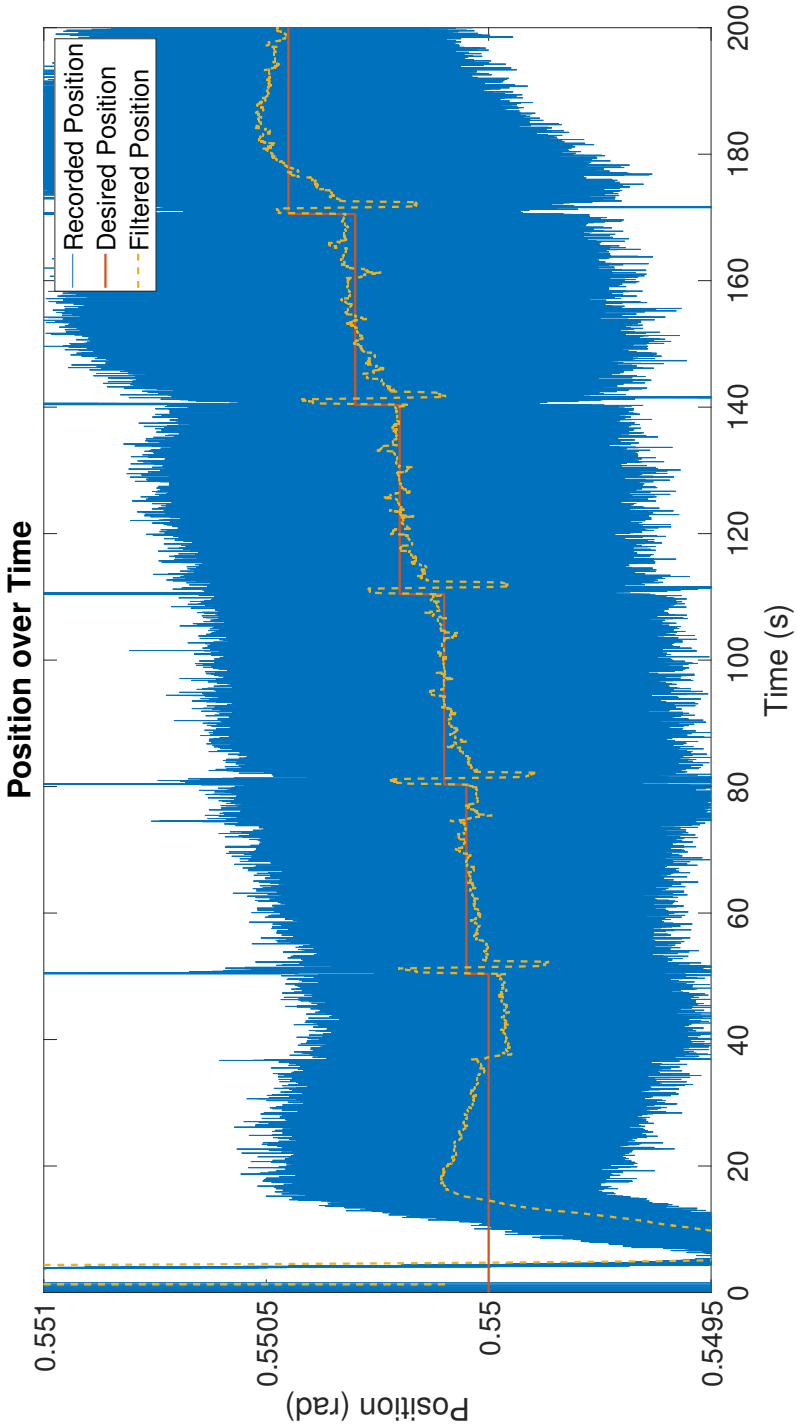


Figure 5.16: Close-up of step responses inside same linear segment of linear controller.

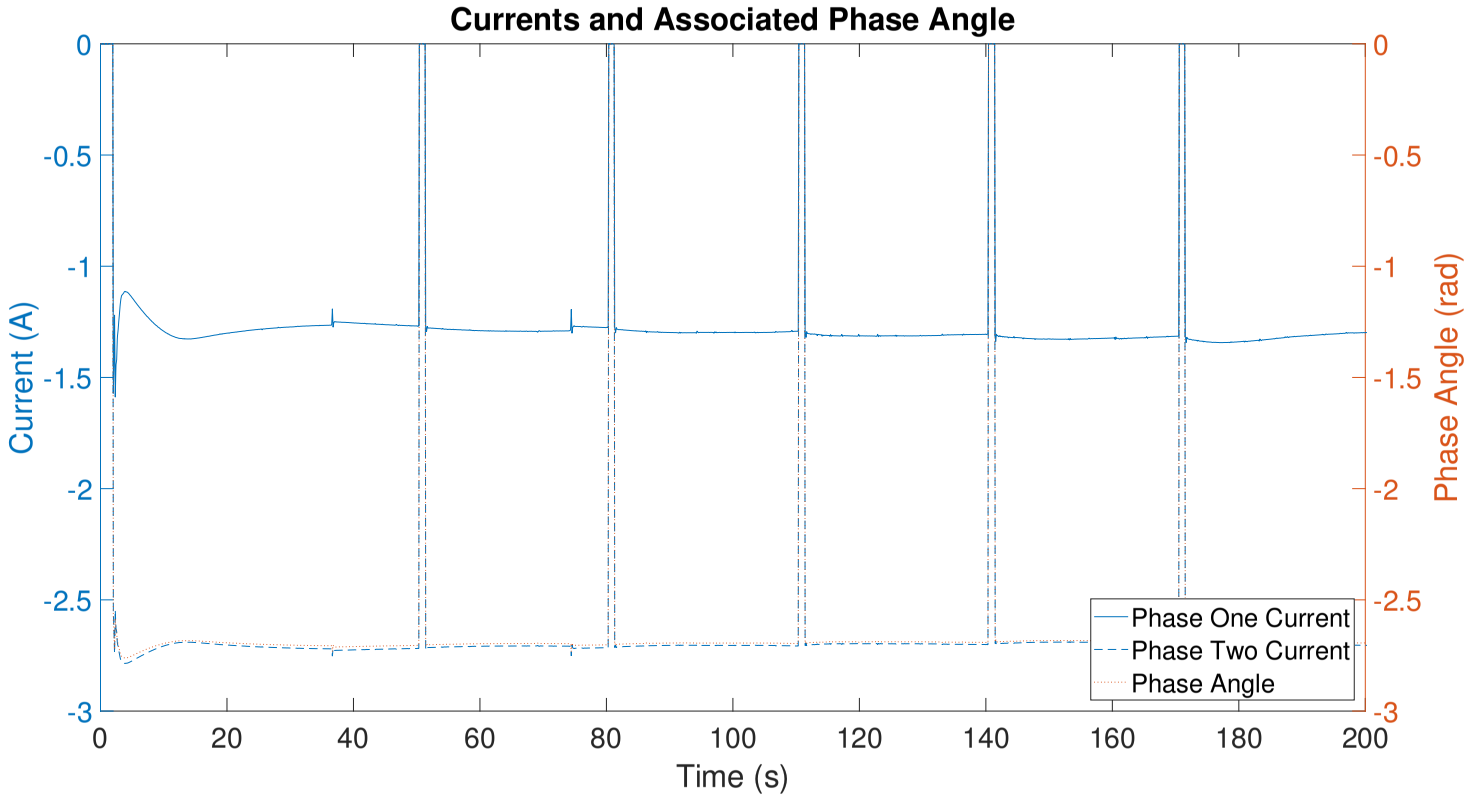


Figure 5.17: Applied controller phases and phase angle during stepped response.

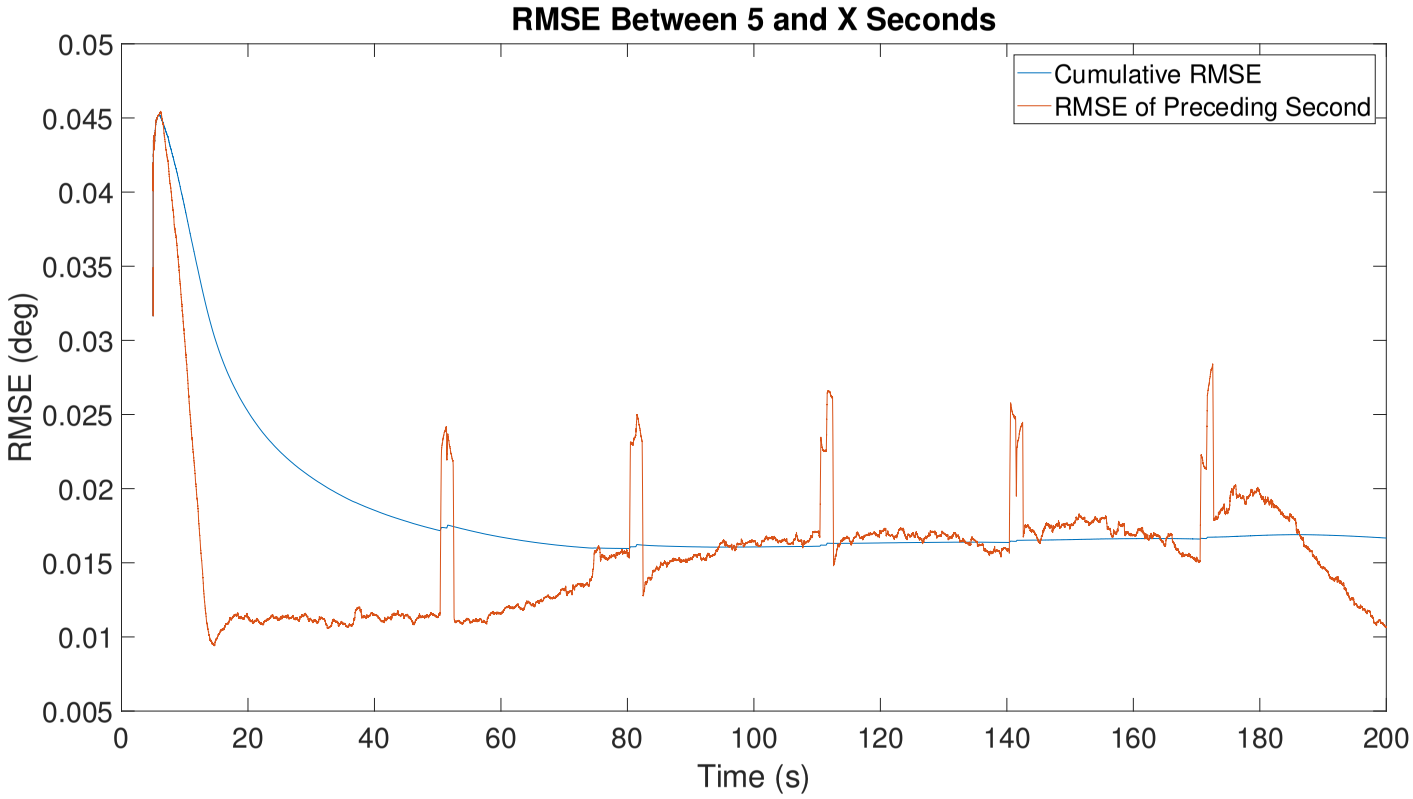


Figure 5.18: Applied controller RMSE convergence over time during step response.

6. SUMMARY AND CONCLUSIONS

In this thesis, a two-phase external-rotor PMAC motor making use of a Halbach magnet array for high-precision positioning was proposed, designed, modeled, and tested. The system was made from non-ferromagnetic materials to ensure the motor's internal field from the Halbach magnet array was as close to sinusoidal as possible. This setup was modeled and confirmed in Maxwell SV. This property was then taken advantage of to design a system that was able to generate a torque that was independent of both rotor position and current distribution due to a lack of cogging torque from winding asymmetries. The motor was then built and characterized. During the processing of data, corrections were made to account for discrepancies in the DAQ system clocks. With these corrections, the full rotation of the rotor was mapped based on data from the three embedded Hall-effect sensors. Linearized segments from this data were then generated to allow for rotor positioning based only on this data.

A position control system was then implemented using the motor characterization and linearization methods. The motor system was able to identify its absolute angular position and then respond to user input to reach desired positions. The system was then able to accurately move between step inputs of $8.73 \cdot 10^{-7}$ radians (0.00005°). This corresponds to a circumferential error of just over 58 nm.

6.1 Motor Characteristics Summary

Based on the experiments performed to characterize the system, Table 6.1 lists some of the motor's properties. The max speed was found by slowing accelerating the motor until the rotor slipped and lost synchronicity. The max torque was calculated by multiplying the average of the two torque constants by the maximum per-phase current. The friction torque corresponds to the average torque from friction during one rotation of the rotor. The

resolutions were obtained from the RMSE of the position data against the desired position.

Table 6.1: Summary of important system characteristics.

Property	**Value**	**Units**
Max Current	9	A
Max Speed	1140	rpm
Max Torque	0.554	N·m
Back-EMF Constant	0.0089	V/rpm
Average Friction Torque	0.0599	N·m
Angular Resolution	0.00005	°
Linear Resolution	58	nm

6.2 Further Study

During the position control of the system, the voltages from each Hall-effect sensor was converted to a position, and these positions were averaged with equal weighting to determine the actual position. Weighting based on the variance of each sensor could slightly improve the response of the system. In order to properly do this, a baseline variance for each Hall-effect sensor would need to be measured. The change in variance based on both the current voltage of the sensors as well as the currents in the phases would need to be taken into account to do a proper weighting. That is, not only does the currents in the phases affect the sensors' variances, but more likely than not, the variance would be affected by the current voltage level being output by the Hall-effect sensors.

Additionally, adaptive control systems could be implemented to improve the system's response to external inputs. With more powerful hardware, it would also be possible to decrease the sampling time or implement more complex or nonlinear control systems. This could particularly be used to generate more complex curves to fit the Hall-effect sensors output. In this way, the linearized segments demonstrated in this series of experiments

could be made of fitting segments of a higher order designed to further reduce the error inherent when converting the Hall-effect voltages to positions.

Finally, given a different distribution of Hall-effect sensors, the position of the rotor throughout the entire rotation could be mapped. In this way, only the linear regions of the Hall-effect sensors could be used provided that switching between Hall-effect sensors was implemented. In this way, if the voltage from each Hall-effect sensor was outside of a certain range, it would not be used to determine position. This type of positioning could help ensure the phase current was being applied at the correct angle to ensure uniform torque generation around the full revolution of the rotor.

REFERENCES

[1] D. L. Trumper, M. E. Williams, and T. H. Nguyen, "Magnet arrays for synchronous machines," in *Proceedings of Conference Record of the 1993 IEEE Industry Applications Conference Twenty-Eighth IAS Annual Meeting*, vol. 1, pp. 9–18, October 1993.

[2] "K&J Magnetics - Magnetic Field Visualization." Available: https://www.kjmagnetics.com/magfield.asp?pName=BX044-N52.

[3] J. Ofori-Tenkorrang and J. H. Lang, "A comparative analysis of torque production in halbach and conventional surface-mounted permanent-magnet synchronous motors," in *Industry Applications Conference, 1995. Thirtieth IAS Annual Meeting, IAS '95., Conference Record of the 1995 IEEE*, vol. 1, pp. 657–663, October 1995.

[4] Z. Q. Zhu, "Recent development of halbach permanent magnet machines and applications," in *2007 Power Conversion Conference - Nagoya*, pp. K–9–K–16, April 2007.

[5] J. A. Guemes, A. M. Iraolagoitia, J. I. D. Hoyo, and P. Fernandez, "Torque analysis in permanent-magnet synchronous motors: A comparative study," *IEEE Transactions on Energy Conversion*, vol. 26, no. 1, pp. 55–63, March 2011.

[6] F. Zhao, T. A. Lipo, and B. I. Kwon, "A novel two-phase permanent magnet synchronous motor modeling for torque ripple minimization," *IEEE Transactions on Magnetics*, vol. 49, no. 5, pp. 2355–2358, May 2013.

[7] V. SimÃşn-Sempere, M. Burgos-PayÃąn, and J. R. Cerquides-Bueno, "Cogging torque cancellation by magnet shaping in surface-mounted permanent-magnet motors," *IEEE Transactions on Magnetics*, vol. 53, no 7, pp. 1–7, July 2017.

[8] T. M. Jahns and W. L. Soong, "Pulsating torque minimization techniques for

permanent magnet ac motor drives-a review," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 2, pp. 321–330, April 1996.

[9] L. Wu, W. Jin, J. Ni, and J. Ying, "A cogging torque reduction method for surface mounted permanent magnet motor," in *2007 International Conference on Electrical Machines and Systems (ICEMS)*, pp. 769–773, October 2007.

[10] D. Wang, X. Wang, D. Qiao, Y. Pei, and S. Y. Jung, "Reducing cogging torque in surface-mounted permanent-magnet motors by nonuniformly distributed teeth method," *IEEE Transactions on Magnetics*, vol. 47, no. 9, pp. 2231–2239, September 2011.

[11] T. Li and G. Slemon, "Reduction of cogging torque in permanent magnet motors," *IEEE Transactions on Magnetics*, vol. 24, no. 6, pp. 2901–2903, November 1988.

[12] N. Mohan, *Electric Machines and Drives*. Hoboken: Wiley, 2012.

[13] S. Sadeghi and L. Parsa, "Improved technique for minimizing torque pulsation in halbach array permanent magnet machines," *COMPEL - The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, vol. 31, no. 6, pp. 1590–1602, 2012.

[14] R. P. Praveen, M. H. Ravichandran, V. T. S. Achari, V. P. J. Raj, G. Madhu, and G. R. Bindu, "A novel slotless pmbldc motor for precise positioning applications," in *Proceedings of 2010 International Conference on Communication Control and Computing Technologies*, pp. 250–254, October 2010.

[15] Z. Jiao, T. Wang, and L. Yan, "Design of a tubular linear oscillating motor with a novel compound halbach magnet array," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 1, pp. 498–508, February 2017.

[16] X. Hao, P. Xing, and L. Bai, "Design and analysis of moving magnet synchronous surface motor with linear halbach array," *Procedia Engineering*, vol. 16, pp. 108 – 118, 2011. International Workshop on Automobile, Power and Energy Engineering.

[17] D. L. Trumper, W.-J. Kim, and M. E. Williams, "Design and analysis framework for linear permanent-magnet machines," *IEEE Transactions on Industry Applications*, vol. 32, no. 2, pp. 371–379, March 1996.

[18] A. K. M. ParvezIqbal, F. A. M. Mokhtar, K. S. M. Sahari, and I. Aris, "A review of permanent magnet linear motor with halbach array," *Journal of Engineering and Applied Sciences*, vol. 11, pp. 1752 – 1761, January 2016.

[19] J. W. Jeon, M. Caraiani, D. H. Hwang, J. H. Lee, D. S. Kang, Y. J. Kim, and S. S. Kim, "High-precision and decomposition control of several permanent magnet linear motors for magnetic levitation," in *Proceedings of the 2006 37th IEEE Power Electronics Specialists Conference*, pp. 1–6, June 2006.

[20] G. H. Kang, Y. D. Son, G. T. Kim, and J. Hur, "A novel cogging torque reduction method for interior-type permanent-magnet motor," *IEEE Transactions on Industry Applications*, vol. 45, no. 1, pp. 161–167, January 2009.

[21] K. L. Yung, S. T. Mak, and D. K. W. Cheng, "Dual control of closed-loop stepping motor precision servo," in *Power Electronics and Drive Systems, 1999. PEDS '99. Proceedings of the IEEE 1999 International Conference on*, vol. 2, pp. 803–808, August 1999.

[22] C. C. Chan, K. T. Chau, J. Z. Jiang, W. Xia, M. Zhu, and R. Zhang, "Novel permanent magnet motor drives for electric vehicles," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 2, pp. 331–339, April 1996.

[23] A. N. Standard, "American national standard magnet wire," tech. rep., American National Standards Institute, Inc., 10 2015.

[24] The Engineering Toolbox, "Wire Gauges - Current Ratings." Available: http://www.engineeringtoolbox.com/wire-gauges-d_419.html.

[25] "K&J Magnetics - Specifications." Available: https://www.kjmagnetics.com/specs.asp.

