An Open-Loop Method for Reduction of Torque Ripple and An Associated Thermal-Management Technique

by

Arthur Joseph Kalb

S.B., Massachusetts Institute of Technology (1991)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 1997

Author .....
Department of Electrical Engineering and Computer Science
November 13, 1996

Certified by .....
Jeffrey H. Lang
Professor
Electrical Engineering and Computer Science
Thesis Co-supervisor

Certified by .....
Stephen D. Umans
Principal Research Engineer
Electrical Engineering and Computer Science
Thesis Co-supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

An Open-Loop Method for Reduction of Torque Ripple and An Associated Thermal-Management Technique

by

Arthur Joseph Kalb

Submitted to the Department of Electrical Engineering and Computer Science
on November 13, 1996, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

ABSTRACT

This thesis studies torque-ripple reduction in an axial-flux nine-pole-pair doubly-wound synchronous motor with three phases on the rotor and stator. Torque ripple is measured as a function of the magnitude of the three-phase rotor and stator currents with a fixed electrical angle between them. This thesis then develops an open-loop technique that uses the measured data to generate a modulation of the magnitudes of the currents as a function of position. It is demonstrated that this modulation significantly reduces torque ripple. This thesis also develops a method which reduces peak temperatures in the motor while still providing constant torque at a fixed position. This thermal-management technique can be unified with the ripple-reduction technique with minimal effort.

Thesis Co-supervisor: Jeffrey H. Lang
Title: Professor, Electrical Engineering and Computer Science

Thesis Co-supervisor: Stephen D. Umans
Title: Principal Research Engineer, Electrical Engineering and Computer Science

Acknowledgments

I thank my advisors, Jeffrey Lang and Steve Umans, for their support, advice, and patience. Special consideration and thanks go to my colleague Deron Jackson for supplying data after my physical departure from MIT. I would also like to thank him for providing me with Figures 1-1 and 5-1. I thank John Hollerbach at McGill University for funding this research, and National Semiconductor Corporation for their financial support of my studies. I would like to express my appreciation to my friends and colleagues for their friendship and to my parents for their incalculable generosity and love. Most of all, I would like to thank my wife, Margaret, for her loving companionship, and my children for the joy they bring into my life. I dedicate this thesis to her and to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 An Overview of the Problem

The continuing development of direct-drive robots poses new challenges for the design and control of motors. The direct-drive approach to robotics directly couples the torque generating motor to a mechanical assembly, or linkage, without the use of reducers. The removal of gearing eliminates backlash and friction, allowing for more accurate positioning of the linkage. It also simplifies the dynamics of the arm, making them more predictable and thus more suitable to modern and classical control methods.

There are drawbacks to the direct-drive approach. Any variation in motor torque now couples directly to the load. Thus, to obtain accurate forces at the end of the linkage, that is, compliance and force control, motors with low torque ripple must be utilized. At the same time, because these motors are operating without reducers, they must generate large torques at low velocity. Compounding the problem further, these motors are often situated on a distal link thereby presenting a load to the linkage causing a reduction of positioning accuracy due to deflection, as well as a degradation of dynamic response as a result of increased inertia. Minimizing the impact of this requires the use of smaller, lighter-weight motors which generally runs contrary to the demands for large torque output with low ripple. The only exception would be for motors that use large currents, in which case thermal dissipation becomes a

limitation. Thus, direct-drive robotics requires motors capable of delivering a large torque output at low ripple without increasing weight or thermal dissipation. All these design considerations tradeoff against each other.

The thermal dissipation issue merits greater consideration. As mentioned above, the combined demands on the motor to generate large torques with little weight require that motors for direct-drive applications make use of large currents. These large currents can cause overheating. This problem is compounded in situations where the motor is enclosed or when the motor is used to hold a static load. In the case of a static load, the motor must produce a constant torque. This normally implies that the windings are fed constant currents, instead of time-varying sinusoids of current. In this way, some winding is dissipating higher average power than it would if the motor were rotating, creating the possibility for localized hot spots in the motor. Since localized hot spots above the motor's temperature specification are unacceptable, the temperature at the hot spots establishes the upper bound on torque output.

This thesis addresses the tradeoffs inherent in direct-drive motor design by presenting control techniques to minimize torque ripple and localized temperature peaks. These techniques do not compromise the demands for high torque over all velocities, accurate positioning, and fast dynamic response. They do provide an enhancement in compliance and force control as well as increases in torque output and motor reliability by reducing thermal stress. Furthermore, the two techniques can be merged into one without significant additional effort.