[26] E. Gumlich and D. I. P. Goerens, “Magnetic properties of iron-carbon and iron-silicon alloys,” *Transactions of the Faraday Society*, vol. 8, pp. 98–114, October 1912.

[27] W.-J. Kim, *High-Precision Planar Magnetic Levitation*. Ph.d. thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1997.

APPENDIX A

MATLAB CODE

A.1 Load Impedance

```
rpm = linspace(0,1800,18000);
rps = rpm/60;
sps = rps*9;

Ra1 = 0.6870;
La1 = 116.1*10^-6;

Ra2 = 0.6937;
La2 = 121.6*10^-6;

XLa1 = 2*pi*sps*La1;
XLa2 = 2*pi*sps*La2;

Xa1 = sqrt(XLa1.^2 + Ra1.^2);
Xa2 = sqrt(XLa2.^2 + Ra2.^2);

plot(rpm, Xa1,'r',rpm,Xa2,'b','LineWidth',4);
xlabel('Motor Speed (RPM)');
ylabel('Phase Impedance (\Omega)');
title('Phase Impedance vs Motor Speed');
legend('Phase 1','Phase 2');
% ylim([0 1])
```

A.2 Gain Calibration

A.2.1 Gain Calibration Collection

```
d = daq.getDevices;
s = daq.createSession('ni');
Fs = 32000;
runTime = 5; %Actual run time is double this
Len = runTime*Fs;
T = 1/Fs;
t = linspace(0,runTime,Len);
windowSize = 100;
settleTime = 1;

R1 = 0.6870; % ohms
R2 = 0.6937; % ohms
L1 = 116.1*10^-6; % henries
L2 = 121.6*10^-6; % henries


s.Rate = Fs;

aOChannel0 = 'ao0';      %Pin 22
aOChannel1 = 'ao1';      %Pin 21
aI0Plus = 'ai15';
aI0Min = 'ai13';

aI1Plus = 'ai12';
aI1Min = 'ai10';

aBoardOutput0 = 'ai6';
aBoardOutput1 = 'ai8';
% aBoardOutputGND = 'ai8';

% Channels Not Working 28,30,33,57,60,65,68,
% Channels Working 23,25,26,31,34,58,61,63,66,


addAnalogOutputChannel(s,'Dev1',aOChannel0,'voltage');
```

```
addAnalogOutputChannel(s,'Dev1',aOChannel1,'voltage');
ch15 = addAnalogInputChannel(s,'Dev1', aI0Plus, '
   Voltage');
ch13 = addAnalogInputChannel(s,'Dev1', aI0Min, 'Voltage
   ');
ch12 = addAnalogInputChannel(s,'Dev1', aI1Plus, '
   Voltage');
ch10 = addAnalogInputChannel(s,'Dev1', aI1Min, 'Voltage
   ');
ch6 = addAnalogInputChannel(s,'Dev1', aBoardOutput0, '
   Voltage');
% ch8 = addAnalogInputChannel(s,'Dev1', aBoardOutputGND
   , 'Voltage');
ch8 = addAnalogInputChannel(s,'Dev1', aBoardOutput1, '
   Voltage');
% legend('AI0+','AI0-','AI1+','AI1-','ABoard0+','
   ABoard1+');

ch15.TerminalConfig = 'SingleEnded';
ch13.TerminalConfig = 'SingleEnded';
ch12.TerminalConfig = 'SingleEnded';
ch10.TerminalConfig = 'SingleEnded';
ch6.TerminalConfig = 'SingleEnded';
ch8.TerminalConfig = 'SingleEnded';
% ch9.TerminalConfig = 'SingleEnded';

out0 = zeros(Len,2);
out1 = zeros(Len,2);
sig = [];

% out0(:,1) = transpose([linspace(-3,-0.5,L/2),
   linspace(0.5,3,L/2)]);
% out1(:,2) = transpose([linspace(-3,-0.5,L/2),
   linspace(0.5,3,L/2)]);

out0(:,1) = [linspace(-4,-4,Len/8), linspace(-3,-3,Len
   /8),...
    linspace(-2,-2,Len/8), linspace(-1,-1,Len/8) ...
```

```
63      linspace(1,1,Len/8), linspace(2,2,Len/8),...
64      linspace(3,3,Len/8), linspace(4,4,Len/8)]';
65  out1(:,2) = [linspace(-4,-4,Len/8), linspace(-3,-3,Len
        /8),...
66      linspace(-2,-2,Len/8), linspace(-1,-1,Len/8) ...
67      linspace(1,1,Len/8), linspace(2,2,Len/8),...
68      linspace(3,3,Len/8), linspace(4,4,Len/8)]';
69
70  queueOutputData(s, [-4*ones(settleTime*Fs,1) zeros(
        settleTime*Fs,1)]);
71  [dontCare,dontCare] = s.startForeground;
72
73  queueOutputData(s, out0);
74  [data1,time1] = s.startForeground;
75  queueOutputData(s,[0 0]);
76  [dontCare,dontCare] = s.startForeground;
77
78  queueOutputData(s, [zeros(settleTime*Fs,1) -4*ones(
        settleTime*Fs,1)]);
79  [dontCare,dontCare] = s.startForeground;
80
81  queueOutputData(s, out1);
82  [data2,time2] = s.startForeground;
83  queueOutputData(s,[0 0]);
84  [dontCare,dontCare] = s.startForeground;
85
86  preData1 = data1;
87  preData2 = data2;
88
89  aO1 = (data1(:,1)-data1(:,2))./R1;
90  aO2 = (data2(:,3)-data2(:,4))./R2;
91  aO1Board = data1(:,5);
92  aO2Board = data2(:,6);
93
94  preGain1 = aO1(:,1)./out0(:,1);
95  preGain2 = aO2(:,1)./out1(:,2);
96
97  b = (1/windowSize)*ones(1,windowSize);
```

```
a = 1;
sig1f = filter(b,a,aO1);
sig2f = filter(b,a,aO2);

% m1 = (L*sum(out0(:,1).*aO1) - sum(out0(:,1))*sum(aO1)
   )/...
% (L*sum(out0(:,1).^2) - sum(out0(:,1))^2)

m1 = mean(preGain1);
% b1 = sum(aO1-m1*out0(:,1))/L;
b1 = (sum(aO1)-m1*sum(out0(:,1)))/Len;
m2 = mean(preGain2);
b2 = (sum(aO2)-m2*sum(out1(:,1)))/Len;

% plot(time,data,time,aO1,time,aO2);
% legend('Volt1+','Volt1-','Volt2+','Volt2-','Volt1
   +-','Volt2+-');
figure(1)
plot(time1, aO1Board, time1, aO1, time1, preGain1,time1
   ,sig1f);
legend('Voltage Input','Current Measured','Gain','
   Filtered Current','Location','southeast');
xlabel('Time (secs)');
ylabel('Voltage (V) and Current (A)');
title('Phase 1 Pre-Calibration');
figure(2)
plot(time2, out1(:,2), time2, aO2, time2, preGain2,
   time2,sig2f);
legend('Voltage Input','Current Measured','Gain','
   Filtered Current','Location','southeast');
xlabel('Time (secs)');
ylabel('Voltage (V) and Current (A)');
title('Phase 2 Pre-Calibration');

% figure(3)
% Y = fft(data(:,4));
% P2 = abs(Y/L);
% P1 = P2(1:L/2+1);
```

```
% P1(2:end-1) = 2*P1(2:end-1);
% f = Fs*(0:(L/2))/L;
% figure(3)
% plot(f,P1);

%% Validation

y0 = (out0-b1)./m1;
y1 = (out1-b2)./m2;

queueOutputData(s, [-4*ones(settleTime*Fs,1) zeros(
   settleTime*Fs,1)]);
[dontCare,dontCare] = s.startForeground;

queueOutputData(s, y0);
[data1,time1] = s.startForeground;
queueOutputData(s,[0 0]);
[dontCare,dontCare] = s.startForeground;

queueOutputData(s, [zeros(settleTime*Fs,1) -4*ones(
   settleTime*Fs,1)]);
[dontCare,dontCare] = s.startForeground;

queueOutputData(s, y1);
[data2,time2] = s.startForeground;
queueOutputData(s,[0 0]);
[dontCare,dontCare] = s.startForeground;

postData1 = data1;
postData2 = data2;

aO1 = (data1(:,1)-data1(:,2))./R1;
aO2 = (data2(:,3)-data2(:,4))./R2;
aO1Board = data1(:,5);
aO2Board = data2(:,6);

postGain1 = aO1(:,1)./out0(:,1);
postGain2 = aO2(:,1)./out1(:,2);
```

```
166
167 b = (1/windowSize)*ones(1,windowSize);
168 a = 1;
169 sig1f = filter(b,a,aO1);
170 sig2f = filter(b,a,aO2);
171
172 figure(3)
173 plot(time1, out0(:,1), time1, aO1, time1, postGain1,
       time1,sig1f);
174 legend('Voltage Input','Current Measured','Gain','
       Filtered Current','Location','southeast');
175 xlabel('Time (secs)');
176 ylabel('Voltage (V) and Current (A)');
177 title('Phase 1 Post-Calibration');
178 figure(4)
179 plot(time2, out1(:,2), time2, aO2, time2, postGain2,
       time2,sig2f);
180 legend('Voltage Input','Current Measured','Gain','
       Filtered Current','Location','southeast');
181 xlabel('Time (secs)');
182 ylabel('Voltage (V) and Current (A)');
183 title('Phase 2 Post-Calibration');
184
185 % save('steppedCalibration.mat');
```

A.2.2 Gain Calibration Data Plotting

```
1  load('linearCalibration.mat');
2  % load('steppedCalibration.mat');
3
4  aO1 = preData1(:,1)-preData1(:,2);
5  aO2 = preData2(:,3)-preData2(:,4);
6  aO1Board = preData1(:,5);
7  aO2Board = preData2(:,6);
8
9  gain1 = aO1(:,1)./out0(:,1);
10 gain2 = aO2(:,1)./out1(:,2);
11
12 b = (1/windowSize)*ones(1,windowSize);
```

```
a = 1;
sig1f = filter(b,a,aO1);
sig2f = filter(b,a,aO2);
gain1f = filter(b,a,gain1);
gain2f = filter(b,a,gain2);

% m1 = (L*sum(out0(:,1).*aO1) - sum(out0(:,1))*sum(aO1))/...
% (L*sum(out0(:,1).^2) - sum(out0(:,1))^2)

m1 = mean(gain1);
% b1 = sum(aO1-m1*out0(:,1))/L;
b1 = (sum(aO1)-m1*sum(out0(:,1)))/L;
m2 = mean(gain2);
b2 = (sum(aO2)-m2*sum(out1(:,1)))/L;

figure(1)
plot(time1, out0(:,1), time1, aO1Board, time1, aO1, time1,sig1f, time1, gain1f);
legend('Desired 1','Board Output 1','Amp Output 1','Amp Output Filtered 1','Gain 1 Filtered','Location','southeast');
xlabel('Time (secs)');
ylabel('Voltage');
title('Phase 1 Pre-Calibration');
figure(2)
plot(time2, out1(:,2), time2, aO2Board, time2, aO2, time2,sig2f, time2, gain2f);
legend('Desired 2','Board Output 2','Amp Output 2','Amp Output Filtered 2','Gain 2 Filtered','Location','southeast');
xlabel('Time (secs)');
ylabel('Voltage');
title('Phase 2 Pre-Calibration');

aO1 = postData1(:,1)-postData1(:,2);
aO2 = postData2(:,3)-postData2(:,4);
aO1Board = postData1(:,5);
```

```
aO2Board = postData1(:,6);

gain1 = aO1(:,1)./out0(:,1);
gain2 = aO2(:,1)./out1(:,2);

b = (1/windowSize)*ones(1,windowSize);
a = 1;
sig1f = filter(b,a,aO1);
sig2f = filter(b,a,aO2);
gain1f = filter(b,a,gain1);
gain2f = filter(b,a,gain2);

figure(3)
plot(time1, out0(:,1), time1, aO1Board, time1, aO1,
   time1, sig1f, time1, gain1f);
legend('Desired 1','Board Output 1','Amp Output 1','Amp
    Output Filtered 1','Gain 1 Filtered','Location','
   southeast');
xlabel('Time (secs)');
ylabel('Voltage');
title('Phase 1 Post-Calibration');
figure(4)
plot(time2, out1(:,2), time2, aO2Board, time2, aO2,
   time2, sig2f, time2, gain2f);
legend('Desired 2','Board Output 2','Amp Output 2','Amp
    Output Filtered 2','Gain 2 Filtered','Location','
   southeast');
xlabel('Time (secs)');
ylabel('Voltage');
title('Phase 2 Post-Calibration');

figure(5);
plot(time1,preData1);
xlabel('Time (secs)');
ylabel('Voltage');
legend('AI0+','AI0-','AI1+','AI1-','ABoard0+','ABoard1+
   ');
title('Phase 1 Pre-Calibration Raw Voltages');
```

```
figure(6);
plot(time2,preData2);
xlabel('Time (secs)');
ylabel('Voltage');
legend('AI0+','AI0-','AI1+','AI1-','ABoard0+','ABoard1+
   ');
title('Phase 2 Pre-Calibration Raw Voltages');
figure(7);
plot(time1,postData1);
xlabel('Time (secs)');
ylabel('Voltage');
legend('AI0+','AI0-','AI1+','AI1-','ABoard0+','ABoard1+
   ');
title('Phase 1 Post-Calibration Raw Voltages');
figure(8);
plot(time2,postData2);
xlabel('Time (secs)');
ylabel('Voltage');
legend('AI0+','AI0-','AI1+','AI1-','ABoard0+','ABoard1+
   ');
title('Phase 2 Post-Calibration Raw Voltages');
```

A.3 Motor Friction Force

A.3.1 Motor Deceleration Processing from Video

```
close all;
load('videoPoints.mat');
time = videoPoints(:,1);
pos = videoPoints(:,2);

for i = 2:length(pos)
    if (pos(i) < pos(i-1))
        pos(i:end) = pos(i:end) + 360;
    end
end

posCorrected = pos - pos(1);
[fitOb gof] = fit(time,posCorrected,'poly2');
coeffs = coeffvalues(fitOb);
a = coeffs(1);
b = coeffs(2);
c = coeffs(3);

xs = linspace(0,time(end),1000);
ys = a*xs.^2 + b*xs + c;

figure(1);
hold on;
plot(xs,ys,':','LineWidth',5);
plot(time,posCorrected,'.','MarkerSize',24);
title('Motor Position During Deceleration');
xlabel('Time (sec)');
ylabel('Position (degree)');
legend('Best Fit Line','Raw Data','Location','SouthEast
   ');
% grid minor
hold off;
```

A.3.2 Motor Deceleration Processing from Sensor

```
load('friction_Meas.mat');
close all

segs = 18;
I = 0.001414;

testResult{2,end}{end} = testResult{2,end}{end}(end);
propsLeg = testResult{1,:};
props = cell2mat(testResult{2,:});

%Assign properties and basic vectors
sr = props(1); % samping rate
timeVector = slowDownTime;
inputSig0 = slowDown(:,5) - (max(slowDown(:,5)) + min(
   slowDown(:,5)))/2;
inputSign0 = sign(inputSig0);
inputSign0Diff = [0; diff(inputSign0)];
crossoverIndices0 = find(inputSign0Diff);
numCrosses = length(crossoverIndices0);
crossoverIndicesDiff0 = diff([1; crossoverIndices0]);
crossoverTimes0 = timeVector(crossoverIndices0);
crossoverTimeDiff0 = diff([0 ;crossoverTimes0]); % sec
   (difference)
crossoverVel0 = (1./crossoverTimeDiff0)./2; % Hz
motorVelAtCross0 = crossoverVel0./9; % Hz
slowingPos = [360/segs*linspace(0,numCrosses,numCrosses
   +1)]';

weightVec = linspace(1,numCrosses+1,numCrosses+1);
[fitOb gof] = fit([0; crossoverTimes0],slowingPos,'
   poly2');%,'Weights',weightVec);
coeffs = coeffvalues(fitOb);
a = coeffs(1);
b = coeffs(2);
c = coeffs(3);
xs = linspace(0,crossoverTimes0(end),1000);
ys = a*xs.^2 + b*xs + c;

```

```
[fitOb2 gof2] = fit(crossoverTimes0,360*
   motorVelAtCross0,'poly2');%,'Weights',weightVec);
coeffs2 = coeffvalues(fitOb2);
a2 = coeffs2(1);
b2 = coeffs2(2);
c2 = coeffs2(3);
xs2 = linspace(0,crossoverTimes0(end),1000);
ys2 = a2*xs2.^2 + b2*xs2 + c2;

b3 = 0.002;
c3 = -log(c2)*I;
xs3 = xs2;
ys3 = exp(-(b3*xs3 + c3)./I);

posFromVel = cumsum(motorVelAtCross0.*
   crossoverTimeDiff0)*360;

figure(1);
plot(xs2,ys2,'-.','LineWidth',4);
hold on;
plot(crossoverTimes0,360*motorVelAtCross0,'.','
   MarkerSize',14);
plot(xs3,ys3,':','LineWidth',4);
hold off;
ylabel('Motor Velocity (Degrees)');
xlabel('Time (sec)');
title('Motor Slow Down Over Time');
legend('Best Fit Line','Raw Data','Damping','Location',
   'NorthEast');


figure(2);
plot(xs,ys,'-.','LineWidth',4);
hold on;
plot([0; crossoverTimes0],slowingPos,'.','MarkerSize'
   ,14);
% plot(crossoverTimes0,posFromVel);
hold off;
```

```
68 ylabel('Motor Position (Degrees)');
69 xlabel('Time (sec)');
70 title('Motor Slow Down Over Time');
71 legend('Best Fit Line','Raw Data','Location','SouthEast
      ');
```

A.4 Torque Characteristics

A.4.1 Torque Measurement

```
d = daq.getDevices;
s = daq.createSession('ni');
Fs = 8000;
numberOfScans = Fs*2;
maxI = 3;

s.Rate = Fs;

aOChannel0 = 'ao0';      %Pin 22
aOChannel1 = 'ao1';      %Pin 21
aI0Plus = 'ai15';
aI0Min = 'ai13';

aI1Plus = 'ai12';
aI1Min = 'ai10';

aBoardOutput0 = 'ai6';
aBoardOutput1 = 'ai8';
% aBoardOutputGND = 'ai8';

% Channels Not Working 28,30,33,57,60,65,68,
% Channels Working 23,25,26,31,34,58,61,63,66,


addAnalogOutputChannel(s,'Dev1',aOChannel0,'voltage');
addAnalogOutputChannel(s,'Dev1',aOChannel1,'voltage');
ch15 = addAnalogInputChannel(s,'Dev1', aI0Plus, '
   Voltage');
ch13 = addAnalogInputChannel(s,'Dev1', aI0Min, 'Voltage
   ');
ch12 = addAnalogInputChannel(s,'Dev1', aI1Plus, '
   Voltage');
ch10 = addAnalogInputChannel(s,'Dev1', aI1Min, 'Voltage
   ');
```

```
ch6 = addAnalogInputChannel(s,'Dev1', aBoardOutput0, '
   Voltage');
% ch8 = addAnalogInputChannel(s,'Dev1', aBoardOutputGND
   , 'Voltage');
ch8 = addAnalogInputChannel(s,'Dev1', aBoardOutput1, '
   Voltage');
% legend('AI0+','AI0-','AI1+','AI1-','ABoard0+','
   ABoard1+');

ch15.TerminalConfig = 'SingleEnded';
ch13.TerminalConfig = 'SingleEnded';
ch12.TerminalConfig = 'SingleEnded';
ch10.TerminalConfig = 'SingleEnded';
ch6.TerminalConfig = 'SingleEnded';
ch8.TerminalConfig = 'SingleEnded';
% ch9.TerminalConfig = 'SingleEnded';

prompt = ['Please input desired current in amps: '];
allData = [];
totTime = 0;
totTimeVec = [];
allDataTotTime = [];

while (true)
    desCurrent = input(prompt);

    if (desCurrent == 0)
       break;
    end

    if (desCurrent > maxI)
       desCurrent = maxI;
       disp(['Max current is ' num2str(maxI) ' amps.'])
          ;
    end

    if (desCurrent < -maxI)
       desCurrent = -maxI;
```

```
64          disp(['Max negative current is -' num2str(maxI)
               ' amps.']);
65      end
66
67      phase1Out = desCurrent*ones(numberOfScans,1);
68      phase2Out = 0*ones(numberOfScans,1);
69
70      queueOutputData(s,[phase1Out phase2Out]);
71      [captured_data,time] = s.startForeground();
72      beep();
73      totTimeVec = [totTimeVec; time+totTime];
74
75      allData = [allData; time,captured_data];
76      allDataTotTime = [allDataTotTime; time+totTime,
           captured_data];
77
78      phase1Volt = allDataTotTime(:,2) - allDataTotTime
           (:,3);
79      plot(totTimeVec,phase1Volt,totTimeVec,
           allDataTotTime(:,6));
80
81      totTime = totTime + time(end);
82
83  end
84
85  queueOutputData(s,[0 0]);
86  s.startForeground();
```

A.4.2 Torque Data Plotting

```
1  load('torque_measurements.mat');
2
3  d = 13.3096; % cm
4  r = d/2/100; % m
5  massKg = massMeasured/1000; % kg
6  g = 9.81; % m/s^2
7  weightTorque = massKg*g*r;
8
9  [fitOb1,gof1] = fit(currentReq1,weightTorque,'poly1');
```

```
[fitOb2,gof2] = fit(currentReq2,weightTorque,'poly1');
coeffs1 = coeffvalues(fitOb1);
coeffs2 = coeffvalues(fitOb2);
coeffBoth(1) = (coeffs1(1) + coeffs2(1))/2*sqrt(2);
coeffBoth(2) = (coeffs1(2) + coeffs2(2))/2;

X1(1) = min(currentReq1);
X1(2) = max(currentReq1);
X2(1) = min(currentReq2);
X2(2) = max(currentReq2);

Y1 = coeffs1(1).*X1 + coeffs1(2);
Y2 = coeffs2(1).*X2 + coeffs2(2);

figure(1);
plot(currentReq1,weightTorque,'.','MarkerSize',24);
hold on;
plot(currentReq2,weightTorque,'.','MarkerSize',24);
plot(X1,Y1,':','LineWidth',3.5);
plot(X2,Y2,':','LineWidth',3.5);
hold off;

xlabel('Current (A)');
ylabel('Torque (N*m)');
title('Torque vs Current');
legend('Phase 1 Data','Phase 2 Data','Phase 1 Fit Line'
   ,'Phase 2 Fit Line','Location','NorthWest');
```

A.5 Motor Current/Hall-Effect Sensor Interaction

A.5.1 Sensor Current Relation Data Collection

```
settleTime = 1;
runTime = 2;
sampleRate = 8000;
maxCurrent = 3;

aOChannel0 = 'ao0';      %Pin 22
aOChannel1 = 'ao1';      %Pin 21

aI0Plus = 'ai15';
aI0Min = 'ai13';
aI1Plus = 'ai12';
aI1Min = 'ai10';

aIH07 = 'ai9';          %Pin
aIH12 = 'ai11';           %Pin
aIH02 = 'ai14';          %Pin

d = daq.getDevices;
s = daq.createSession('ni');
s.Rate = sampleRate;
s.DurationInSeconds = runTime;

addAnalogOutputChannel(s,'Dev1',aOChannel0,'voltage');
addAnalogOutputChannel(s,'Dev1',aOChannel1,'voltage');
addAnalogInputChannel(s,'Dev1', aI0Plus, 'Voltage');
addAnalogInputChannel(s,'Dev1', aI0Min, 'Voltage');
addAnalogInputChannel(s,'Dev1', aI1Plus, 'Voltage');
addAnalogInputChannel(s,'Dev1', aI1Min, 'Voltage');
addAnalogInputChannel(s,'Dev1', aIH07, 'Voltage'); %Pin
    26
addAnalogInputChannel(s,'Dev1', aIH12, 'Voltage'); %Pin
    58
addAnalogInputChannel(s,'Dev1', aIH02, 'Voltage'); %Pin
    23

```

```
ch15.TerminalConfig = 'SingleEnded';
ch13.TerminalConfig = 'SingleEnded';
ch12.TerminalConfig = 'SingleEnded';
ch10.TerminalConfig = 'SingleEnded';
ch9.TerminalConfig = 'SingleEnded';
ch11.TerminalConfig = 'SingleEnded';
ch14.TerminalConfig = 'SingleEnded';

tVecSettle = sampleRate*settleTime;
tVec = sampleRate*runTime;
totPhaseTime = settleTime*2 + runTime*2;

outSettle = [maxCurrent*ones(tVecSettle/2,1); zeros(
   tVecSettle/2,1)];
outLine = [linspace(0,maxCurrent,tVec/2), linspace(
   maxCurrent,0,tVec/2)]';;
outZeroSettle = zeros(tVecSettle,1);
outZero = zeros(tVec,1);

% Phase 1
queueOutputData(s,[outSettle outZeroSettle]);
[settlePhase1Pos,settleTimePhase1Pos] = s.
   startForeground;

queueOutputData(s,[outLine outZero]);
[phase1Pos,timePhase1Pos] = s.startForeground;

queueOutputData(s,[-outSettle outZeroSettle]);
[settlePhase1Neg,settleTimePhase1Neg] = s.
   startForeground;

queueOutputData(s,[-outLine outZero]);
[phase1Neg,timePhase1Neg] = s.startForeground;

% Phase 2
queueOutputData(s,[outZeroSettle outSettle]);
[settlePhase2Pos,settleTimePhase2Pos] = s.
   startForeground;
```

```
66
67 queueOutputData(s,[outZero outLine]);
68 [phase2Pos,timePhase2Pos] = s.startForeground;
69
70 queueOutputData(s,[outZeroSettle -outSettle]);
71 [settlePhase2Neg,settleTimePhase2Neg] = s.
      startForeground;
72
73 queueOutputData(s,[outZero -outLine]);
74 [phase2Neg,timePhase2Neg] = s.startForeground;
75
76 % save('sensor_current_relation.mat');
```

A.5.2 Sensor Current Relation Data Plotting

```
1  load('sensor_current_relation.mat');
2  close all;
3
4  timeVec = [linspace(0,totPhaseTime,totPhaseTime*
      sampleRate)]';
5  phaseCurrents = [outSettle; outLine; -outSettle; -
      outLine];
6  halbachData1 = [settlePhase1Pos; phase1Pos;
      settlePhase1Neg; phase1Neg];
7  halbachData2 = [settlePhase2Pos; phase2Pos;
      settlePhase2Neg; phase2Neg];
8
9  tSettled = settleTime*sampleRate/2;
10 phase1PosRefValue = mean(settlePhase1Pos(tSettled:end
      ,:));
11 phase1NegRefValue = mean(settlePhase1Neg(tSettled:end
      ,:));
12 phase2PosRefValue = mean(settlePhase2Pos(tSettled:end
      ,:));
13 phase2NegRefValue = mean(settlePhase2Neg(tSettled:end
      ,:));
14
15 phase1PosMeanVec = ones(runTime*sampleRate,1)*
      phase1PosRefValue(5:7);
```

```
phase1NegMeanVec = ones(runTime*sampleRate,1)*
   phase1NegRefValue(5:7);
phase2PosMeanVec = ones(runTime*sampleRate,1)*
   phase2PosRefValue(5:7);
phase2NegMeanVec = ones(runTime*sampleRate,1)*
   phase2NegRefValue(5:7);

currentAfterSettle = [outLine; -outLine];
phase1HalbachAdj = [phase1Pos(:,5:7); phase1Neg(:,5:7)]
    - [phase1PosMeanVec; phase1NegMeanVec];
phase2HalbachAdj = [phase2Pos(:,5:7); phase2Neg(:,5:7)]
    - [phase2PosMeanVec; phase2NegMeanVec];

[fitPhase1H07 gofP1H07] = fit(currentAfterSettle,
   phase1HalbachAdj(:,1),'poly1');
coeffsP1H07(1,1:2) = coeffvalues(fitPhase1H07);
[fitPhase1H12 gofP1H12] = fit(currentAfterSettle,
   phase1HalbachAdj(:,2),'poly1');
coeffsP1H12(1,1:2) = coeffvalues(fitPhase1H12);
[fitPhase1H02 gofP1H02] = fit(currentAfterSettle,
   phase1HalbachAdj(:,3),'poly1');
coeffsP1H02(1,1:2) = coeffvalues(fitPhase1H02);

[fitPhase2H07 gofP2H07] = fit(currentAfterSettle,
   phase2HalbachAdj(:,1),'poly1');
coeffsP2H07(1,1:2) = coeffvalues(fitPhase2H07);
[fitPhase2H12 gofP2H12] = fit(currentAfterSettle,
   phase2HalbachAdj(:,2),'poly1');
coeffsP2H12(1,1:2) = coeffvalues(fitPhase2H12);
[fitPhase2H02 gofP2H02] = fit(currentAfterSettle,
   phase2HalbachAdj(:,3),'poly1');
coeffsP2H02(1,1:2) = coeffvalues(fitPhase2H02);

fitVec = linspace(-maxCurrent,maxCurrent,20);
P1H07fitLine = fitVec*coeffsP1H07(1) + coeffsP1H07(2);
P1H12fitLine = fitVec*coeffsP1H12(1) + coeffsP1H12(2);
P1H02fitLine = fitVec*coeffsP1H02(1) + coeffsP1H02(2);
P2H07fitLine = fitVec*coeffsP2H07(1) + coeffsP2H07(2);
```

```
P2H12fitLine = fitVec*coeffsP2H12(1) + coeffsP2H12(2);
P2H02fitLine = fitVec*coeffsP2H02(1) + coeffsP2H02(2);

figure(1);
yyaxis left
plot(timeVec,phaseCurrents,'--');
ylabel('Current (A)');
hold on;
yyaxis right
plot(timeVec(1:4000:end),halbachData1(1:4000:end,5),'x'
    );
plot(timeVec(1:4000:end),halbachData1(1:4000:end,6),'-'
    );
plot(timeVec(1:4000:end),halbachData1(1:4000:end,7),'o'
    );
plot(timeVec,halbachData1(:,5));
plot(timeVec,halbachData1(:,6));
plot(timeVec,halbachData1(:,7));
hold off;
xlabel('Time (s)');
ylabel('Voltage (V)');
legend('Current','H07','H12','H02');
title('Phase 1 Hall-Effect Calibration');

figure(2);
yyaxis left
plot(timeVec,phaseCurrents,'--');
ylabel('Current (A)');
hold on;
yyaxis right
plot(timeVec(1:4000:end),halbachData2(1:4000:end,5),'x'
    );
plot(timeVec(1:4000:end),halbachData2(1:4000:end,6),'-'
    );
plot(timeVec(1:4000:end),halbachData2(1:4000:end,7),'o'
    );
plot(timeVec,halbachData2(:,5));
plot(timeVec,halbachData2(:,6),'-');
```

```
plot(timeVec,halbachData2(:,7),'-');
hold off;
xlabel('Time (s)');
ylabel('Voltage (V)');
legend('Current','H07','H12','H02');
title('Phase 2 Hall-Effect Calibration');

figure(3);
plot(currentAfterSettle,phase1HalbachAdj);
hold on;
h1 = plot(fitVec,P1H07fitLine,'^-');
h2 = plot(fitVec,P1H12fitLine,'o-');
h3 = plot(fitVec,P1H02fitLine,'*-');
hold off;
xlabel('Current (s)');
ylabel('Voltage (V)');
legend([h1,h2,h3],'H07','H12','H02');
title('Phase 1 Hall-Effect Calibration');

figure(4);
plot(currentAfterSettle,phase2HalbachAdj);
hold on;
h1 = plot(fitVec,P2H07fitLine,'^-');
h2 = plot(fitVec,P2H12fitLine,'o-');
h3 = plot(fitVec,P2H02fitLine,'*-');
hold off;
xlabel('Current (s)');
ylabel('Voltage (V)');
legend([h1,h2,h3],'H07','H12','H02');
title('Phase 2 Hall-Effect Calibration');

save('sensorCurrentFit.mat','coeffsP1H12','coeffsP1H02'
   ,'coeffsP2H07');
```

A.6 Motor Back-EMF Measurements

A.6.1 Back-EMF Measurement

```
runTime = 10;
sampleRate = 16000;
rpm = 900;
desiredMaxCurrent = 3.5;
windowSize = 1;
emfCollectTime = 2;
movMeanWin = 100;

Ra1 = 0.6870;
La1 = 116.1*10^-6;

Ra2 = 0.6937;
La2 = 121.6*10^-6;

m1 = 0.9216;
m2 = 1.0001;
b1 = 0;
b2 = 0;

aOChannel0 = 'ao0';      %Pin 22
aOChannel1 = 'ao1';      %Pin 21
aI0Plus = 'ai15';
aI0Min = 'ai13';

aI1Plus = 'ai12';
aI1Min = 'ai10';

aIH07 = 'ai9';          %Pin
aIH12 = 'ai11';           %Pin
aIH02 = 'ai14';          %Pin

aBoardOutput0 = 'ai6';
aBoardOutput1 = 'ai8';

finalSigFreq = rpm*9/60;
```

```
finalSpdTime = round(2*(1.2*log(finalSigFreq/9) +
   .8158),0);
steadyStateTime = runTime - finalSpdTime;
sigSpeedAccVector = linspace(0,finalSigFreq,
   finalSpdTime*sampleRate)';
sigSpeedVector = [sigSpeedAccVector; finalSigFreq*ones(
   steadyStateTime*sampleRate,1)];
finalSpdPoint = finalSpdTime*sampleRate;

phase1Impedance = sqrt(Ra1.^2 + (2*pi*sigSpeedVector*
   La1).^2);
phase2Impedance = sqrt(Ra2.^2 + (2*pi*sigSpeedVector*
   La2).^2);

filename = [num2str(rpm) 'rpm' num2str(finalSpdTime)
   ...
    'secAccel' num2str(runTime) 'secRun.xlsx'];
y0 = [];
y1 = [];

taccel = linspace(0,finalSpdTime,sampleRate*
   finalSpdTime)';
tconst = linspace(finalSpdTime,runTime,sampleRate*(
   runTime-finalSpdTime))';
y0accel = chirp(taccel,0,finalSpdTime,finalSigFreq);
y1accel = chirp(taccel,0,finalSpdTime,finalSigFreq,'
   linear',90);

yFreq =  [(finalSigFreq-0)/finalSpdTime*taccel;
   finalSigFreq*ones(length(tconst),1)];

y0ChirpEnd = y0accel(end);
y1ChirpEnd = y1accel(end);

if (y0ChirpEnd == 1)
    y0const = cos(2*pi*finalSigFreq*(tconst-
       finalSpdTime));
elseif (y0ChirpEnd == -1)
```

```
63      y0const = -cos(2*pi*finalSigFreq*(tconst-
            finalSpdTime));
64  elseif (abs(y0ChirpEnd) < 0.01)
65      if (y0accel(end-1) > y0ChirpEnd)
66          y0const = -sin(2*pi*finalSigFreq*(tconst-
                finalSpdTime));
67      else
68          y0const = sin(2*pi*finalSigFreq*(tconst-
                finalSpdTime));
69      end
70  end
71
72  if (y1ChirpEnd == 1)
73      y1const = cos(2*pi*finalSigFreq*(tconst-
            finalSpdTime));
74  elseif (y1ChirpEnd == -1)
75      y1const = -cos(2*pi*finalSigFreq*(tconst-
            finalSpdTime));
76  elseif (abs(y1ChirpEnd) < 0.01)
77      if (y1accel(end-1) > y1ChirpEnd)
78          y1const = -sin(2*pi*finalSigFreq*(tconst-
                finalSpdTime));
79      else
80          y1const = sin(2*pi*finalSigFreq*(tconst-
                finalSpdTime));
81      end
82  end
83
84  y0 = [y0accel; y0const]';
85  y1 = [y1accel; y1const]';
86  t = [taccel; tconst]';
87  % plot(t,y0,t,y1);
88
89  y0Desired = desiredMaxCurrent*y0;
90  y1Desired = desiredMaxCurrent*y1;
91
92  y0 = (desiredMaxCurrent*y0-b1)/m1;
93  y1 = (desiredMaxCurrent*y1-b2)/m2;
```

```
94
95  % plot([t t],[y0 y1]);
96
97  d = daq.getDevices;
98  s = daq.createSession('ni');
99  s.Rate = sampleRate;
100
101 addAnalogOutputChannel(s,'Dev1',aOChannel0,'voltage');
102 addAnalogOutputChannel(s,'Dev1',aOChannel1,'voltage');
103
104 ch15 = addAnalogInputChannel(s,'Dev1', aI0Plus, '
       Voltage');
105 ch13 = addAnalogInputChannel(s,'Dev1', aI0Min, 'Voltage
       ');
106 ch12 = addAnalogInputChannel(s,'Dev1', aI1Plus, '
       Voltage');
107 ch10 = addAnalogInputChannel(s,'Dev1', aI1Min, 'Voltage
       ');
108 ch6 = addAnalogInputChannel(s,'Dev1', aBoardOutput0, '
       Voltage');
109 ch8 = addAnalogInputChannel(s,'Dev1', aBoardOutput1, '
       Voltage');
110 % legend('AI0+','AI0-','AI1+','AI1-','ABoard0+','
       ABoard1+');
111
112 ch15.TerminalConfig = 'SingleEnded';
113 ch13.TerminalConfig = 'SingleEnded';
114 ch12.TerminalConfig = 'SingleEnded';
115 ch10.TerminalConfig = 'SingleEnded';
116 ch6.TerminalConfig = 'SingleEnded';
117 ch8.TerminalConfig = 'SingleEnded';
118
119 output_data0 = [y0'];
120 output_data1 = [y1'];
121
122 queueOutputData(s,[output_data0 output_data1]);
123 % plot([output_data0 output_data1]);
124 % title('Output Data Queued');
```

```
125
126 [captured_data,time] = s.startForeground();
127 beep();
128 emfCollect = emfCollectTime*sampleRate;
129 queueOutputData(s,[zeros(emfCollect,1) zeros(emfCollect
      ,1)]);
130 [emfAfterData,timeAfter] = s.startForeground();
131
132 sig = [];
133 sig(:,1) = captured_data(:,1)-captured_data(:,2);
134 sig(:,2) = captured_data(:,3)-captured_data(:,4);
135
136 current1 = sig(:,1)./phase1Impedance;
137 current2 = sig(:,2)./phase2Impedance;
138
139 b = (1/windowSize)*ones(1,windowSize);
140 a = 1;
141 sig1f = filter(b,a,sig(:,1));
142 sig1f = [sig1f((windowSize+1)/2:end); zeros((windowSize
      -1)/2,1)];
143 sig2f = filter(b,a,sig(:,2));
144 sig2f = [sig2f((windowSize+1)/2:end); zeros((windowSize
      -1)/2,1)];
145
146 emfAfter1 = emfAfterData(:,1) - emfAfterData(:,2);
147 emfAfter2 = emfAfterData(:,3) - emfAfterData(:,4);
148
149 SSData = captured_data(finalSpdPoint-1:end,:);
150 SSTime = time(finalSpdPoint-1:end);
151
152 save('backEMF.mat');
```