1.2 Specific Statement of the Problem

There are many options when it comes to implementing a direct-drive system. There are options in both the linkage construction and in the motors used to drive the linkage. Asada and Youcef-Toumi [4] give an excellent introduction to linkage mechanisms in direct-drive robots as well as a presentation of some of their innovative work. However, linkage mechanisms are not a subject of this research and attention is devoted solely to the improvement of motor performance.

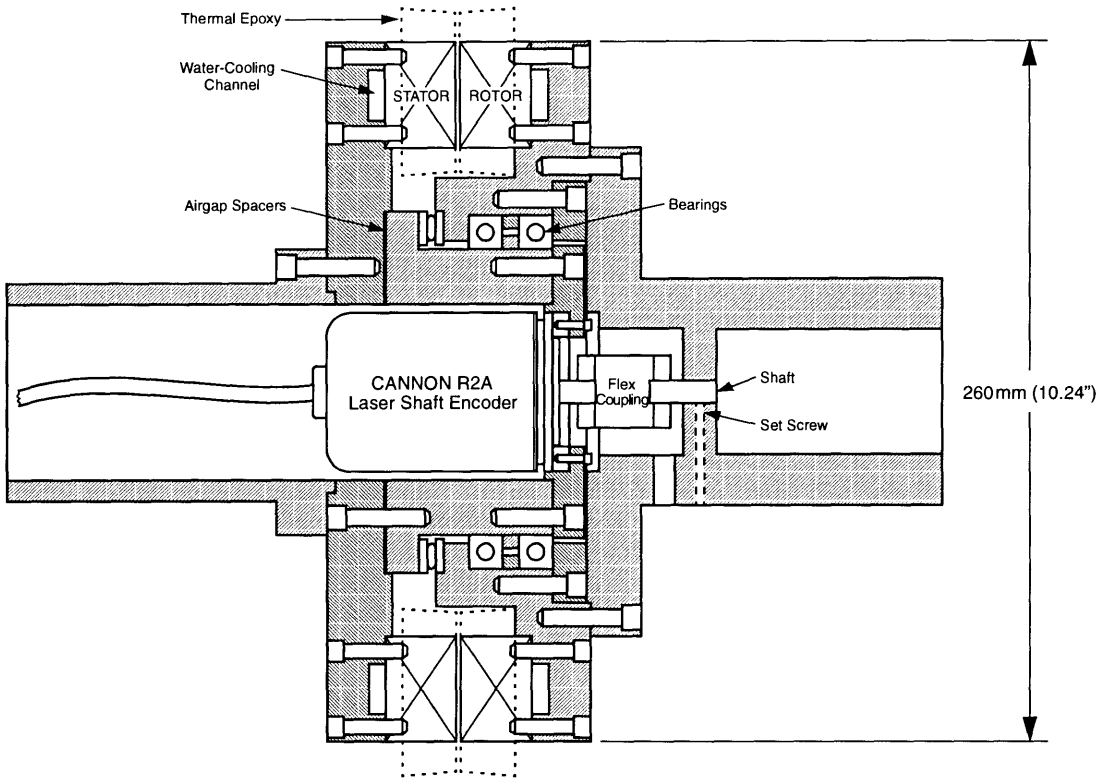

Figure 1-1: A Cross-section of the Axial Flux Motor.

Previously, brushless DC motors (BLDCM) and variable-reluctance motors (VRM) have been used for direct-drive robot applications. The motor used for research in this thesis is neither a BLDCM nor a VRM. Instead, a synchronous axial-air-gap motor has been chosen, owing to its large torque output per unit mass. Figure 1-1 depicts the mechanical construction of the motor. The machine used has a symmetrical construction with the rotor and stator identically machined and wound. There are nine pole-pairs on both the rotor and stator with three wye-connected electrical phases per side. Windings are formed from stranded 12-gauge insulated wire distributed in a 1-2-1-0-0-0-$\overline{1}$-$\overline{2}$-$\overline{1}$-0-0-0 fashion for a total of 108 slots per side. Slots are skewed to minimize slot ripple. Terminal blocks, attached to the rotor and stator, are employed to make the connection between the windings and an external driving source. Cooling is achieved by circulating water in a channel in the back-iron of the each motor half. To supply the flow of water, cooling lines must be attached to both the rotor and the stator. Another interesting element of the motor design is that a rotary encoder is

embedded in the stator side with a coupling to the rotor shaft. The encoder provides a measurement of position which is crucial to the success of the techniques described in this thesis. The motor is cylindrical with an outer radius of 260mm and a depth of 54mm. For this thesis, the adjustable air-gap has been set at 51.2 mils. The motor is intended to generate approximately 100 Nm of torque, provided the gap is set appropriately. However, this torque level has not been achieved as a result of thermal constraints and the relatively large gap setting.