A.6.2 Back-EMF Data Plotting

```
1 close all;
2 load('backEMF.mat');
3
4 n = length(time);
5 tVec = time;
```

```
6
7  Vs1 = sig(:,1);
8  Vs2 = sig(:,2);
9
10 Ra1 = 0.6870;
11 La1 = 116.1*10^-6;
12
13 Ra2 = 0.6937;
14 La2 = 121.6*10^-6;
15
16 current1 = y0Desired';
17 diffC1 = [0; diff(current1)./diff(time)];
18 current2 = y1Desired';
19 diffC2 = [0; diff(current2)./diff(time)];
20
21 eps1 = Vs1 - current1*Ra1 - La1*diffC1;
22 eps2 = Vs2 - current2*Ra2 - La2*diffC2;
23
24 %%%% Phase 1 Back EMF Sectional Voltages
25 sig1Sign = sign(y0Desired)';
26 sig1SignDiff = [0; diff(sig1Sign)];
27 crossoverIndices1 = find(sig1SignDiff);
28 crossoverHalf1 = crossoverIndices1(2:2:length(
      crossoverIndices1));
29 maxV1 = [];
30 maxV1Ind = [];
31 maxEps1 = [];
32 maxEps1Ind = [];
33
34 firstInd = 1;
35 for i = 1:length(crossoverHalf1)
36     secondInd = crossoverHalf1(i);
37     [maxV1(i,1) maxV1Ind(i,1)] = max(Vs1(firstInd:
          secondInd));
38     [maxEps1(i,1) maxEps1Ind(i,1)] = max(eps1(firstInd:
          secondInd));
39     maxV1Ind(i,1) = maxV1Ind(i,1) + firstInd - 1;
40     maxEps1Ind(i,1) = maxEps1Ind(i,1) + firstInd - 1;
```

```
41      firstInd = secondInd+1;
42  end
43
44  maxEps1 = maxEps1(2:end);
45  maxEps1Ind = maxEps1Ind(2:end);
46
47  eps1Speeds = sigSpeedVector(maxEps1Ind)*60./9;
48  [fitOb1 gof1] = fit(eps1Speeds,maxEps1,'poly1');
49  coeffs1 = coeffvalues(fitOb1);
50  a1 = coeffs1(1);
51  b1 = coeffs1(2);
52  xs1 = [eps1Speeds(1);eps1Speeds(end)];
53  ys1 = a1*xs1+b1;
54
55  %%%% Phase 2 Back EMF Sectional Voltages
56  sig2Sign = sign(y1Desired)';
57  sig2SignDiff = [0; diff(sig2Sign)];
58  crossoverIndices2 = find(sig2SignDiff);
59  crossoverHalf2 = crossoverIndices2(2:2:length(
        crossoverIndices2));
60  maxV2 = [];
61  maxV2Ind = [];
62  maxEps2 = [];
63  maxEps2Ind = [];
64
65  firstInd = 1;
66  for i = 1:length(crossoverHalf2)
67      secondInd = crossoverHalf2(i);
68      [maxV2(i,1) maxV2Ind(i,1)] = max(Vs2(firstInd:
            secondInd));
69      [maxEps2(i,1) maxEps2Ind(i,1)] = max(eps2(firstInd:
            secondInd));
70      maxV2Ind(i,1) = maxV2Ind(i,1) + firstInd - 1;
71      maxEps2Ind(i,1) = maxEps2Ind(i,1) + firstInd - 1;
72      firstInd = secondInd+1;
73  end
74
75  maxEps2 = maxEps2(2:end);
```

```
maxEps2Ind = maxEps2Ind(2:end);

eps2Speeds = sigSpeedVector(maxEps2Ind)*60./9;
[fitOb2 gof2] = fit(eps2Speeds,maxEps2,'poly1');
coeffs2 = coeffvalues(fitOb2);
a2 = coeffs2(1);
b2 = coeffs2(2);
xs2 = [eps2Speeds(1);eps2Speeds(end)];
ys2 = a2*xs2+b2;

figure(1);
plot(tVec,Vs1,tVec,eps1,'*');
xlabel('Time (sec)');
ylabel('Voltage (Volts)');
title('Phase 1 Back-EMF Voltage');
legend('Phase Voltage','Phase Back-EMF','Location','
   Southwest');

figure(2);
plot(tVec,Vs2,tVec,eps2);
xlabel('Time (sec)');
ylabel('Voltage (Volts)');
title('Phase 2 Back-EMF Voltage');
legend('Phase Voltage','Phase Back-EMF','Location','
   Southwest');

figure(3);
plot(timeAfter,emfAfter1,'--',timeAfter,emfAfter2,'-.')
   ;
xlabel('Time After No Input (sec)');
ylabel('Voltage (Volts)');
title(['Back-EMF After ' num2str(rpm) ' RPM']);
legend('Phase 1 Back-EMF','Phase 2 Back-EMF');

figure(4);
hold on;
plot(eps1Speeds,maxEps1,'.',eps2Speeds,maxEps2,'^','
   MarkerSize',4);
```

```
110 plot(xs1,ys1,':',xs2,ys2,'-.','LineWidth',3)
111 hold off;
112 xlabel('Motor Speed (RPM)');
113 ylabel('Voltage (Volts)');
114 title(['Back-EMF vs RPM']);
115 legend('Phase 1 Back-EMF Peaks','Phase 2 Back-EMF Peaks
    ','Phase 1 Curve Fit','Phase 2 Curve Fit','Location'
    ,'NorthWest');
```

A.7 Motor Speed Control

```
runTime = 2;
sampleRate = 32000;
rpm = 60;
desiredMaxVolt = 3;
windowSize = 1;

aOChannel0 = 'ao0';      %Pin 22
aOChannel1 = 'ao1';      %Pin 21
aI0Plus = 'ai15';
aI0Min = 'ai13';

aI1Plus = 'ai12';
aI1Min = 'ai10';

aIH07 = 'ai9';          %Pin
aIH12 = 'ai11';           %Pin
aIH02 = 'ai14';          %Pin

finalSigFreq = round(rpm*9/60);
finalSpdTime = round((1.2*log(finalSigFreq/6) + .8158)
   ,0);
filename = [num2str(rpm) 'rpm' num2str(finalSpdTime)
   ...
    'secAccel' num2str(runTime) 'secRun.xlsx'];
y0 = [];
y1 = [];

taccel = linspace(0,finalSpdTime,sampleRate*
   finalSpdTime)';
tconst = linspace(finalSpdTime,runTime,sampleRate*(
   runTime-finalSpdTime))';
y0accel = chirp(taccel,0,finalSpdTime,finalSigFreq);
y1accel = chirp(taccel,0,finalSpdTime,finalSigFreq,'
   linear',90);

```

```
31 yFreq =  [(finalSigFreq-0)/finalSpdTime*taccel;
      finalSigFreq*ones(length(tconst),1)];
32
33 y0ChirpEnd = y0accel(end);
34 y1ChirpEnd = y1accel(end);
35
36 if (y0ChirpEnd == 1)
37     y0const = cos(2*pi*finalSigFreq*(tconst-
          finalSpdTime));
38 elseif (y0ChirpEnd == -1)
39     y0const = -cos(2*pi*finalSigFreq*(tconst-
          finalSpdTime));
40 elseif (abs(y0ChirpEnd) < 0.01)
41     if (y0accel(end-1) > y0ChirpEnd)
42         y0const = -sin(2*pi*finalSigFreq*(tconst-
              finalSpdTime));
43     else
44         y0const = sin(2*pi*finalSigFreq*(tconst-
              finalSpdTime));
45     end
46 end
47
48 if (y1ChirpEnd == 1)
49     y1const = cos(2*pi*finalSigFreq*(tconst-
          finalSpdTime));
50 elseif (y1ChirpEnd == -1)
51     y1const = -cos(2*pi*finalSigFreq*(tconst-
          finalSpdTime));
52 elseif (abs(y1ChirpEnd) < 0.01)
53     if (y1accel(end-1) > y1ChirpEnd)
54         y1const = -sin(2*pi*finalSigFreq*(tconst-
              finalSpdTime));
55     else
56         y1const = sin(2*pi*finalSigFreq*(tconst-
              finalSpdTime));
57     end
58 end
59
```

```
y0 = [y0accel; y0const]';
y1 = [y1accel; y1const]';
t = [taccel; tconst]';

plot(t,y0,t,y1);

y0Desired = desiredMaxVolt*y0;
y1Desired = desiredMaxVolt*y1;

y0 = (desiredMaxVolt*y0-b1)/m1;
y1 = (desiredMaxVolt*y1-b2)/m2;

% plot([t t],[y0 y1]);

d = daq.getDevices;
s = daq.createSession('ni');
s.Rate = sampleRate;

addAnalogOutputChannel(s,'Dev1',aOChannel0,'voltage');
addAnalogOutputChannel(s,'Dev1',aOChannel1,'voltage');
addAnalogInputChannel(s,'Dev1', aI0Plus, 'Voltage');
addAnalogInputChannel(s,'Dev1', aI0Min, 'Voltage');
addAnalogInputChannel(s,'Dev1', aI1Plus, 'Voltage');
addAnalogInputChannel(s,'Dev1', aI1Min, 'Voltage');
addAnalogInputChannel(s,'Dev1', aIH07, 'Voltage'); %Pin
    26
addAnalogInputChannel(s,'Dev1', aIH12, 'Voltage'); %Pin
    58
addAnalogInputChannel(s,'Dev1', aIH02, 'Voltage'); %Pin
    23

% output_data0 = 3*sin(linspace(0,2*pi*runTime*
   finalSigFreq,sampleRate*runTime)');
% output_data1 = 3*cos(linspace(0,2*pi*runTime*
   finalSigFreq,sampleRate*runTime)');
% output_data = linspace(3,3,sampleRate*time)';
output_data0 = [y0'];
output_data1 = [y1'];
```

```
% Last zero value is so channels are turned off

queueOutputData(s,[output_data0 output_data1]);
% plot([output_data0 output_data1]);
% title('Output Data Queued');

[captured_data,time] = s.startForeground();
queueOutputData(s,[0 0]);
[dontCare,dontcare] = s.startForeground();

sig = [];
sig(:,1) = captured_data(:,1)-captured_data(:,2);
sig(:,2) = captured_data(:,3)-captured_data(:,4);

b = (1/windowSize)*ones(1,windowSize);
a = 1;
sig1f = filter(b,a,sig(:,1));
sig1f = [sig1f((windowSize+1)/2:end); zeros((windowSize
   -1)/2,1)];
sig2f = filter(b,a,sig(:,2));
sig2f = [sig2f((windowSize+1)/2:end); zeros((windowSize
   -1)/2,1)];

% export_data = [y0Desired' y1Desired' sig(:,1) sig
   (:,2) sig1f sig2f];

% plot(export_data);
leg = {'Desired 0','Desired 1','Sig 0','Sig 1','
   Filtered 0','Filtered 1'};
% legend(leg);
% title(['Accel to ' num2str(rpm) ' RPM in ' num2str(
   finalSpdTime) ' sec']);
infoLeg = {'Sample Rate','Desired Volt','Run Time','RPM
   ','FinalSigFreq','Accel Time'};
runInfo = {sampleRate,desiredMaxVolt,runTime,rpm,
   finalSigFreq,taccel};
exportLeg = {'Time','AI0+','AI0-','AI1+','AI1-','H07','
   H12','H2','Output 0','Output 1'};
```

```
123 export_data = [time,captured_data(:,1),captured_data
       (:,2),captured_data(:,3),captured_data(:,4)...
124     captured_data(:,5),captured_data(:,6),captured_data
          (:,7),output_data0,output_data1];
125
126 testResult = {infoLeg;runInfo;exportLeg;export_data};
127 save('testResults4.mat','testResult');
128 % xlswrite(filename,infoLeg,1,'A1');
129 % xlswrite(filename,runInfo,1,'A2');
130 % xlswrite(filename,leg,1,'A3');
131 % xlswrite(filename,export_data,1,'A4');
```

A.8 Data Processing

A.8.1 Initial Data Processing

```
close all
clear;
c = 0;

% Load data and pull out saved properties and data
load('six_hundred_RPM.mat');
load('sensorCurrentFit.mat');
testResult{2,end}{end} = testResult{2,end}{end}(end);
propsLeg = testResult{1,:};
props = cell2mat(testResult{2,:});
dataLeg = testResult{3,:};
data = testResult{4,:};

%Compensate Hall-effects for phase currents
data(:,6) = data(:,6) - coeffsP2H07(1).*data(:,10);
data(:,7) = data(:,7) - coeffsP1H12(1).*data(:,9);
data(:,8) = data(:,8) - coeffsP1H02(1).*data(:,9);

%Assign properties and basic vectors
sr = props(1); % samping rate
accelTime = props(6); %time it takes motor to
   accelerate
wSS = props(4)/60; % motor steady-state speed % Hz
timeWs = 1/wSS; % samping time % sec
totalPoints = length(data);
accelPoints = accelTime*sr; % number of data points
   before SS
ssPoints = totalPoints - accelPoints;
% sigFreq = [ws*9*ones(1,ssPoints+1)]'; % Hz
sigFreq = [linspace(0,wSS*9,accelPoints) wSS*9*ones(1,
   ssPoints)]'; % signal speed (assumed linear
   acceleration) % Hz
sigFreqRot = sigFreq/sr; % Rotations per sample
sigFreqPos = 2*pi*mod(cumsum(sigFreqRot),9); % signal
   position % radians
```

```
rotorPos = sigFreqPos/9; % radians

SSDataTime = 2*accelTime*sr; % when to start analyzing
   data
lagFactor = 0; % Lag factor for time variability
   compensation
rotorPosAccel = rotorPos(1:SSDataTime) - sigFreqRot(1:
   SSDataTime)*lagFactor;
rotorPosAccel = rotorPosAccel - min(rotorPosAccel);
dataAccel = data(1:SSDataTime,:);
% dataSS = data(SSDataTime+1:SSDataTime+pointsPerRot
   *50,:);
dataSS = data(SSDataTime+1:end,:);

% Start at steady-state time
t = dataSS(:,1);
startUsefulTime = t(1);
pos = 2*pi/timeWs*mod(t,timeWs);
rots = (t(end)-t(1))*wSS;
pointsPerRot = sr/wSS;

% Differential output voltage calculation
output0 = dataSS(:,2) - dataSS(:,3);
output1 = dataSS(:,4) - dataSS(:,5);

plot(t,output0,t,output1,t,dataSS(:,6));
figure(1);
plot(pos,dataSS(:,6),'.',pos,dataSS(:,7),'.',pos,dataSS
   (:,8),'.','MarkerSize',3);
hold on;
h1 = plot(pos(1:100:end/round(rots)),dataSS(1:100:end/
   round(rots),6),'^');
h2 = plot(pos(1:100:end/round(rots)),dataSS(1:100:end/
   round(rots),7),'x');
h3 = plot(pos(1:100:end/round(rots)),dataSS(1:100:end/
   round(rots),8),'o');
hold off;
legend([h1,h2,h3],dataLeg(6:8))
```

```
61 title('Hall Effect Sensors During Steady State
      Operation');
62 xlabel('Position (rad)');
63 ylabel('Voltage (V)');
64
65 figure(2);
66 plot(pos,output0,'.',pos,output1,'.',pos,dataSS(:,9),'.
      ',pos,dataSS(:,10),'.');
67 legend(['AI0','AI1',dataLeg(9:10)])
68 title('Voltage During SS');
69 xlabel('Position');
70 ylabel('Voltage');
71
72 figure(3);
73 plot(rotorPos,data(:,6),'.',rotorPos,data(:,7),'.',
      rotorPos,data(:,8),'.');
74 legend(dataLeg(6:8))
75 title('Entire Run Hall Effect Sensor Data');
76 xlabel('Position');
77 ylabel('Voltage');
78
79 figure(4);
80 % plot(rotorPos(1:SSDataTime-1),dataAccel(:,6),'.');
81 plot(rotorPosAccel,dataAccel(:,6),'.',rotorPosAccel,
      dataAccel(:,7),'.',rotorPosAccel,dataAccel(:,8),'.')
      ;
82 title(['Accelerating Portion Hall Effect Data with Lag
      Offset c = ' num2str(lagFactor)]);
83 xlabel('Position');
84 ylabel('Voltage');
85
86 % Data correlation properties
87 % h7range = max(dataSS(:,6)) - min(dataSS(:,6));
88 % h7slope = h7range/(360/9); % volts/deg
89 h7mat = reshape(dataSS(:,7),pointsPerRot,[]); % volts
90 % h7max = max(h7mat,[],2);
91 % h7min = min(h7mat,[],2);
92 % h7diff = h7max-h7min;
```

```
% h7LeastPrecision = max(h7diff)/h7slope; % deg
% h7sdPrecision = max(h7sd)/h7slope % deg
% h7normPrecision = norm(h7diff)/length(h7diff) % deg
```

A.8.2 Data Time Compensation

```
close all
%% Show Phase Shift

h7sd = std(h7mat,0,2);
h7sdAverage = mean(h7sd); % deg
numPlots = size(h7mat,2); % number of rotations of
   motor over data

figure(1);
posInRot = (1:pointsPerRot)/pointsPerRot*2*pi;
plot(posInRot,h7mat);
xlabel('Position (rad)');
ylabel('Hall-Effect Sensor (Volt)');
title(['Hall Effect Voltage Over ' num2str(numPlots) '
   Rotations']);

figure(2);
plot(posInRot,h7mat);
title(['Matlab Chosen Colors Over ' num2str(numPlots) '
    Rotations']);
xlabel('Position (rad)');
ylabel('Hall-Effect Sensor (Volt)');
xlim([0.1 0.26]);
ylim([3.5 3.9]);

figure(3);
hold on
for i = 1:numPlots
    rat = (i-1)/numPlots;
    plot(posInRot,h7mat(:,i),'Color',[rat/2 rat rat]);
end
hold off
xlabel('Position (rad)');
```

```
ylabel('Hall-Effect Sensor (Volt)');
title(['Hall Effect Voltage Over ' num2str(numPlots) '
   Rotations']);

figure(4);
hold on
numPlots = size(h7mat,2);
for i = 1:numPlots
    rat = (i-1)/numPlots;
    plot(posInRot,h7mat(:,i),'Color',[rat/2 rat rat]);
end
hold off
xlabel('Position (rad)');
ylabel('Hall-Effect Sensor (Volt)');
title(['Black to Blue Transition Over ' num2str(
   numPlots) ' Rotations']);
xlim([0.1 0.26]);
ylim([3.5 3.9]);
%
% figure(5);
% hold on
% numPlots = size(h7mat,2);
% for i = 1:numPlots
%     plot3(posInRot,h7mat(:,i),i*ones(3200,1),'Color
   ',[0 0 (i-1)/numPlots]);
% end
% hold off
% xlabel('Position (rad)');
% ylabel('Hall-Effect Sensor (Volt');
% zlabel('Rotation Iteration');
% title(['Black to Blue Transition Over ' num2str(
   numPlots) ' Rotations']);

%% Phase Shift Compensator
% Time Based Interpolation Compensator

timeVector = data(:,1);
inputSig0 = data(:,9);
```

```
65 inputSign0 = sign(inputSig0);
66 inputSign0Diff = [0; diff(inputSign0)];
67 crossoverIndices0 = find(inputSign0Diff);
68 crossoverIndicesDiff0 = diff([1; crossoverIndices0]);
69 crossoverTimes0 = timeVector(crossoverIndices0);
70 crossoverTimeDiff0 = diff(crossoverTimes0); % sec (
      difference)
71 crossoverVel0 = [0; 1./crossoverTimeDiff0]./2; % Hz
72 motorVelAtCross0 = crossoverVel0./9; % Hz
73
74 motorVel0 = zeros(length(data),1);
75 motorPos0 = zeros(length(data),1);
76 crossoverIndicesPlus1Ind = [1; crossoverIndices0];
77 indx = 0;
78
79 for i=1:length(motorVelAtCross0)-1;
80     els = crossoverIndicesDiff0(i)+1;
81     elsVec = linspace(0,1,els);
82     velSlope = (motorVelAtCross0(i+1) -
         motorVelAtCross0(i));
83     vels = velSlope*elsVec + motorVelAtCross0(i);
84     ind1 = crossoverIndicesPlus1Ind(i);
85     ind2 = crossoverIndicesPlus1Ind(i+1);
86
87     motorVel0(ind1:ind2,1) = vels;
88     motorPos0(ind1:ind2,1) = linspace(0,pi,els) + pi*
         indx;
89     indx = indx + 1;
90     if (indx == 18)
91         indx = 0;
92     end
93
94 end
95 motorVel0(ind2:end,1) = motorVelAtCross0(end);
96 motorPos0(ind2:crossoverIndices0(end),1) = linspace(0,
      pi,crossoverIndicesDiff0(end)+1) + pi*indx;
97 remPoints = length(data) - crossoverIndices0(end);
98
```

```
indx = indx + 1;
if (indx == 18)
   indx = 0;
end

motorPos0(crossoverIndices0(end):end) = linspace(0,pi*
   remPoints/crossoverIndicesDiff0(end),remPoints+1) +
   pi*indx;
posSS = motorPos0(SSDataTime+1:end,:);

figure(6);
keepRunning = 1;
hold on;
tempPosSS = posSS./(9);
tempDataSS = dataSS(:,7);
numPlotsCount = 0;

while (keepRunning)
    ind1 = find(tempPosSS == 0,2,'first');
    if (length(ind1)<2)
       break;
    end
    if (numPlotsCount == 0)
        if (ind1(1) ~= 0)
           ind1 = ind1(1)-1;
        else
            ind1 = ind1(2)-1;
        end
    else
        ind1 = ind1(2)-1;
    end
    rat = (numPlotsCount)/numPlots;
    plot(tempPosSS(1:ind1),tempDataSS(1:ind1),'Color'
       ,[0 0 rat]);
    numPlotsCount = numPlotsCount + 1;
    tempPosSS = tempPosSS(ind1+1:end);
    tempDataSS = tempDataSS(ind1+1:end);
end
```

```
% xlim([1.57 1.75]);
% ylim([4.5 4.9]);

plot(posSS./9,dataSS(:,7),'.','Color',[0.929 0.694
   0.125]); % Same data from while loop but without
   color gradient
xlabel('Position (rad)');
ylabel('Hall-Effect Sensor (Volt');
title(['Hall Effect Voltage Over ' num2str(numPlots) '
   Rotations - Time Interpolation Compensation']);
hold off

%%% Verification of Time Interpolation Compensation
posDiff = 2*pi*[0; diff(timeVector)].*motorVel0;
posCumSum = cumsum(posDiff);
posModSS = posCumSum(SSDataTime+1:end);
posModSSAdj = posModSS - posModSS(1);
ind1 = 1;
numPlotsCount = 0;
posModInd = [];

figure(7);
hold on
for i = 1:numPlots
    ind2 = find(posModSSAdj > 2*pi*i,1) - 1;
    if (isempty(ind2))
       ind2 = length(posModSSAdj);
    end
    rat = (numPlotsCount)/numPlots;
    numPlotsCount = numPlotsCount + 1;
    posModInd(i,:) = [ind1, ind2];
    plot(mod(posModSSAdj(ind1:ind2),2*pi),dataSS(ind1:
       ind2,7),'Color',[rat/2 rat rat]);
    ind1 = ind2 + 1;

end
posCtrlTestData = posModSSAdj(posModInd(end,1):
   posModInd(end,2));
```

```
datCtrlTestData = dataSS(posModInd(end,1):posModInd(end,2),[6 7 8]);
save('ctrlTestData.mat','posCtrlTestData','datCtrlTestData');

xlabel('Position (rad)');
ylabel('Hall-Effect Sensor (Volt)');
title(['Hall Effect Voltage Over ' num2str(numPlots) ' Rotations - Time Interpolation Compensation']);
% plot(mod(posModSSAdj,2*pi),dataSS(:,7),'.');
xlim([0.1 0.26]);
ylim([3.5 3.9]);
hold off

adjustedPosition = mod(posModSSAdj,2*pi);

%%%%% Linear Phase Shift Compensator

figure(8);
phaseShiftA = (1.325-1.335)/numPlots; % Based on phase shift width
hold on
for i = 1:numPlots
    plot(posInRot-(i-1)*phaseShiftA,h7mat(:,i),'Color',[0 0 (i-1)/numPlots]);
end
hold off
xlabel('Position (rad)');
ylabel('Hall-Effect Sensor (Volt');
title(['Hall Effect Voltage Over ' num2str(numPlots) ' Rotations - Time Shift Compensation']);
% xlim([1.4 1.58]);
% ylim([4.5 4.9]);
```