The objective of this thesis is to create an open-loop technique for torque-ripple reduction and a technique for thermal management. The ripple-reduction technique is allowed one calibration or characterization, mimicking a manufacturing calibration. The reason for doing the ripple correction on an open-loop basis is simple. Accurate torque sensors are expensive, and do not fit the form factor of the arm assembly. Also, torque control loops complicate the dynamics of the system, if not making it unstable, and they at least impose design constraints on the system. A thermal-management technique to avoid the formation of hot spots is also examined. It also has become possible to merge the two techniques into one. In this thesis, the techniques are tested experimentally. On the path to a working system, many practical issues have surfaced and are discussed in this document. It is recommended that future systems consider these matters up front.

1.3 An Overview of the Solution

This thesis first develops a torque-ripple-reduction technique. In the region of operation for this thesis, the magnetics of the motor are assumed to behave linearly. This assumption may become invalid if the magnetics are saturated in the attempt to maximize torque output. Future research can be done to investigate the impact of saturation on the effectiveness of the technique. For this thesis, the assumption enables the torque to be expressed as a function of position for fixed phase currents. Furthermore, the torque scales quadratically with a linear scaling of the currents in the motor. The position dependence can be deduced by characterizing the motor

using known currents in the windings. In this thesis, the characterization has been done using equal amplitude three-phase sinusoids of current in both the rotor and the stator windings, maintaining a constant electrical angle between them. As the rotor position varies, the three-phase currents rotate to maintain constant torque. However, the reality is that using three-phase sinusoids generates a torque profile with considerable torque variation versus position, or torque ripple. It is shown in this thesis that if the three-phase currents are amplitude modulated versus position, then the torque ripple can be reduced by a factor of three. The amount of amplitude modulation at a position is proportional to the square root of the ratio of the desired torque to the characterized torque at that position.

This thesis then explores the heating in the motor. A model of the temperature changes is created. Then, based on the model and utilizing the unique construction of the motor, a technique for eliminating temperature peaks is proposed. Since the motor in this thesis has both a three-phase rotor and stator, it is possible to constantly rotate both sets of currents so that the phase angle between them remains constant. This provides a nearly constant torque, while at the same time spreading the dissipation uniformly among the windings. Experiments have been conducted to verify the performance of the technique and the results show the predicted improvement. However, the gains are extremely slight for this motor.

The final aspect explored in this thesis is the unification of the ripple-reduction technique with the thermal-management technique. This is successfully accomplished and data is reported. The combination of the techniques does not compromise the benefits achieved by each technique in isolation from the other.

1.4 Organization

This thesis has two goals: to produce a torque-ripple-reduction technique, and to create a thermal-management method. This document presents the research in the following manner. Chapter 2 presents an overview of previous research efforts in torque-ripple reduction. Chapter 3 produces a model that is used to study torque

and torque ripple. Using this model, the chapter derives the governing equations for the ripple-reduction technique. Chapter 4 explains the implementation of the ripple-reduction technique, documents the data collection method and provides data supporting the effectiveness of the ripple-reduction technique. Chapter 5 formulates a model of temperature changes in this motor, presents the thermal-management method and offers data supporting the success of the method. The chapter also combines the ripple-reduction technique with the thermal-management method and provides data supporting the combined technique's effectiveness. Chapter 6 concludes the main body of the document with an analysis of the strengths and weaknesses of the present work and suggested ways to improve upon the weaknesses.

Chapter 2

Background

2.1 Introduction

A survey of the literature on torque-ripple reduction reveals that efforts have focused on four approaches. One approach to the problem is to change the characteristics of the motor. This can be valuable when used in conjunction with control techniques, thereby removing some of the burden from the control system. As indicated in the introductory chapter however, there are situations where it is desirable to remove the performance burden from the motor and shift it to the control system. Nonetheless, more research effort in the future can go into investigating the engineering tradeoffs between optimizing the motor and using a more sophisticated control system.