A.8.3 Data Segment Linearization

```
close all
maxV = 0.75;
maxX = 0.05;
```

```
4
5  %% Linear Regions
6
7  dataSS = data(SSDataTime+1:end,:);
8  posInRotRep = adjustedPosition;
9
10 meanDataSS = mean(dataSS(:,7));
11 dataSSAdj = dataSS(:,7) - meanDataSS;
12
13 %Remove first (incomplete) linear region
14 indDel = find(abs(dataSSAdj) - maxV > 0,1)+1;
15 dataSSAdj = dataSSAdj(indDel(1):end);
16 posInRotRep = posInRotRep(1:end-indDel(1)+1);
17
18 %Remove data outside linear sections based on maxV
19 indDel = find(abs(dataSSAdj) - maxV > 0);
20 dataSSAdj(indDel) = [];
21 posInRotRep(indDel) = [];
22
23 figure(1);
24 plot(posInRotRep,dataSSAdj,'.');
25 xlabel('Position (rad)');
26 ylabel('Centered Hall-Effect Sensor (Volt)');
27 title(['Linear Regions of Data']);
28 gapIndices = find(diff(posInRotRep) - maxX > 0);
29 endIndices = find(diff(posInRotRep) < 0);
30
31 dataSSAdj = dataSSAdj + meanDataSS;
32 dataSSMat = {};
33 posRotMat = {};
34 lineSegs = 18;
35 lastInd = 1;
36 segCount = 1;
37
38 % Create dataSSMat & posRotMat with size(i,j,k) where
     length(i) is number
39 % of data points per jth linear region of kth rotation
40 for i = 1:endIndices+1
```

```
41      for j = 1:lineSegs-1
42
43          if (segCount <= length(gapIndices))
44              thisInd = gapIndices(segCount);
45          else
46              thisInd = length(dataSSAdj);
47          end
48          segCount = segCount + 1;
49          dat = dataSSAdj(lastInd:thisInd);
50          lenDat = length(dat);
51          dataSSMat(1:lenDat,j,i) = num2cell(dat);
52          posRotMat(1:lenDat,j,i) = num2cell(posInRotRep(
               lastInd:thisInd));
53          lastInd = thisInd + 1;
54          if (segCount > length(gapIndices))
55              break;
56          end
57      end
58
59      if (segCount > length(gapIndices))
60          dat = dataSSAdj(gapIndices(end)+1:end);
61          lenDat = length(dat);
62          dataSSMat(1:lenDat,j+1,i) = num2cell(dat);
63          posRotMat(1:lenDat,j+1,i) = num2cell(
               posInRotRep(gapIndices(end)+1:end));
64          break;
65      end
66
67      if (i <= length(endIndices))
68         thisInd = endIndices(i);
69      else
70         thisInd = length(dataSSAdj);
71      end
72      dat = dataSSAdj(lastInd:thisInd);
73      lenDat = length(dat);
74      dataSSMat(1:lenDat,lineSegs,i) = num2cell(dat);
75      posRotMat(1:lenDat,lineSegs,i) = num2cell(
           posInRotRep(lastInd:thisInd));
```

```
76      lastInd = thisInd + 1;
77
78  end
79
80  avgDif = [];
81  for i = 1:lineSegs
82      firstXs = cell2mat(squeeze(posRotMat(:,i,1)));
83      lastXs = cell2mat(squeeze(posRotMat(:,i,numPlots)))
            ;
84      avgDif(i) = mean(firstXs) - mean(lastXs);
85  end
86  linShiftAlpha = 0;%mean(avgDif);
87
88  minVecVec = [];
89  sds = [];
90  ps = [];
91  posMin = {};
92  datMin = {};
93  rsquare = [];
94  % gof = {};
95
96  for i = 1:lineSegs
97
98      dat = squeeze(dataSSMat(:,i,:));
99      pos = squeeze(posRotMat(:,i,:));
100     inter = cellfun('size', dat, 1);
101     vectorSizes = sum(inter);
102     minVec = min(vectorSizes);
103     minVecVec(i) = minVec;
104
105     dat = cell2mat(dat(1:minVec,:));
106     pos = cell2mat(pos(1:minVec,:));
107     addMat = repmat(linspace(0,linShiftAlpha,numPlots),
            minVec,1);
108     posShift = pos + addMat;
109     posMin(1:minVec,i,:) = num2cell(posShift);
110     datMin(1:minVec,i,:) = num2cell(dat);
111
```

```
%     [p,S] = polyfit(posShift,dat,1);
%     rval = corr2(posShift,dat);
%     rsquare(i,1) = rval.^2;
    tempPos = reshape(posShift,[],1);
    tempDat = reshape(dat,[],1);
    [fitobject gof(i)] = fit(tempPos,tempDat,'poly1');
    ps(i,1:2) = coeffvalues(fitobject);
%     ps(i,1:2) = p;
%     [y, sDelta] = polyval(p,posShift,S);
%     sds(i,:) = mean(sDelta);
end

figure(2);
hold on;
for i = 1:lineSegs
    for j = 1:numPlots
        xs = cell2mat(posMin(:,i,j));
        ys = cell2mat(datMin(:,i,j));
        if (i == lineSegs && j == numPlots)
            plotted1 = plot(xs,ys,'Color',[0 0 (j-1)/
               numPlots]);
        else
            plot(xs,ys,'Color',[0 0 (j-1)/numPlots]);
        end
    end
end

for i = 1:lineSegs
    x1 = (-maxV + meanDataSS - ps(i,2))/ps(i,1);
    x2 = (maxV + meanDataSS - ps(i,2))/ps(i,1);
    plotted2 = plot([x1,x2],[-maxV,maxV]+meanDataSS,':r
       ','LineWidth',1.5);
end
hold off;
xlabel('Position (rad)');
ylabel('Hall-Effect Sensor (Volt');
title(['Hall Effect Voltage Over ' num2str(numPlots) '
   Rotations - Linearized Region Time Shift
```

```
   Compensation']);
147 legend([plotted1 plotted2],'Time Adjusted Plot','
   Linearized Model')
```