The last three approaches to ripple reduction are control methods. An obvious approach is the use of classical feedback techniques. A more interesting approach is the use of indirect feedback methods, which employ observers or predictors to calculate feedback quantities, like flux or torque, based on easily measured quantities, such as phase current and position. Finally, there exist open-loop control techniques similar to the present work. The following sections provide an overview of previous works which use the various approaches.

2.2 Reducing Ripple through Motor Design

One method of ripple reduction is to design the motor in a way that minimizes torque ripple. Yet, the minimization of ripple invariably trades off with maximum torque production. All the design techniques revolve around shaping the flux distributions so as to minimize ripple. Techniques such as distributed windings, pitch shortening, multi-phase windings, multi-star windings, pole-arc optimization, and slot skewing have been developed to accomplish this. A paper by Nogarede and Lajoie-Mazenc [13] reviews most of these techniques. Slot skewing has also been discussed in many papers, and a paper by Kim, Sim and Won [9] provides a reasonable discussion of the topic. The two papers named above are a sampling of the available literature on these issues.

2.2.1 The Work of Nogarede and Lajoie-Mazenc

Distributed Windings

The elementary model of a motor calls for each winding to create a rectangular magnetomotive force (mmf) distribution. Doing this creates space harmonics of mmf and thus harmonics of flux. These flux harmonics cause torque ripple. Distributing a winding over several teeth gives the designer the flexibility to shape the mmf to be approximately sinusoidal, thereby making the mmf more sinusoidal in space. This technique is also called "conductor distribution" [13]. A textbook by Fitzgerald, Kingsley and Umans [5, pp. 554–568] gives the elementary mathematics of distributed windings.

The limitation to this technique comes in that the conductors require some amount of space. As the number of slots for the conductors increases, there is less surface area for teeth. This has two implications. First, it ultimately limits the number of conductors on the motor and correspondingly the ability to shape a sinusoidal mmf. Second, the reduction in surface area for the magnetic paths increases leakage and thereby decreases motor efficiency.

Pole-Arc Optimization

Nogarede and Lajoie-Mazenc [13] describe a technique for reducing ripple whereby the angular widths of the poles are reduced. Although their work was done on a BLDCM, the conclusions they arrive at are applicable to other synchronous motors. They assert that by selecting the appropriate arc width the particularly prominent ripple harmonic can be nullified. It should be emphasized that the technique is limited to reducing one harmonic. They also report that this technique reduces the mmf fundamental amplitude and thus the average torque.

Pitch Shortening

Another fairly basic idea reported by Nogarede and Lajoie-Mazenc is pitch shortening [13]. There is a contemporary introduction to pitch shortening in the book by Fitzgerald, Kingsley and Umans [5, pp. 554—568]. When fractional pitch windings are created, the fundamental of the mmf wave is reduced, but the harmonics are generally reduced by a proportionally larger amount. This reduces torque but reduces torque ripple by an even larger factor. Nogarede and Lajoie-Mazenc report that by appropriately selecting the pitch factor, it is possible to nullify two space harmonics of the mmf.

Multi-Phase Windings

The use of multi-phase windings can also decrease torque ripple [13]. The increase in the number of phases reportedly increases the frequency of the ripple harmonics which more than proportionally reduces the torque ripple. It can be visualized that the more phases there are to work with the more able the designer is to shape the flux sinusoidally.

Multi-Star Windings

The technique of using multi-star windings [13] also reduces torque ripple. In this method, multi-phase windings are repeated, possibly several times, with some angle

of spatial separation between them. If the separation angle is chosen correctly, some harmonics can be selectively eliminated. This technique has the drawback of using quite a bit of winding area while not maximizing the torque for given current levels and winding area.

2.2.2 The Work of Kim, Sim and Won

Slot Skewing and Magnet Skewing

Kim, Sim and Won [9] analyze the effect of slot skewing for a BLDCM. Typically, variations in permeance caused by slots cause torque ripple. The idea behind skewing is that the variation in permeance can be almost eliminated by skewing the slots by one slot pitch. This will provide nearly constant tooth area between stator and rotor poles as the motor rotates. A constant area maintains an almost constant permeance and thus constant torque. In reality, there are still some variations due to edge effects of the teeth and slots.