A.8.4 All Data Channels Compensation and Linearization

```
1  close all
2  %% Show Phase Shift
3  maxV6 = 1.5;
4  maxV7 = 0.75;
5  maxV8 = 0.50;
6
7  maxX = 0.05;
8  h7sd = std(h7mat,0,2);
9  h7sdAverage = mean(h7sd); % deg
10 numPlots = size(h7mat,2); % number of rotations of
      motor over data
11 hallEffectVectors = [6 7 8]; % Hall Effect 7, 12, 2
12
13 %% Phase Shift Compensator
14 % Time Based Interpolation Compensator
15
16 timeVector = data(:,1);
17 inputSig0 = data(:,9);
18 inputSign0 = sign(inputSig0);
19 inputSign0Diff = [0; diff(inputSign0)];
20 crossoverIndices0 = find(inputSign0Diff);
21 crossoverIndicesDiff0 = diff([1; crossoverIndices0]);
22 crossoverTimes0 = timeVector(crossoverIndices0);
23 crossoverTimeDiff0 = diff(crossoverTimes0); % sec (
      difference)
24 crossoverVel0 = [0; 1./crossoverTimeDiff0]./2; % Hz
25 motorVelAtCross0 = crossoverVel0./9; % Hz
26
27 motorVel0 = zeros(length(data),1);
28 motorPos0 = zeros(length(data),1);
29 crossoverIndicesPlus1Ind = [1; crossoverIndices0];
30 indx = 0;
31
```

```
for i=1:length(motorVelAtCross0)-1;
    els = crossoverIndicesDiff0(i)+1;
    elsVec = linspace(0,1,els);
    velSlope = (motorVelAtCross0(i+1) -
       motorVelAtCross0(i));
    vels = velSlope*elsVec + motorVelAtCross0(i);
    ind1 = crossoverIndicesPlus1Ind(i);
    ind2 = crossoverIndicesPlus1Ind(i+1);

    motorVel0(ind1:ind2,1) = vels;
    motorPos0(ind1:ind2,1) = linspace(0,pi,els) + pi*
       indx;
    indx = indx + 1;
    if (indx == 18)
       indx = 0;
    end

end
motorVel0(ind2:end,1) = motorVelAtCross0(end);
motorPos0(ind2:crossoverIndices0(end),1) = linspace(0,
   pi,crossoverIndicesDiff0(end)+1) + pi*indx;
remPoints = length(data) - crossoverIndices0(end);

indx = indx + 1;
if (indx == 18)
   indx = 0;
end

%%% Motor Speed Influence on Hall Sensors
% figure(8);
% hold on;
% for i = 1:18
%    vecPlot = i:18:length(crossoverIndices0);
%    crossImp = motorVelAtCross0(vecPlot);
%    dataImp = hallCrossover(vecPlot);
%    plot(crossImp,dataImp);
% end
% hold off;
```

```
% title('Motor Speed vs Hall-Effect Sensor');
% xlabel('Motor Speed (RPS)');
% ylabel('Hall-Effect Voltage (V)');

%%% Verification of Time Interpolation Compensation
posDiff = 2*pi*[0; diff(timeVector)].*motorVel0;
posCumSum = cumsum(posDiff);
posModSS = posCumSum(SSDataTime+1:end);
posModSSAdj = posModSS - posModSS(1);
ind1 = 1;
posModInd = [];
adjustedPosition = mod(posModSSAdj,2*pi);

%% Linear Regions

posInRotRep = adjustedPosition;

meanDataSS = mean(dataSS(:,hallEffectVectors));
dataSSAdj = dataSS(:,hallEffectVectors) - meanDataSS;

%Remove first (incomplete) linear region from all three
    datasets
indDel6 = find(abs(dataSSAdj(:,1)) - maxV6 > 0,1);
indDel7 = find(abs(dataSSAdj(:,2)) - maxV7 > 0,1);
indDel8 = find(abs(dataSSAdj(:,3)) - maxV8 > 0,1);
indDel = max([indDel6 indDel7 indDel8])+1;

dataSSAdj = dataSSAdj(indDel:end,:);
posInRotRep = posInRotRep(1:end-indDel+1);

[maxH07Voltage, maxH07VoltageInd] = max(dataSSAdj(:,1))
   ;
maxH07Voltage = maxH07Voltage + meanDataSS(1);
maxH07VoltagePos = posInRotRep(maxH07VoltageInd);
[minH07Voltage, minH07VoltageInd] = min(dataSSAdj(:,1))
   ;
minH07Voltage = minH07Voltage + meanDataSS(1);
minH07VoltagePos = posInRotRep(minH07VoltageInd);
```

```
102
103 %Remove data outside linear sections based on maxV
104 dataSSAdj6 = dataSSAdj(:,1);
105 dataSSAdj7 = dataSSAdj(:,2);
106 dataSSAdj8 = dataSSAdj(:,3);
107 indDel6 = find(abs(dataSSAdj6) - maxV6 > 0);
108 indDel7 = find(abs(dataSSAdj7) - maxV7 > 0);
109 indDel8 = find(abs(dataSSAdj8) - maxV8 > 0);
110
111 posInRotRep6 = posInRotRep;
112 posInRotRep7 = posInRotRep;
113 posInRotRep8 = posInRotRep;
114
115 dataSSAdj6(indDel6) = [];
116 dataSSAdj6 = dataSSAdj6 + meanDataSS(1);
117 posInRotRep6(indDel6) = [];
118 dataSSAdj7(indDel7) = [];
119 dataSSAdj7 = dataSSAdj7 + meanDataSS(2);
120 posInRotRep7(indDel7) = [];
121 dataSSAdj8(indDel8) = [];
122 dataSSAdj8 = dataSSAdj8 + meanDataSS(3);
123 posInRotRep8(indDel8) = [];
124
125 figure(1);
126 plot(posInRotRep6,dataSSAdj6,'.',posInRotRep7,
      dataSSAdj7,'.',posInRotRep8,dataSSAdj8,'.');
127 hold on;
128 h1 = plot(posInRotRep6(1:20:round(end/rots)),dataSSAdj6
      (1:20:round(end/rots)),'^');
129 h2 = plot(posInRotRep7(1:20:round(end/rots)),dataSSAdj7
      (1:20:round(end/rots)),'x');
130 h3 = plot(posInRotRep8(1:20:round(end/rots)),dataSSAdj8
      (1:20:round(end/rots)),'o');
131 % plot(maxH07VoltagePos,maxH07Voltage,'.',
      minH07VoltagePos,minH07Voltage,'.');
132 hold off;
133 xlabel('Position (rad)');
134 ylabel('Centered Hall-Effect Sensor (Volt)');
```

```
135 title(['Linear Regions of Data']);
136 legend([h1 h2 h3],'Hall Effect 7','Hall Effect 12','
       Hall Effect 2');
137
138 gapIndices6 = find(diff(posInRotRep6) - maxX > 0);
139 endIndices6 = find(diff(posInRotRep6) < 0);
140 gapIndices7 = find(diff(posInRotRep7) - maxX > 0);
141 endIndices7 = find(diff(posInRotRep7) < 0);
142 gapIndices8 = find(diff(posInRotRep8) - maxX > 0);
143 endIndices8 = find(diff(posInRotRep8) < 0);
144
145 if (length(gapIndices6) ~= length(gapIndices6) ||
       length(gapIndices7) ~= length(gapIndices8))
146     error('gapIndices are not the same length');
147 end
148
149 dataSSMat6 = {};
150 posRotMat6 = {};
151 dataSSMat7 = {};
152 posRotMat7 = {};
153 dataSSMat8 = {};
154 posRotMat8 = {};
155 lineSegs = 18;
156 lastInd6 = 1;
157 lastInd7 = 1;
158 lastInd8 = 1;
159 segCount = 1;
160
161 % Create dataSSMat & posRotMat with size(i,j,k) where
       length(i) is number
162 % of data points per jth linear region of kth rotation
163 for i = 1:endIndices6+1
164     for j = 1:lineSegs-1
165
166         if (segCount <= length(gapIndices6))
167             thisInd6 = gapIndices6(segCount);
168             thisInd7 = gapIndices7(segCount);
169             thisInd8 = gapIndices8(segCount);
```

```
170         else
171             thisInd6 = length(dataSSAdj6);
172             thisInd7 = length(dataSSAdj7);
173             thisInd8 = length(dataSSAdj8);
174         end
175         segCount = segCount + 1;
176 
177         dat6 = dataSSAdj6(lastInd6:thisInd6);
178         dat7 = dataSSAdj7(lastInd7:thisInd7);
179         dat8 = dataSSAdj8(lastInd8:thisInd8);
180         lenDat6 = length(dat6);
181         lenDat7 = length(dat7);
182         lenDat8 = length(dat8);
183 
184         dataSSMat6(1:lenDat6,j,i) = num2cell(dat6);
185         posRotMat6(1:lenDat6,j,i) = num2cell(
              posInRotRep6(lastInd6:thisInd6));
186         dataSSMat7(1:lenDat7,j,i) = num2cell(dat7);
187         posRotMat7(1:lenDat7,j,i) = num2cell(
              posInRotRep7(lastInd7:thisInd7));
188         dataSSMat8(1:lenDat8,j,i) = num2cell(dat8);
189         posRotMat8(1:lenDat8,j,i) = num2cell(
              posInRotRep8(lastInd8:thisInd8));
190 
191         lastInd6 = thisInd6 + 1;
192         lastInd7 = thisInd7 + 1;
193         lastInd8 = thisInd8 + 1;
194         if (segCount > length(gapIndices6))
195             break;
196         end
197     end
198 
199     if (segCount > length(gapIndices6))
200         dat6 = dataSSAdj6(gapIndices6(end)+1:end);
201         dat7 = dataSSAdj7(gapIndices7(end)+1:end);
202         dat8 = dataSSAdj8(gapIndices8(end)+1:end);
203         lenDat6 = length(dat6);
204         lenDat7 = length(dat7);
```

```
205         lenDat8 = length(dat8);
206         dataSSMat6(1:lenDat6,j+1,i) = num2cell(dat6);
207         posRotMat6(1:lenDat6,j+1,i) = num2cell(
               posInRotRep6(gapIndices6(end)+1:end));
208         dataSSMat7(1:lenDat7,j+1,i) = num2cell(dat7);
209         posRotMat7(1:lenDat7,j+1,i) = num2cell(
               posInRotRep7(gapIndices7(end)+1:end));
210         dataSSMat8(1:lenDat8,j+1,i) = num2cell(dat8);
211         posRotMat8(1:lenDat8,j+1,i) = num2cell(
               posInRotRep8(gapIndices8(end)+1:end));
212         break;
213     end
214
215     if (i <= length(endIndices6))
216        thisInd6 = endIndices6(i);
217        thisInd7 = endIndices7(i);
218        thisInd8 = endIndices8(i);
219     else
220        thisInd6 = length(dataSSAdj6);
221        thisInd7 = length(dataSSAdj7);
222        thisInd8 = length(dataSSAdj8);
223     end
224     dat6 = dataSSAdj6(lastInd6:thisInd6);
225     dat7 = dataSSAdj7(lastInd7:thisInd7);
226     dat8 = dataSSAdj8(lastInd8:thisInd8);
227     lenDat6 = length(dat6);
228     lenDat7 = length(dat7);
229     lenDat8 = length(dat8);
230
231     dataSSMat6(1:lenDat6,lineSegs,i) = num2cell(dat6);
232     posRotMat6(1:lenDat6,lineSegs,i) = num2cell(
           posInRotRep6(lastInd6:thisInd6));
233     lastInd6 = thisInd6 + 1;
234     dataSSMat7(1:lenDat7,lineSegs,i) = num2cell(dat7);
235     posRotMat7(1:lenDat7,lineSegs,i) = num2cell(
           posInRotRep7(lastInd7:thisInd7));
236     lastInd7 = thisInd7 + 1;
237     dataSSMat8(1:lenDat8,lineSegs,i) = num2cell(dat8);
```

```
238     posRotMat8(1:lenDat8,lineSegs,i) = num2cell(
            posInRotRep8(lastInd8:thisInd8));
239     lastInd8 = thisInd8 + 1;
240
241 end
242
243 % avgDif = [];
244 % for i = 1:lineSegs
245 %     firstXs = cell2mat(squeeze(posRotMat(:,i,1)));
246 %     lastXs = cell2mat(squeeze(posRotMat(:,i,numPlots)
        ));
247 %     avgDif(i) = mean(firstXs) - mean(lastXs);
248 % end
249 % linShiftAlpha = 0;%mean(avgDif);
250
251 minVecVec = [];
252 sds = [];
253 ps = [];
254 posMin = {};
255 datMin = {};
256
257 figure(3);
258 hold on;
259 for i = 1:lineSegs
260
261     dat6 = squeeze(dataSSMat6(:,i,:));
262 %     dat6 = num2cell(cell2mat(dat6) + meanDataSS(1));
263     pos6 = squeeze(posRotMat6(:,i,:));
264     inter6 = cellfun('size', dat6, 1);
265     dat7 = squeeze(dataSSMat7(:,i,:));
266 %     dat7 = num2cell(cell2mat(dat7) + meanDataSS(2));
267     pos7 = squeeze(posRotMat7(:,i,:));
268     inter7 = cellfun('size', dat7, 1);
269     dat8 = squeeze(dataSSMat8(:,i,:));
270 %     dat8 = num2cell(cell2mat(dat8) + meanDataSS(3));
271     pos8 = squeeze(posRotMat8(:,i,:));
272     inter8 = cellfun('size', dat8, 1);
273
```

```
274     vectorSizes6 = sum(inter6);
275     minVec6 = min(vectorSizes6);
276     minVecVec6(i) = minVec6;
277     vectorSizes7 = sum(inter7);
278     minVec7 = min(vectorSizes7);
279     minVecVec7(i) = minVec7;
280     vectorSizes8 = sum(inter8);
281     minVec8 = min(vectorSizes8);
282     minVecVec8(i) = minVec8;
283
284     dat6 = cell2mat(dat6(1:minVec6,:));
285     pos6 = cell2mat(pos6(1:minVec6,:));
286     dat7 = cell2mat(dat7(1:minVec7,:));
287     pos7 = cell2mat(pos7(1:minVec7,:));
288     dat8 = cell2mat(dat8(1:minVec8,:));
289     pos8 = cell2mat(pos8(1:minVec8,:));
290
291     addMat = 0;%repmat(linspace(0,linShiftAlpha,
          numPlots),minVec,1);
292     posShift6 = pos6 + addMat;
293     posMin6(1:minVec6,i,:) = num2cell(posShift6);
294     datMin6(1:minVec6,i,:) = num2cell(dat6);
295     posShift7 = pos7 + addMat;
296     posMin7(1:minVec7,i,:) = num2cell(posShift7);
297     datMin7(1:minVec7,i,:) = num2cell(dat7);
298     posShift8 = pos8 + addMat;
299     posMin8(1:minVec8,i,:) = num2cell(posShift8);
300     datMin8(1:minVec8,i,:) = num2cell(dat8);
301
302 %     [p6,S6] = polyfit(posShift6,dat6,1);
303 %     ps6(i,1:2) = p6;
304 %     [y6, sDelta6] = polyval(p6,posShift6,S6);
305 %     sds6(i,:) = mean(sDelta6);
306
307     tempPos6 = reshape(posShift6,[],1);
308     tempDat6 = reshape(dat6,[],1);
309 %     [fitobject6 gof6(i)] = fit(tempPos6,tempDat6,'
      poly1');
```

```
310     [fitobject6 gof6(i)] = fit(tempDat6,tempPos6,'poly1
            ');
311     ps6(i,1:2) = coeffvalues(fitobject6);
312     plot(tempDat6,tempDat6*ps6(i,1) + ps6(i,2),'.');
313
314     tempPos7 = reshape(posShift7,[],1);
315     tempDat7 = reshape(dat7,[],1);
316 %       [fitobject7 gof7(i)] = fit(tempPos7,tempDat7,'
        poly1');
317     [fitobject7 gof7(i)] = fit(tempDat7,tempPos7,'poly1
            ');
318     ps7(i,1:2) = coeffvalues(fitobject7);
319
320     tempPos8 = reshape(posShift8,[],1);
321     tempDat8 = reshape(dat8,[],1);
322 %       [fitobject8 gof8(i)] = fit(tempPos8,tempDat8,'
        poly1');
323     [fitobject8 gof8(i)] = fit(tempDat8,tempPos8,'poly1
            ');
324     ps8(i,1:2) = coeffvalues(fitobject8);
325 end
326
327 figure(2);
328 hold on;
329 for i = 1:lineSegs
330 %     for j = 1:numPlots
331         xs6 = cell2mat(squeeze(posMin6(:,i,:)));
332         ys6 = cell2mat(squeeze(datMin6(:,i,:)));
333         xs7 = cell2mat(squeeze(posMin7(:,i,:)));
334         ys7 = cell2mat(squeeze(datMin7(:,i,:)));
335         xs8 = cell2mat(squeeze(posMin8(:,i,:)));
336         ys8 = cell2mat(squeeze(datMin8(:,i,:)));
337
338 %         bounds6(i,1) = min(min(xs6));
339 %         bounds6(i,2) = max(max(xs6));
340 %         bounds7(i,1) = min(min(xs7));
341 %         bounds7(i,2) = max(max(xs7));
342 %         bounds8(i,1) = min(min(xs8));
```

```
343 %           bounds8(i,2) = max(max(xs8));
344
345         v6Bounds(i,1) = sign(mean(ys6(1,:)) -
               meanDataSS(1))*maxV6 + meanDataSS(1);
346         v6Bounds(i,2) = sign(mean(ys6(end,:)) -
               meanDataSS(1))*maxV6 + meanDataSS(1);
347         v7Bounds(i,1) = sign(mean(ys7(1,:)) -
               meanDataSS(2))*maxV7 + meanDataSS(2);
348         v7Bounds(i,2) = sign(mean(ys7(end,:)) -
               meanDataSS(2))*maxV7 + meanDataSS(2);
349         v8Bounds(i,1) = sign(mean(ys8(1,:)) -
               meanDataSS(3))*maxV8 + meanDataSS(3);
350         v8Bounds(i,2) = sign(mean(ys8(end,:)) -
               meanDataSS(3))*maxV8 + meanDataSS(3);
351
352         p6Bounds(i,1) = v6Bounds(i,1)*ps6(i,1) + ps6(i
               ,2);
353         p6Bounds(i,2) = v6Bounds(i,2)*ps6(i,1) + ps6(i
               ,2);
354         p7Bounds(i,1) = v7Bounds(i,1)*ps7(i,1) + ps7(i
               ,2);
355         p7Bounds(i,2) = v7Bounds(i,2)*ps7(i,1) + ps7(i
               ,2);
356         p8Bounds(i,1) = v8Bounds(i,1)*ps8(i,1) + ps8(i
               ,2);
357         p8Bounds(i,2) = v8Bounds(i,2)*ps8(i,1) + ps8(i
               ,2);
358
359         bounds6(i,1) = min(min(xs6));
360         bounds6(i,2) = max(max(xs6));
361         bounds7(i,1) = min(min(xs7));
362         bounds7(i,2) = max(max(xs7));
363         bounds8(i,1) = min(min(xs8));
364         bounds8(i,2) = max(max(xs8));
365
366 %           rat = (j-1)/numPlots;
367         rat = 1;
368         if (i == lineSegs)% && j == numPlots)
```

```
369             plotted16 = plot(xs6,ys6,'.','Color',[0 0
                    rat],'MarkerSize',3);
370             plotted17 = plot(xs7,ys7,'.','Color',[0 rat
                     0],'MarkerSize',3);
371             plotted18 = plot(xs8,ys8,'.','Color',[rat 0
                     0],'MarkerSize',3);
372             plotted16 = plotted16(end);
373             plotted17 = plotted17(end);
374             plotted18 = plotted18(end);
375         else
376             plot(xs6,ys6,'.','Color',[0 0 rat],'
                    MarkerSize',3);
377             plot(xs7,ys7,'.','Color',[0 rat 0],'
                    MarkerSize',3);
378             plot(xs8,ys8,'.','Color',[rat 0 0],'
                    MarkerSize',3);
379         end
380 %     end
381 end
382
383 for i = 1:lineSegs
384     x16 = (-maxV6 + meanDataSS(1) - ps6(i,2))/ps6(i,1);
385     x26 = (maxV6 + meanDataSS(1) - ps6(i,2))/ps6(i,1);
386     x17 = (-maxV7 + meanDataSS(2) - ps7(i,2))/ps7(i,1);
387     x27 = (maxV7 + meanDataSS(2) - ps7(i,2))/ps7(i,1);
388     x18 = (-maxV8 + meanDataSS(3) - ps8(i,2))/ps8(i,1);
389     x28 = (maxV8 + meanDataSS(3) - ps8(i,2))/ps8(i,1);
390
391 %     plotted26 = plot([x16,x26],[-maxV6,maxV6] +
    meanDataSS(1),':y','MarkerSize',4,'LineWidth',1.5);
392 %     plotted27 = plot([x17,x27],[-maxV7,maxV7] +
    meanDataSS(2),':m','MarkerSize',4,'LineWidth',1.5);
393 %     plotted28 = plot([x18,x28],[-maxV8,maxV8] +
    meanDataSS(3),':c','MarkerSize',4,'LineWidth',1.5);
394
395     plotted26 = plot(p6Bounds(i,:),v6Bounds(i,:),'-.^k'
        ,'MarkerSize',4,'LineWidth',1.5);
```

```
396     plotted27 = plot(p7Bounds(i,:),v7Bounds(i,:),'-.xm'
            ,'MarkerSize',4,'LineWidth',1.5);
397     plotted28 = plot(p8Bounds(i,:),v8Bounds(i,:),'-.oc'
            ,'MarkerSize',4,'LineWidth',1.5);
398 end
399 hold off;
400 xlabel('Position (rad)');
401 ylabel('Hall-Effect Sensor (Volt');
402 title(['Hall Effect Voltage Over ' num2str(numPlots) '
        Rotations - Linearized Region Time Shift
        Compensation']);
403 plotVec = [plotted16 plotted17 plotted18 plotted26
        plotted27 plotted28];
404 legend(plotVec,'Hall Effect 7','Hall Effect 12','Hall
        Effect 2','Linearized Model 7','Linearized Model 12'
        ,'Linearized Model 2')
405
406 save('posFromVCTRLEssentials.mat','bounds6','bounds7','
        bounds8','dataSSAdj6','dataSSAdj7',...
407     'dataSSAdj8','lineSegs','maxV6','maxV7','maxV8','
            meanDataSS','posInRotRep6',...
408     'posInRotRep7','posInRotRep8','ps6','ps7','ps8','
            v6Bounds','v7Bounds'...
409     ,'v8Bounds','p6Bounds','p7Bounds','p8Bounds','
            maxH07Voltage','maxH07VoltagePos'...
410     ,'minH07Voltage','minH07VoltagePos');
411
412 % boundSep6 = (bounds6(2:end,1) + bounds6(1:end-1,2))
        /2;
413 % boundSep7 = (bounds7(2:end,1) + bounds7(1:end-1,2))
        /2;
414 % boundSep8 = (bounds8(2:end,1) + bounds8(1:end-1,2))
        /2;
415 % x = (1:lineSegs-1)';
416 % boundLines6 = fit(x,boundSep6,'poly1');
417 % boundLineCoeffs6(1:2) = coeffvalues(boundLines6);
418 % boundLines7 = fit(x,boundSep7,'poly1');
419 % boundLineCoeffs7(1:2) = coeffvalues(boundLines7);
```

```
% boundLines8 = fit(x,boundSep8,'poly1');
% boundLineCoeffs8(1:2) = coeffvalues(boundLines8);
%
% figure(3);
% hold on;
% plot(posInRotRep6,dataSSAdj6,'.',posInRotRep7,
   dataSSAdj7,'.',posInRotRep8,dataSSAdj8,'.');
% for i = 1:lineSegs-1
%     x6 = i*boundLineCoeffs6(1) + boundLineCoeffs6(2);
%     x7 = i*boundLineCoeffs7(1) + boundLineCoeffs8(2);
%     x8 = i*boundLineCoeffs7(1) + boundLineCoeffs8(2);
%     plot([x6 x6],[-maxV6,maxV6],':','LineWidth',1.5);
%     plot([x7 x7],[-maxV7,maxV7],':','LineWidth',1.5);
%     plot([x8 x8],[-maxV8,maxV8],':','LineWidth',1.5);
% end
%
% hold off;
% xlabel('Position (rad)');
% ylabel('Centered Hall-Effect Sensor (Volt)');
% title(['Linear Regions of Data']);
% legend('Hall Effect 7','Hall Effect 12','Hall Effect
   2','Hall Effect 7 Boundaries');
```