2.3 Direct Feedback Methods for Torque-Ripple Reduction

Classical and modern feedback techniques can be used to control torque. These techniques are widely used and well understood and provide good system performance. The problem with feedback techniques is that they require torque feedback which, in turn, requires a sensor. In general, sensors do not fit the form factor of the direct-drive approach. Furthermore, there is a tradeoff between the accuracy of the torque measurement and positioning accuracy. Asada and Youcef-Toumi [4] provide a good discussion of the problems of torque feedback control in direct-drive motors.

2.4 Indirect Feedback Methods

Indirect feedback is a method by which desired feedback values are computed from measurements of easily measured quantities. Typically either torque or flux is computed from position and phase-current measurements. The computed quantities are then compared to a reference value to generate an error signal which drives the system plant. The technique is useful in that it can compensate for non-linearities. The limitation comes in that there are loop dynamics associated with the feedback system which can limit system response. There is in addition the obvious limitation that the indirectly fed-back quantity depends on the modeling accuracy between it and the measured quantities.

2.4.1 The Work of Low, Tseng, Lee, Lim and Lock

Low, Tseng, Lee, Lim and Lock [10] present a model-based estimation of torque using current and position in a permanent-magnet motor. They attempt to compensate for variations in torque due to flux space harmonics but do not address the issue of slot ripple. The authors model the torque as a function of current and a spatially varying inductance. Using position and phase-current feedback, they calculate the value of inductance based on their model. This model does not address the variation of inductance with saturation. Using the estimated inductance, they compute the estimated torque and use this value in a feedback loop. They report graphical data showing significant ripple reduction. No quantitative data is given to indicate the success of their work.

2.4.2 Ilić-Spong, Miller, MacMinn and Thorp

Ilić-Spong, Miller, MacMinn and Thorp [6] present a control method for providing instantaneous torque control for a switched-reluctance motor. Their technique involves a coordinate transformation that makes the torque of a rotating motor appear as a time-independent linear function of the transformed inputs. Using models of inductance, they calculate what flux is required at each position for constant torque. From

the motor, they take measurements of the phase currents and use this information, combined with inductance models, to make estimations of the flux in the machine. A control loop then acts to cause the actual flux to equal, as estimated by output measurements, the desired flux, as calculated from inductance models. This theoretically should force the output torque equal to the desired value.

The advantage of the control structure presented in this paper is that the system is controlled as a multi-variable, linear, time-independent system. For the switched-reluctance motor they chose as an example, the coordinate transformation is quite simple. Furthermore, for a switched-reluctance motor, the multi-variable linear equations describing the system are decoupled, allowing them to be treated as a collection of single-variable equations.

This thesis uses a somewhat simpler approach, avoiding the complexity of the state estimation. The estimation is replaced by a one-time characterization and subsequent table lookup. Another limitation of the instantaneous torque control system is that it utilizes a feedback control system which limits the torque control response by the dynamics of the feedback loop. This thesis does not have a control loop and therefore provides a wide-bandwidth operation.

2.5 Open-Loop Control of Torque

Open-loop methods of torque control utilize a one-time calibration to achieve reduced torque ripple thereafter. Assuming that the motor torque is a function of position and phase currents, it should be possible to control the currents to minimize, and theoretically eliminate, torque ripple. This reduction in ripple comes without introducing new system dynamics, caused by a torque observer or feedback system. This approach depends on the modeling accuracy of torque as a function of position and currents.

2.5.1 Newman and Patel

Newman and Patel [12] describe a torque-ripple reduction algorithm for switched-reluctance motors in the AdeptOne robot. They point out that torque is a function of phase currents and position and that there is no unique inverse mapping from torque to phase currents. If one imposes an additional constraint of minimizing heating, they argue that at least one of the three phase currents, and maybe two, should be set to zero. After imposing this additional constraint, they are able to exhaustively map out torque versus position and versus one free variable in the domain of possible currents.

They generate the map using an iterative process. They start with some reasonable guess at what the current waveforms should look like. They then measure the torque ripple for these profiles. This data can then be used to modify the profiles. The modification is done as a scaling of the old profiles at each rotor position and torque level in such a way as to theoretically eliminate torque ripple.

This technique achieves a reduction of torque ripple from about 25% to 10%. It is not clear from the paper whether saturation is a contributing factor to the ripple.

2.5.2 Kamiya, Shigyo, Makino and Matsui

Kamiya, Shigyo, Makino and Matsui [8] present a ripple-reduction technique based on a phase and amplitude modulation of the three-phase currents of a BLDCM. It is not clear from the paper describing their work how they calculate the modulations, whether from empirical data or by some other means.

The authors explicitly address the issues of torque ripple caused by cogging and spatial variations in air-gap flux. Their work is presented in a transformed coordinate system, namely the DQ0 coordinate system. They first propose to add a component to the quadrature current in order to eliminate current-independent cogging torque. The modified quadrature current is then modulated to eliminate the ripple caused by flux space harmonics. This does not seem to be quite the right approach, as the term added to offset the cogging torque is modulated. Ideally, the offset term would

not be modulated. Nonetheless, Kamiya, Shigyo, Makino and Matsui report ripple reduction by approximately a factor of five, measured peak to peak.

For a fully synchronous motor as in the present research, there is almost no current-independent ripple (only that which is generated by remnant flux in the core). Thus, it is only necessary to eliminate the current-dependent portion of the ripple. The modulation of currents done in this thesis in some ways repeats the work of Kamiya, Shigyo, Makino and Matsui. However, some differences should be pointed out. The chief difference is that the formulation of the present work does not rely on a transformation. Thus, it is computationally simpler and involves fewer concerns for a real-time implementation. Also, this research did not use any phase modulation of the currents.

2.5.3 Matsui, Akao and Wakino

The paper by Matsui, Akao and Wakino [11] uses methods very similar to the paper by Kamiya, Shigyo, Makino and Matsui. The difference comes in that the latter uses a BLDCM and the former uses a VRM. Whereas the BLDCM has a current-independent ripple, the VRM has only current-dependent ripple.

Matsui, Akao and Wakino propose a control method based on vector-controlled induction motors. The description is made in DQ0 coordinates. Torque is expressed as the product of quadrature current with a constant, which is itself proportional to direct-axis current.

$$T = K\, i_q \tag{2.1}$$

$$K = (L_d - L_q)\, i_d \tag{2.2}$$

The authors decide to control quadrature current, leaving the direct-axis current fixed, since the quadrature-axis inductance is smaller than the direct-axis inductance and thus lends itself to faster motor response. The torque ripple can then be described as a sum of a position-dependent term and a term proportional to the product of a

position-dependent term and the quadrature current. The term that is independent of quadrature current arises from the direct-axis poles moving by slots and teeth. Equation 2.3 below expresses this formulation of torque ripple mathematically.

$$\Delta T = K_0 F_0(\theta) + K_1 i_q F_1(\theta) \tag{2.3}$$

The details of their technique are not extremely clear. It appears that they add a term to the quadrature current to cancel the position-dependent term $K_0 F_0(\theta)$ and then scale this in such a way that the current-dependent term, $K_1 i_q F_1(\theta)$, is cancelled. However, the formulas presenting their material are incoherent and several questions arise. Most notably, it is not evident how they produce any average torque. They do produce plots of torque ripple versus current both with and without their ripple-reduction method. They also provide photographs of torque waveforms plotted on an oscilloscope. The Matsui, Akao and Wakino paper shows a best-case ripple reduction by about a factor of four, peak to peak.

The Matsui, Akao and Wakino paper most closely resembles the present work. One area of similarity is the use of empirical data in the framework of an analytical model. However, points of distinction should be made. Their technique uses a transformed coordinate system, which adds to the complexity of the implementation. The technique used in this thesis does not use a transformation. Furthermore, the synchronous motor used in this thesis has three phases on the rotor and stator. Correspondingly, the control technique involves modulating both the rotor and stator phase currents.

Chapter 3

An Approach to Ripple Reduction

3.1 Introduction

This chapter presents the theory of operation for the ripple-reduction technique developed in this thesis. As highlighted earlier, this technique controls the torque without the use of feedback. Its application in this thesis is limited to that of a synchronous motor operating out of saturation. Jackson [7], however, has extended the technique to the saturation region using the same motor.

The basic premise of the technique, as applied in this thesis, is that when the magnetics are not saturated the torque will be a function of position, and will scale quadratically with a linear scaling of the currents in the motor. This relationship between the scaling of torque and the scaling of the currents permits a desired torque to be produced by appropriately selecting the magnitude of the phase currents, I_{mag}, as a function of rotor position, θ. Each phase current, $i_n(\theta)$, can then be written as the product of the magnitude, I_{mag}, and another waveform, $f_n(\theta)$. This can be written in vector notation as $\mathbf{i}(\theta) = I_{mag}\,\mathbf{f}(\theta)$, where the $i_n(\theta)$ and $f_n(\theta)$ make up the rows of $\mathbf{i}(\theta)$ and $\mathbf{f}(\theta)$ respectively.