A.9 Position Control

A.9.1 Position Controller

```
close all;

desiredMaxCurrent = 3;
startCurrent = 1.5;
settlePoints = 100;
vCutOff = 2;
vCutOffMinor = 1.25;
sampleRate = 1200;
global spinPoints;
spinPoints = sampleRate*4;
global scanQueueLimit;
scanQueueLimit = sampleRate/16;
desiredStartPos = 5.05; % rad
zeroEntries = sampleRate*60;
posCorrectTime = 0.5;

global Kp;
global Ki;
global Kd;
global Ksat;
global KiCounter;
global KiSat;
global lastError;
global lastTime;

Ku = 50;
Tu = 2.3;

Kp = 0.2*Ku;
Ki = Tu/2;
Kd = Tu/3;
% Ksat = 1000;
% KiSat = pi/2/9/10;
KiCounter = 0;
lastError = 0;
```

```
lastTime = 0;

global data;
global time;
global position;
global currentPos;
global desiredPos;
global posSegment;
global coeffsPs6;
global coeffsPs7;
global coeffsPs8;
global meanValues;
global maxCurrent;
global dataSize;
global P1H02cToV;
global P1H12cToV;
global P2H07cToV;
global phase1Current;
global phase2Current;
global timeCounter;
global counterAmount;
global outSaver;
global currentElectricalPos;
global desiredElectricalPos;

windowSize = 7;
smoothPkFinder = 150;
load('posFromVCTRLEssentials.mat');
load('sensorCurrentFit.mat');
% load('ctrlTestDataSlow.mat');

P1H02cToV = coeffsP1H02(1);
P1H12cToV = coeffsP1H12(1);
P2H07cToV = coeffsP2H07(1);
phase1Current = 0;
phase2Current = 0;
coeffsPs6 = ps6;
coeffsPs7 = ps7;
```

```
74  coeffsPs8 = ps8;
75  meanValues = meanDataSS;
76  maxCurrent = desiredMaxCurrent;
77  allBounds(:,1) = max([p6Bounds(:,1),p7Bounds(:,1),
       p8Bounds(:,1)],[],2);
78  allBounds(:,2) = min([p6Bounds(:,2),p7Bounds(:,2),
       p8Bounds(:,2)],[],2);
79  posDestinations = flipud((p6Bounds(:,1) + p6Bounds(:,2)
       )/2);
80
81  lowerLimit6 = desiredStartPos > p6Bounds(:,1);
82  upperLimit6 = desiredStartPos < p6Bounds(:,2);
83  posStartSegment = find(and(lowerLimit6,upperLimit6));
84
85  % ps6(:,2) = ps6(:,2) + meanDataSS(1);
86  % ps7(:,2) = ps7(:,2) + meanDataSS(2);
87  % ps8(:,2) = ps8(:,2) + meanDataSS(3);
88
89  d = daq.getDevices;
90  s = daq.createSession('ni');
91  s.Rate = sampleRate;
92
93  aOChannel0 = 'ao0';      %Pin 22
94  aOChannel1 = 'ao1';      %Pin 21
95  aI0Plus = 'ai15';
96  aI0Min = 'ai13';
97
98  aI1Plus = 'ai12';
99  aI1Min = 'ai10';
100
101 aIH07 = 'ai9';          %Pin
102 aIH12 = 'ai11';           %Pin
103 aIH02 = 'ai14';
104
105 addAnalogOutputChannel(s,'Dev1',aOChannel0,'voltage');
106 addAnalogOutputChannel(s,'Dev1',aOChannel1,'voltage');
107 addAnalogInputChannel(s,'Dev1', aI0Plus, 'Voltage');
108 addAnalogInputChannel(s,'Dev1', aI0Min, 'Voltage');
```

```
addAnalogInputChannel(s,'Dev1', aI1Plus, 'Voltage');
addAnalogInputChannel(s,'Dev1', aI1Min, 'Voltage');
addAnalogInputChannel(s,'Dev1', aIH07, 'Voltage'); %Pin
    26
addAnalogInputChannel(s,'Dev1', aIH12, 'Voltage'); %Pin
    58
addAnalogInputChannel(s,'Dev1', aIH02, 'Voltage'); %Pin
    23
ch15.TerminalConfig = 'SingleEnded';
ch13.TerminalConfig = 'SingleEnded';
ch12.TerminalConfig = 'SingleEnded';
ch10.TerminalConfig = 'SingleEnded';
ch9.TerminalConfig = 'SingleEnded';
ch11.TerminalConfig = 'SingleEnded';
ch14.TerminalConfig = 'SingleEnded';
% set(s,'ExternalTriggerTimeout',200);

[posInRotRep6SmoothOrder, posSmoothInd] = sort(
   posInRotRep6);
dataSSAdj6SmoothOrder = dataSSAdj6(posSmoothInd);
dataSSAdj6Smooth = smooth(dataSSAdj6SmoothOrder,
   smoothPkFinder);
dataSSAdj6Smooth = smooth(dataSSAdj6Smooth,
   smoothPkFinder);
% dataSSAdj6Smooth = smooth(dataSSAdj6Smooth,
   smoothPkFinder);

tVec_Spin1 = (linspace(0,spinPoints-1,spinPoints)*2*pi
   *9/spinPoints)';
out0_Spin1 = startCurrent*sin(tVec_Spin1);
out1_Spin1 = startCurrent*cos(tVec_Spin1);

queueOutputData(s,[out0_Spin1(1)*ones(posCorrectTime*
   sampleRate,1), out1_Spin1(1)*ones(posCorrectTime*
   sampleRate,1)]);
[align_for_spin, ~] = s.startForeground();

queueOutputData(s,[out0_Spin1 out1_Spin1]);
```

```
137 [captured_data_Spin1, ~] = s.startForeground();
138 posData1 = linspace(0,2*pi,spinPoints);
139 datData1 = captured_data_Spin1(:,5);
140 smoothDat1 = smooth(datData1,windowSize);
141 smoothDat1 = smooth(smoothDat1,windowSize);
142
143 [maxMeasH07Voltage, maxMeasH07VoltageInd] = max(
       smoothDat1);
144 maxMeasH07VoltagePos = posData1(maxMeasH07VoltageInd);
145 [minMeasH07Voltage, minMeasH07VoltageInd] = min(
       smoothDat1);
146 minMeasH07VoltagePos = posData1(minMeasH07VoltageInd);
147 currentRef = maxMeasH07VoltagePos;
148
149 figure(1);
150 plot(posData1,datData1);
151 hold on;
152 plot(posInRotRep6SmoothOrder,dataSSAdj6Smooth,'--');
153 plot(maxH07VoltagePos,maxH07Voltage,'*',
       minH07VoltagePos,minH07Voltage,'*');
154 plot(maxMeasH07VoltagePos,maxMeasH07Voltage,'^',
       minMeasH07VoltagePos,minMeasH07Voltage,'^');
155 % plot(posData1CO(dipLocs),dipPks,'.');
156 % plot(posInRotRep6SmoothOrder(smoothDipLocs),
       smoothDipPks,'.');
157 xlabel('Motor Position (rad)');
158 ylabel('Hall-Effect Response (Volt)');
159 title(['Initial Spin To Determine Position']);
160 legend('Measured','Reference');
161 % legend('Smoothed Measured Data','Absolute Position
       Reference Frame','Smoothed Absolute Reference Frame
       ');
162 hold off;
163
164 maxMaxDist = mod(maxH07VoltagePos -
       maxMeasH07VoltagePos,2*pi);
165 minMinDist = mod(minH07VoltagePos -
       minMeasH07VoltagePos,2*pi);
```

```
166 if ((maxMaxDist-minMinDist)/maxMaxDist > 0.05)
167     queueOutputData(s,[0 0]);
168     [dontCare,dontcare] = s.startForeground();
169     error('Peak locations do not match');
170 end
171 
172 figure(2);
173 hold on;
174 for i = 1:lineSegs
175     plotted26 = plot(p6Bounds(i,:),v6Bounds(i,:),':y','
        MarkerSize',4,'LineWidth',1.5);
176     plotted27 = plot(p7Bounds(i,:),v7Bounds(i,:),':m','
        MarkerSize',4,'LineWidth',1.5);
177     plotted28 = plot(p8Bounds(i,:),v8Bounds(i,:),':c','
        MarkerSize',4,'LineWidth',1.5);
178 end
179 ylabel('Hall-Effect Response (Volt)');
180 xlabel('Motor Position (rad)');
181 savefig('LinearPlots.fig');
182 h1 = plot(maxH07VoltagePos,maxH07Voltage,'*');
183 h2 = plot(minH07VoltagePos,minH07Voltage,'*');
184 h3 = plot(maxMeasH07VoltagePos,maxMeasH07Voltage,'^');
185 h4 = plot(minMeasH07VoltagePos,minMeasH07Voltage,'^');
186 h5 = plot([maxMaxDist,maxMaxDist],[0.5,4.5]);
187 legend([h1,h3,h2,h4,h5],'Reference Max','Measured Max',
    'Reference Min','Measured Min','Current Pos');
188 currentPos = maxMaxDist;
189 
190 posRefFrameCorrection = mod(desiredStartPos-currentPos
    ,2*pi);
191 % posCorrectedRefFrame = mod(posData1 +
    posRefFrameCorrection,2*pi);
192 tPoints = round(posRefFrameCorrection/(2*pi)*spinPoints
    );
193 tVec_frameCorrect = linspace(0,posRefFrameCorrection*9,
    abs(tPoints))';
194 out0_frameCorrect = startCurrent*sin(tVec_frameCorrect)
    ;
```

```
195 out1_frameCorrect = startCurrent*cos(tVec_frameCorrect)
       ;
196
197 queueOutputData(s,[out0_frameCorrect out1_frameCorrect
       ]);
198 [captured_data_frameCorrect, time_dat_frameCorrect] = s
       .startForeground();
199
200 posCorrectVec = [out0_frameCorrect(end)*ones(sampleRate
       *posCorrectTime,1), out1_frameCorrect(end)*ones(
       sampleRate*posCorrectTime,1)];
201 queueOutputData(s,posCorrectVec);
202 [captured_data_posCorrect, time_dat_posCorrect] = s.
       startForeground();
203 v07 = captured_data_posCorrect(:,5) - posCorrectVec(end
       ,2)*P2H07cToV;
204 p07 = v07*coeffsPs6(posStartSegment,1) + coeffsPs6(
       posStartSegment,2);
205 p07Error = desiredStartPos - mean(p07);
206 currentElectricalPos = atan2(out0_frameCorrect(end),
       out1_frameCorrect(end)) + p07Error;
207
208 tVec_Spin2 = (currentElectricalPos + linspace(0,9*2*pi,
       spinPoints))';
209 out0_Spin2 = startCurrent*sin(tVec_Spin2);
210 out1_Spin2 = startCurrent*cos(tVec_Spin2);
211
212 queueOutputData(s,[out0_Spin2 out1_Spin2]);
213 [captured_data_Spin2, time_dat_Spin2] = s.
       startForeground();
214 posData1 = [mod(linspace(mean(p07),mean(p07)+2*pi,
       spinPoints),2*pi)]';
215 datData1 = captured_data_Spin2(:,5);
216
217 [maxMeasH07Voltage, maxMeasH07VoltageInd] = max(
       smoothDat1);
218 maxMeasH07VoltagePos = posData1(maxMeasH07VoltageInd);
```

```
[minMeasH07Voltage, minMeasH07VoltageInd] = min(
   smoothDat1);
minMeasH07VoltagePos = posData1(minMeasH07VoltageInd);
maxMaxDist = mod(maxH07VoltagePos -
   maxMeasH07VoltagePos,2*pi);
currentRef = maxMeasH07VoltagePos;
currentPos = captured_data_Spin2(end,5)*coeffsPs6(
   posStartSegment,1) + coeffsPs6(posStartSegment,2);
currentPos12 = captured_data_Spin2(end,6)*coeffsPs7(
   posStartSegment,1) + coeffsPs7(posStartSegment,2);
currentPos02 = captured_data_Spin2(end,7)*coeffsPs8(
   posStartSegment,1) + coeffsPs8(posStartSegment,2);

figure(3);
hold on;
for i = 1:lineSegs
    plotted26 = plot(p6Bounds(i,:),v6Bounds(i,:),':y','
       MarkerSize',4,'LineWidth',1.5);
    plotted27 = plot(p7Bounds(i,:),v7Bounds(i,:),':m','
       MarkerSize',4,'LineWidth',1.5);
    plotted28 = plot(p8Bounds(i,:),v8Bounds(i,:),':c','
       MarkerSize',4,'LineWidth',1.5);
end
h1 = plot(maxH07VoltagePos,maxH07Voltage,'*');
h2 = plot(minH07VoltagePos,minH07Voltage,'*');
h3 = plot(maxMeasH07VoltagePos,maxMeasH07Voltage,'^');
h4 = plot(minMeasH07VoltagePos,minMeasH07Voltage,'^');
h5 = plot([currentPos,currentPos],[0.5,4.5]);
% plot(posData1,datData1,'.');
plot(currentPos12,captured_data_Spin2(end,6),'x','
   MarkerSize',10);
plot(currentPos02,captured_data_Spin2(end,7),'o','
   MarkerSize',10);
legend([h1,h3,h2,h4,h5],'Reference Max','Measured Max',
   'Reference Min','Measured Min','Current Pos');

desiredPos = desiredStartPos;
% currentPos = desiredStartPos;
```

```
s.IsContinuous = true;

allData = [];
allTime = [];
data = [];
time = [];
timeCounter = 0;
position = [];
outSaver = [];
posSegment = posStartSegment;
firstRun = 1;
lastTimeCounter = 0;
nowTimeCounter = 0;
desiredPosVec = [desiredStartPos];
desiredPosTimes = [0];

while (true)

    prompt = ['Input desired motor position in radians.
        Current position is ' num2str(currentPos) '
       radians. '];
    desiredPos = input(prompt);
    desiredPosVec = [desiredPosVec; desiredPos];
    desiredPosTimes = [desiredPosTimes; timeCounter];
    lastTimeCounter = nowTimeCounter;
    nowTimeCounter = timeCounter;
    KiCounter = 0;

    if (desiredPos <= 2*pi)
        delPos = desiredPos - currentPos;
        desiredElectricalPos = currentElectricalPos +
           delPos*9;
        if (delPos < 0)
           delPos = delPos + 2*pi;
        end
        delPoints = round(delPos/(2*pi)*spinPoints);
        delVec = linspace(currentElectricalPos,
           desiredElectricalPos,delPoints)';
```

```
280         currentElectricalPos = desiredElectricalPos;
281         delVec = mod(delVec,2*pi);
282         needPoints = scanQueueLimit - delPoints;
283         if (needPoints > 0)
284             delVec = [delVec; delVec(end)*ones(
                    needPoints,1)];
285         end
286         out0 = startCurrent*sin(delVec);
287         out1 = startCurrent*cos(delVec);
288 %         figure(4);
289 %         hold on;
290 %         plot(delVec/9,out0);
291         phase1Current = out0(end);
292         phase2Current = out1(end);
293         queueOutputData(s,[out0 out1]);
294         lowerLimit6 = desiredPos > p6Bounds(:,1);
295         upperLimit6 = desiredPos < p6Bounds(:,2);
296         posSegment = find(and(lowerLimit6,upperLimit6))
                ;
297     end
298
299     if (firstRun)
300         firstRun = 0;
301         s.NotifyWhenDataAvailableExceeds =
                scanQueueLimit;
302         counterAmount = s.
                NotifyWhenDataAvailableExceeds;
303         s.NotifyWhenScansQueuedBelow = 2*scanQueueLimit
                ;
304         lh1 = addlistener(s,'DataAvailable', @(src,
                event) updateEvent(src,event,s));
305         lh2 = addlistener(s,'DataRequired', @(src,event
                ) queueEvent(src,event,s));
306         if (posSegment == posStartSegment)
307             queueOutputData(s,[out0(end)*ones(2*
                    scanQueueLimit,1) out1(end)*ones(2*
                    scanQueueLimit,1)]);
308         end
```

```
309
310         if (desiredPos > 2*pi)
311             delete(lh1);
312             delete(lh2);
313             break;
314         end
315         s.startBackground();
316
317     else
318 %         openfig('LinearPlots.fig','reuse');
319 %         hold on;
320 %         plot(position(lastTimeCounter+1:
    nowTimeCounter,:), data(lastTimeCounter+1:
    nowTimeCounter,:));
321 %         hold off;
322     end
323
324     if (desiredPos > 2*pi)
325         delete(lh1);
326         delete(lh2);
327         stop(s);
328         break;
329     end
330
331     if (isempty(posSegment))
332         delete(lh1);
333         delete(lh2);
334         stop(s);
335         queueOutputData(s,zeros(scanQueueLimit,2));
336         s.startBackground();
337         error('Not inside linearized regions');
338     end
339 end
340
341 queueOutputData(s,zeros(sampleRate,2));
342 s.startBackground();
343
344 % outSaverDelIndices = find(outSaver(:,1) == 0);
```

```
345 % outSaver(outSaverDelIndices,:) = [];
346 figure(5);
347 plot(outSaver);
348 figure(6);
349 plot(time,mean(position,2));
350 hold on;
351 plot([time(1), time(end)],[desiredPosVec(2),
       desiredPosVec(2)]);
352 hold off;
353
354 % save('posCTRLResults.mat');
355 % save('globeVars.mat','data');
356 % for i = 1:length(globalVars)
357     % tempName = globalVars{i};
358     % save('globeVars.mat',char(tempName),'-append');
359 % end
360
361
362 % figure(3);
363 % hold on;
364 % plot(posData1,datData1);
365 % plot(posData1,smoothDat1);
366 % % plot(posData1(locs),pks,'.','MarkerSize',30);
367 % plot(posData1(dipLocs),dipPks,'.','MarkerSize',24);
368 %
369 % plot(posInRotRep6SmoothOrder,dataSSAdj6SmoothOrder);
370 % plot(posInRotRep6SmoothOrder,dataSSAdj6Smooth);
371 % % plot(posInRotRep6SmoothOrder(smoothLocs),smoothPks
       ,'.','MarkerSize',30);
372 % plot(posInRotRep6SmoothOrder(smoothDipLocs),
       smoothDipPks,'.','MarkerSize',24);
373 %
374 % xlabel('Motor Position (rad)');
375 % ylabel('Hall-Effect Response (Volt)');
376 % title(['Critical Points of Current Position vs
       Absolute Position']);
377 % legend('Measured Reference Frame','Smoothed Measured
       Data','Measured Frame Peaks',...
```

```
%     'Absolute Reference Frame','Smoothed Reference
   Data','Absolute Frame Peaks');

% for i = 1:lineSegs
%
%     x16 = (-maxV6 - ps6(i,2))/ps6(i,1);
%     x26 = (maxV6 - ps6(i,2))/ps6(i,1);
%     x17 = (-maxV7 - ps7(i,2))/ps7(i,1);
%     x27 = (maxV7 - ps7(i,2))/ps7(i,1);
%     x18 = (-maxV8 - ps8(i,2))/ps8(i,1);
%     x28 = (maxV8 - ps8(i,2))/ps8(i,1);
%
%     plotted26 = plot([x16,x26],[-maxV6,maxV6] +
   meanDataSS(1),':y','MarkerSize',4,'LineWidth',1.5);
%     plotted27 = plot([x17,x27],[-maxV7,maxV7] +
   meanDataSS(2),':m','MarkerSize',4,'LineWidth',1.5);
%     plotted28 = plot([x18,x28],[-maxV8,maxV8] +
   meanDataSS(3),':c','MarkerSize',4,'LineWidth',1.5);
% end

hold off;
```

A.9.2 Data Collector

```
function updateEvent(src, event, s)

%     persistent tempData;
%     persistent tempTime;
    global data;
    global time;
    global position;
    global currentPos;
    global posSegment;
    global scanQueueLimit;
    global coeffsPs6;
    global coeffsPs7;
    global coeffsPs8;
    global meanValues;
    global phase1Current;
```

```
16      global phase2Current;
17      global dataSize;
18      global P1H02cToV;
19      global P1H12cToV;
20      global P2H07cToV;
21      global timeCounter;
22      global counterAmount;
23
24      % data: aI0+,aI0-,aI1+,aI1-,aIH07,aIH12,aIH02
25      tempData = event.Data;
26      tempTime = event.TimeStamps;
27      timeCounter = timeCounter + counterAmount;
28      v07 = tempData(:,5) - phase2Current*P2H07cToV;
29      v12 = tempData(:,6) - phase1Current*P1H12cToV;
30      v02 = tempData(:,7) - phase1Current*P1H02cToV;
31      tempData(:,5:7) = [v07,v12,v02];
32
33      data(timeCounter-counterAmount+1:timeCounter,:) =
           tempData(:,5:7);
34      time(timeCounter-counterAmount+1:timeCounter) =
           tempTime;
35  %     data = tempData;
36  %     time = tempTime;
37
38  %     p07 = (v07-coeffsPs6(posSegment,2) + meanValues
    (1))./coeffsPs6(posSegment,1);
39  %     p12 = (v12-coeffsPs7(posSegment,2) + meanValues
    (2))./coeffsPs7(posSegment,1);
40  %     p02 = (v02-coeffsPs8(posSegment,2) + meanValues
    (3))./coeffsPs8(posSegment,1);
41
42      p07 = v07*coeffsPs6(posSegment,1) + coeffsPs6(
           posSegment,2);
43      p12 = v12*coeffsPs7(posSegment,1) + coeffsPs7(
           posSegment,2);
44      p02 = v02*coeffsPs8(posSegment,1) + coeffsPs8(
           posSegment,2);
45      tempPosition = [p07, p12, p02];
```

```
46      position(timeCounter-counterAmount+1:timeCounter,:)
            = tempPosition;
47
48  %     currentPos = mean(p07(scanQueueLimit-10:end));
49  %     currentPos = (p07 + p12 + p02)./3;
50  %     disp([mean(p07), mean(p12), mean(p02)]);
51  %     queueOutputData(s,[zeros(scanQueueLimit,2) ]);
52
53  end
```

A.9.3 Output Queuer

```
1   function queueEvent(src, event, s)
2
3       global scanQueueLimit;
4       global currentPos;
5       global desiredPos;
6       global phase1Current;
7       global phase2Current;
8       global maxCurrent;
9       global position;
10      global timeCounter;
11      global counterAmount;
12      global spinPoints;
13      global outSaver;
14      global currentElectricalPos;
15      global desiredElectricalPos;
16      global time;
17
18      global Kp;
19      global Ki;
20      global Kd;
21      global Ksat;
22      global KiSat;
23      global KiCounter;
24      global lastError;
25      global lastTime;
26
```

```
27      currentPos = mean(mean(position(timeCounter-
            counterAmount+1:timeCounter,:)));
28      delPos = desiredPos - currentPos;
29  %     desiredElectricalPos = currentElectricalPos + 9*
        delPos;
30  %     KiCounter = 0;
31      KiCounter = KiCounter + delPos;
32  %     if (abs(KiCounter)*Ki > KiSat)
33  %        KiCounter = sign(KiCounter)*KiSat./Ki;
34  %     end
35
36  %     if (abs(Kp*delPos) < abs(1*Ki*KiCounter))
37  %        KiCounter =  sign(KiCounter)*Kp*abs(delPos)
        ./(1*Ki);
38  %     end
39
40      Ktot = Kp*delPos + Ki*KiCounter + Kd*(delPos -
            lastError)./(time(end) - lastTime);
41      lastTime = time(end);
42      lastError = delPos;
43
44  %     if (abs(Ktot) > Ksat)
45  %        Ktot = sign(Ktot)*Ksat;
46  %     end
47      desiredElectricalPos = currentElectricalPos + Ktot
            /9;
48      disp([position(timeCounter,:),currentPos,currentPos
             + Ktot, time(end)]);
49
50      tVec = [linspace(currentElectricalPos,
            desiredElectricalPos,2*scanQueueLimit)]';
51  %     tVec = [linspace((currentPos + Ktot)*9,(
        currentPos + Ktot)*9,2*scanQueueLimit)]';
52      currentElectricalPos = desiredElectricalPos;
53  %     needPoints = scanQueueLimit - delPoints;
54  %     if (needPoints > 0)
55  %         if (length(tVec) > 0)
```

```
%             tVec = [tVec; tVec(end)*ones(needPoints
   ,1)];
%         else
%             tVec = [tVec(end)*ones(needPoints,1)];
%         end
%     end

    out0 = maxCurrent*sin(tVec);
    out1 = maxCurrent*cos(tVec);
    outSaver(timeCounter-2*scanQueueLimit+1:timeCounter
       ,:) = [out0 out1];
    phase1Current = out0(end);
    phase2Current = out1(end);
    queueOutputData(s,[out0 out1]);
%     queueOutputData(s,[zeros(scanQueueLimit,2) ]);

end
```

A.9.4 Position Data Plotting

```
load('singlestep.mat');
close all;

timeVec = linspace(0,time(end),double(timeCounter));
outDiam = 0.133;

dPos = desiredPosVec(2);
lowerDPosBounds = dPos > allBounds(:,1);
upperDPosBounds = dPos < allBounds(:,2);
regionBounds = find(and(lowerDPosBounds,upperDPosBounds
   ));
avgPos = mean(position,2);

startEvalTime = 35;
startEvalIndex = startEvalTime*sampleRate;
evalPosVector = avgPos(startEvalIndex:end);

evalDelInd = find(evalPosVector > mean(evalPosVector) +
    5*std(evalPosVector));
```

```
timeVecEval = timeVec(startEvalIndex:end);
timeVecEval(evalDelInd) = [];
evalPosVector(evalDelInd) = [];

rmseEvalVec = ((dPos - evalPosVector).^2);
rmseEval = sqrt(sum(rmseEvalVec)./length(evalPosVector));
cumSumLenVec = (1:length(evalPosVector))';
rmseCumSum = sqrt(cumsum(rmseEvalVec)./cumSumLenVec);
degreeError = rmseCumSum*180/pi;
circumError = outDiam*pi*degreeError/360;

outTimeVec = (linspace(0,length(outSaver)./sampleRate,length(outSaver)))';
phaseAngle = atan2(outSaver(:,1),outSaver(:,2));

figure(1);
h0 = plot(timeVec,avgPos);
hold on;
timeLimits = [timeVec(1) timeVec(end)];
h1 = plot(timeLimits,[dPos, dPos],'LineWidth',1);
h2 = plot(timeLimits,allBounds(regionBounds,1)*ones(1,2),'--');
h3 = plot(timeLimits,allBounds(regionBounds,2)*ones(1,2),'--');
h4 = plot(timeVecEval,evalPosVector);
legend([h0, h1, h2, h3, h4],'Recorded Position','Desired Position',...
    'Segment Upper Boundary','Segment Lower Boundary','Evaluated Segment');
ylabel('Position (rad)');
xlabel('Time (sec)');
title('Position over Time');
hold off;

figure(2);
yyaxis left
plot(outTimeVec,outSaver(:,1),'-');
```

```
hold on;
plot(outTimeVec,outSaver(:,2),'--');
ylabel('Current (A)');
yyaxis right
plot(outTimeVec,phaseAngle,':');
ylabel('Phase Angle (rad)');
hold off;
legend('Phase One Current','Phase Two Current','Phase
   Angle');
xlabel('Time (sec)');
title('Currents and Associated Phase Angle');

figure(3);
plot(timeVecEval,degreeError);
xlabel('Time (sec)');
ylabel('RMSE (deg)');
title(['RMSE Between ',num2str(startEvalTime),' and X
   Seconds']);
```