For this thesis, the function $\mathbf{f}(\theta)$, which will be referred to as the waveform shape function, is chosen so that the rotor and stator currents are sinusoidal functions as given by Equations 3.1–3.6. The rotor and stator sinusoids have a relative phase angle, $\delta_S - \delta_R$, between them which is used to set the angle between the rotor and

stator magnetic poles, typically in such a way as to maximize torque output. As the rotor rotates, this angle changes, requiring the stator waveforms to have a position-dependent phase angle. Equations expressing the position-dependence of the waveforms are given in Equations 3.1–3.6. In the equations, the factor of nine provides the transformation from mechanical frequency to electrical frequency for the nine pole-pair motor. Figures 3-1 and 3-2, located below, graphically portray the waveform shapes used for this thesis. f_{ar}, f_{br}, and f_{cr} are the rotor current waveform shapes. f_{as}, f_{bs}, and f_{cs} are the stator current waveform shapes. It should be realized that the rotor currents depicted are constants. This is possible because it is only the relative phase angle between the rotor and stator currents which sets the torque, at least in an idealized model of the motor, and not the absolute angle of either set of currents. Thus, the rotor currents can be held constant, provided the stator currents have the proper phase angle.

$$f_{ar} = \cos(\delta_R) \tag{3.1}$$

$$f_{br} = \cos(\delta_R - \frac{2\pi}{3}) \tag{3.2}$$

$$f_{cr} = \cos(\delta_R - \frac{4\pi}{3}) \tag{3.3}$$

$$f_{as} = \cos(9\,\theta + \delta_S) \tag{3.4}$$

$$f_{bs} = \cos(9\,\theta + \delta_S - \frac{2\pi}{3}) \tag{3.5}$$

$$f_{cs} = \cos(9\,\theta + \delta_S - \frac{4\pi}{3}) \tag{3.6}$$

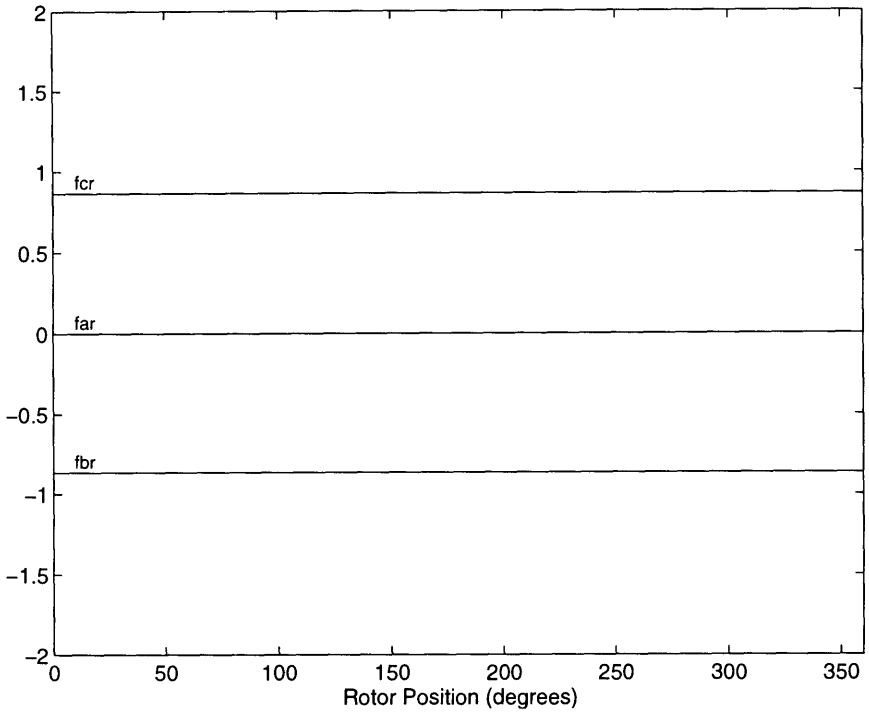

Figure 3-1: Waveform Shapes for the Rotor Currents.