APPENDIX B

SOLIDWORKS DRAWINGS AND COMPONENT LABELS

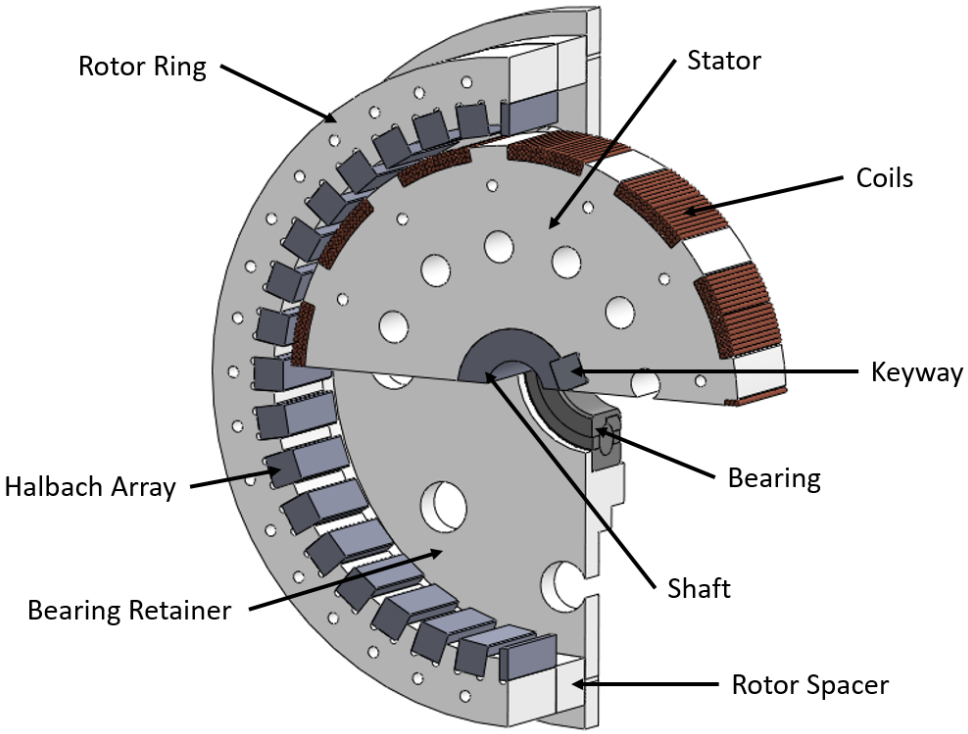


Figure B.1: Inner motor components.

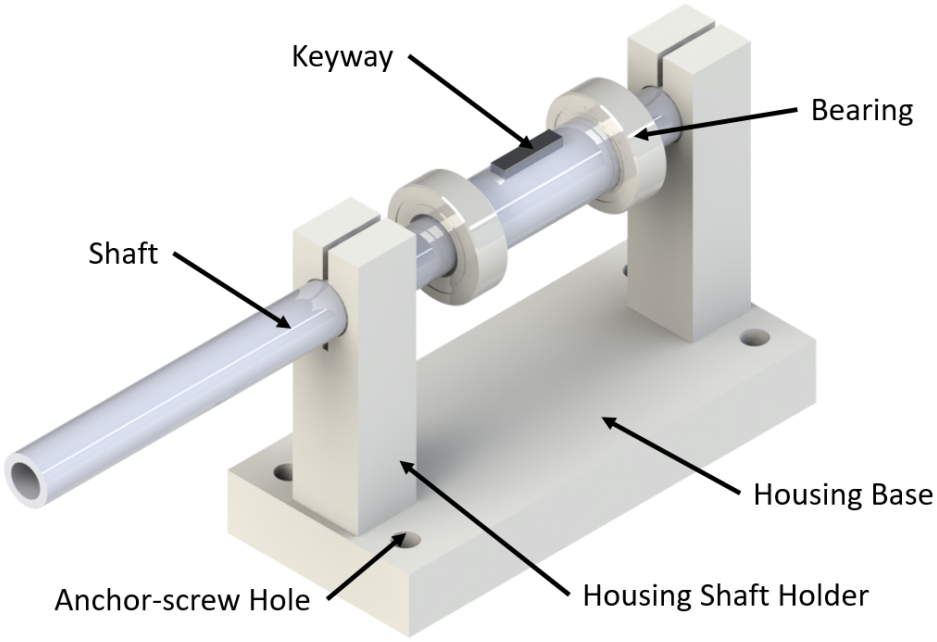
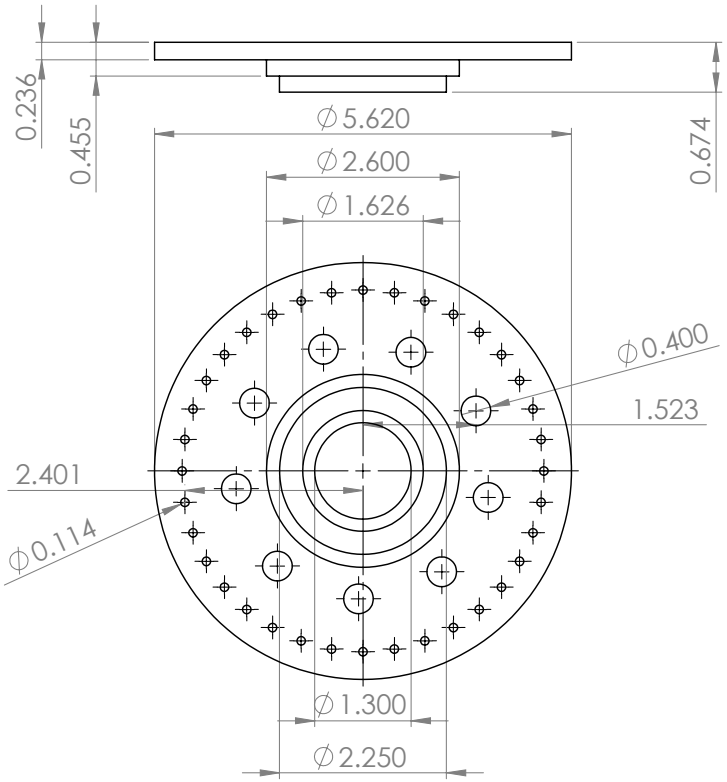
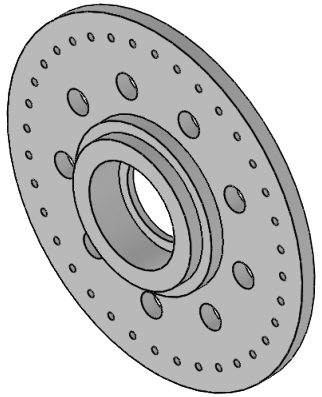


Figure B.2: Motor infrastructure components.



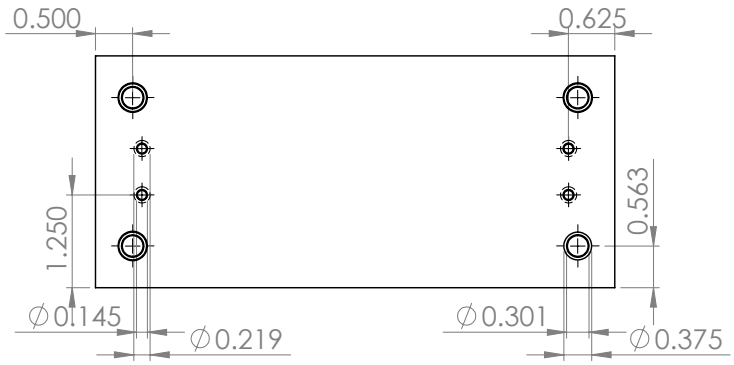


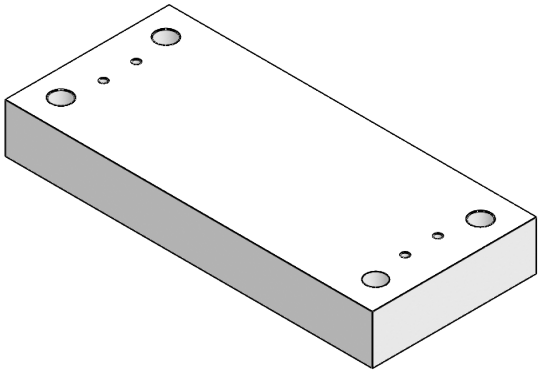
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

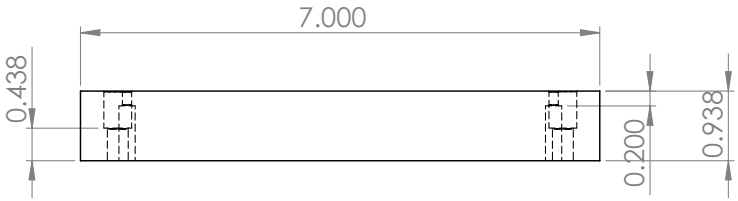
TITLE:

Bearing Retainer

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1





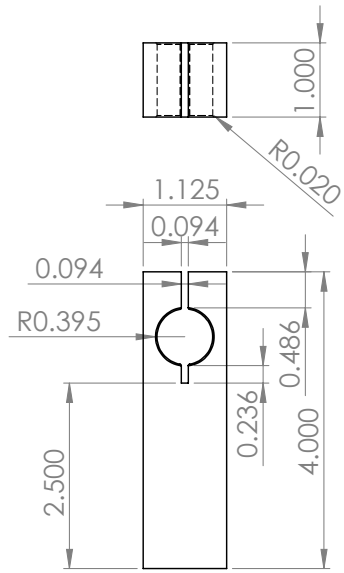


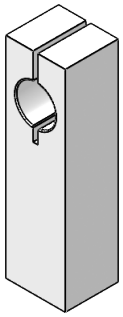
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

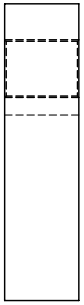
TITLE:

Housing Base

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1





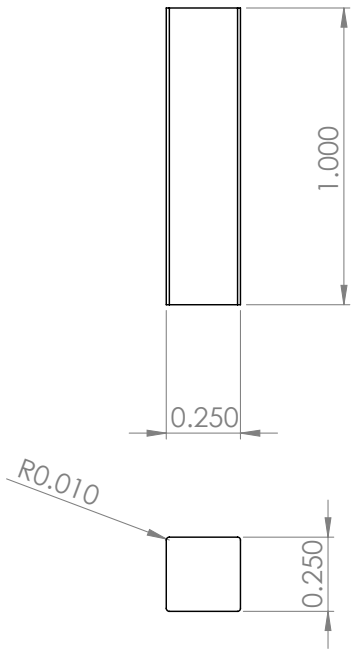


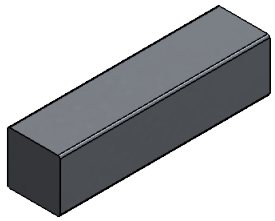
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

Housing Shaft Holder

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1





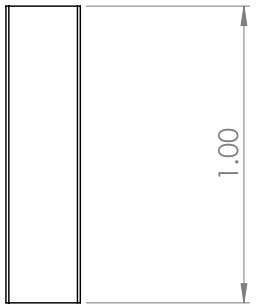
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

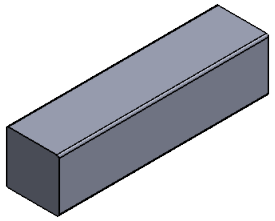
TITLE:

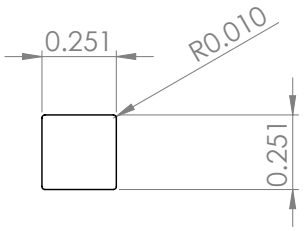
Keyway

SIZE	DWG. NO.	REV
A		

SCALE: 2:1	WEIGHT:	SHEET 1 OF 1







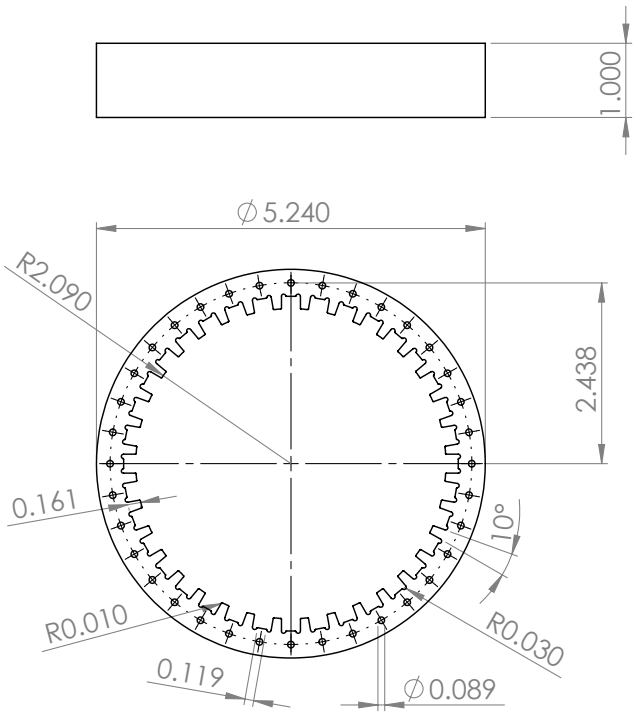
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

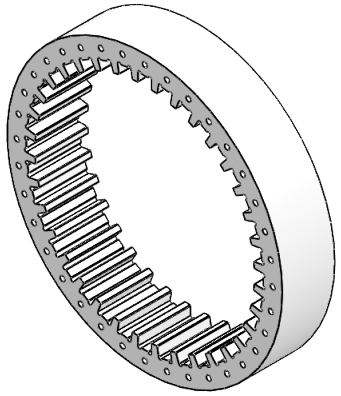
TITLE:

Magnet Bar

SIZE	DWG. NO.	REV
A		

SCALE: 2:1	WEIGHT:	SHEET 1 OF 1





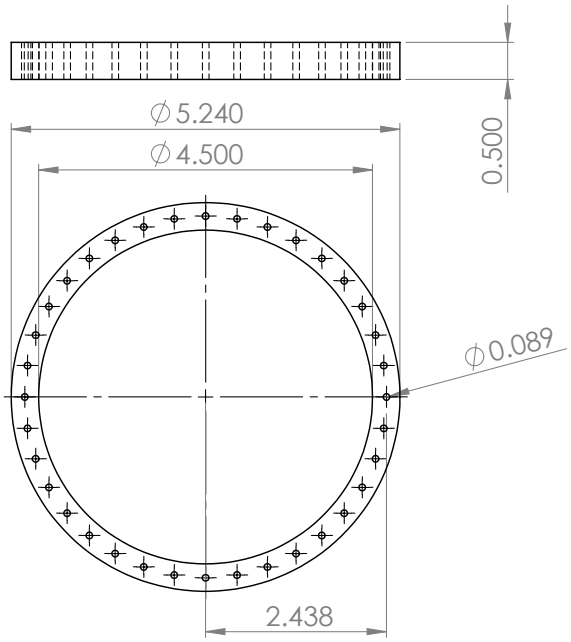
All holes are thru holes

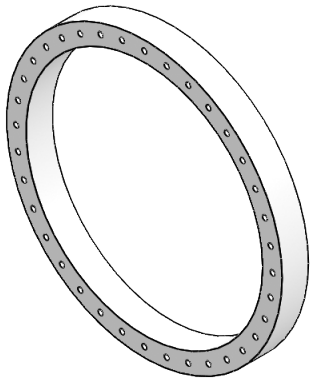
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL ABS	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

Rotor Ring

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1



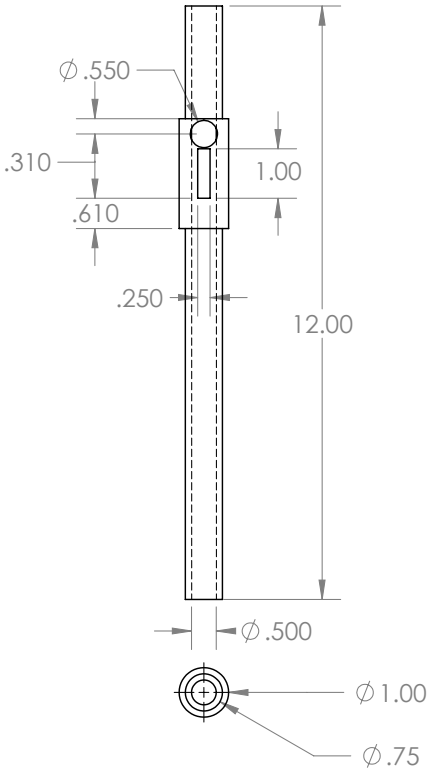


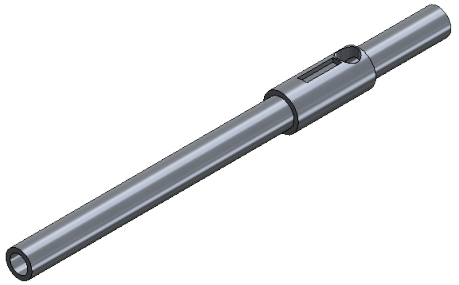
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

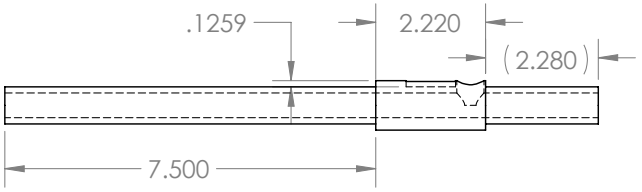
TITLE:

Rotor Spacer

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1





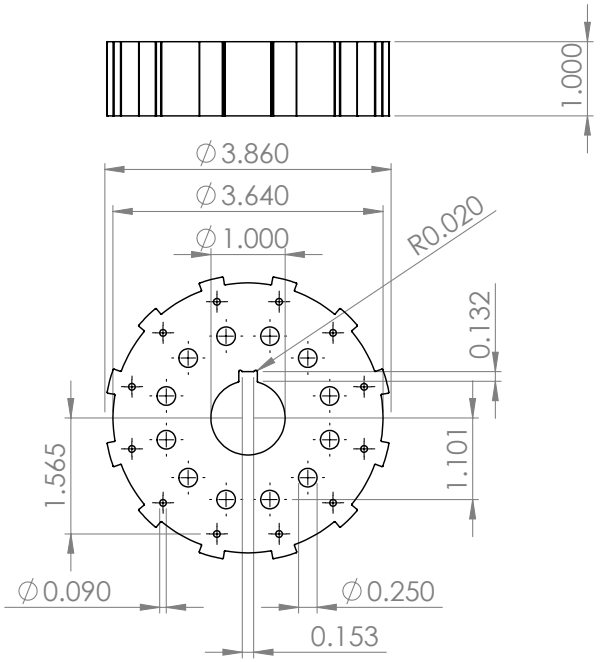


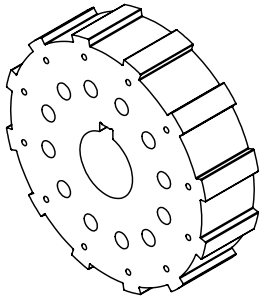
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

Shaft

SIZE	DWG. NO.	REV
A		
SCALE: 1:4	WEIGHT:	SHEET 1 OF 1





All holes are thru holes

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES DIMENSIONS ARE TYPICAL	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

Stator

SIZE	DWG. NO.	REV
A		
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1