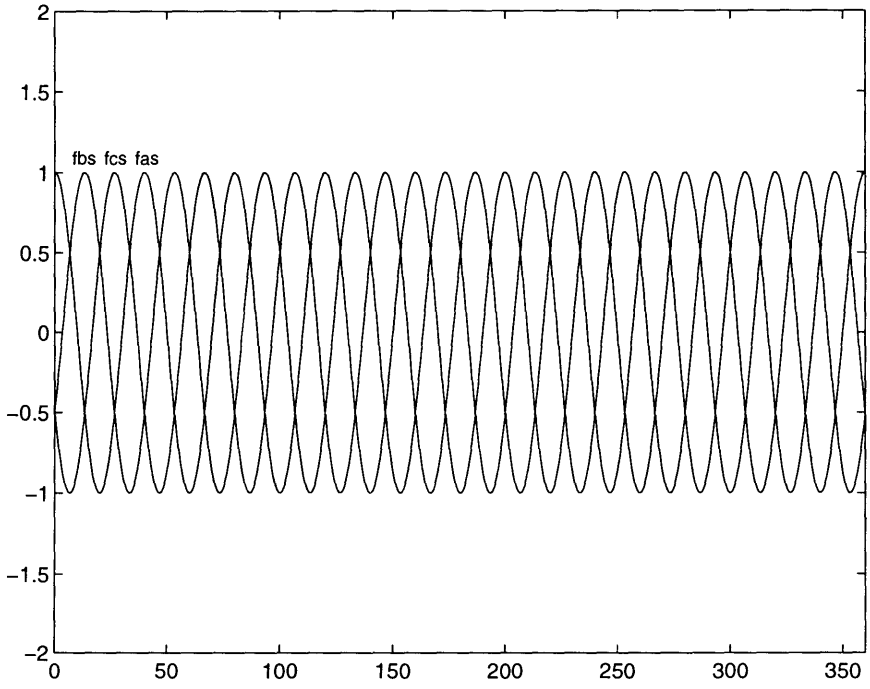

Figure 3-2: Waveform Shapes for the Stator Currents.

It is known that for the simplest models of machine behavior, the waveform shapes described above produce ripple-free torque [17][18], with the average torque being set by the magnitude of the current, I_{mag}, which should be a constant over position. In reality, there are non-idealities in the design and construction of the motor which create torque ripple. This thesis compensates for the torque ripple by modulating the magnitude, I_{mag}, as a non-constant function over position, $\tilde{I}_{mag}(\theta)$. This produces a scaling of the output torque, which, when done properly, reduces the torque ripple. In order to calculate the proper amount of modulation, this chapter proceeds by first deriving the relationship between the current magnitude and the output torque, expressed mathematically in Equation 3.7. In this equation, $\mathbf{L}(\theta)$ is the position-dependent inductance matrix for the windings. It is clear from the equation that the torque is proportional to the square of the magnitude of the current. After deriving this result, this chapter then derives the "correction factor" by which to modulate the magnitude of the current, in order to reduce the torque ripple. The expression for the correction factor is given in Equation 3.8. In this equation, T_0 is the desired ripple-free torque, and $T(\theta, I_{mag}, \mathbf{f}(\theta))$ is the torque obtained when I_{mag} and $\mathbf{f}(\theta)$ are used to specify the currents.

$$T(\theta, I_{mag}, \mathbf{f}(\theta)) = \left(\frac{1}{2} \mathbf{f}^T(\theta) \frac{\partial \mathbf{L}(\theta)}{\partial \theta} \mathbf{f}(\theta) \right) I_{mag}^2 \tag{3.7}$$

$$\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta)) = \sqrt{\frac{T_0}{T(\theta, I_{mag}, \mathbf{f}(\theta))}} \tag{3.8}$$

In this thesis, the calculation of the correction factor uses measurements of torque as a function of position, taken when the currents have three-phase sinusoidal waveform shapes and a constant magnitude. Once the data is collected, correction factors are calculated and ripple-reduced torque data is gathered using currents modulated by the correction factor. The procedure described in the preceding sentences, forms the ripple-reduction algorithm of this thesis. The body of this chapter develops a the-

oretical justification for why magnitude modulation reduces ripple. Chapter 4 details the algorithm and experimental results. The results demonstrate reduced ripple.

3.2 Theoretical Justification

In order to discuss torque ripple and ripple correction on a technical level, it is first necessary to create a mathematical framework within which the discussion can be carried out. To aid in the selection of an appropriate framework, a brief description of the context of the thesis is now given. It should first be noted that the techniques developed in this thesis are aimed at reducing torque ripple in a motor that is not in saturation. This permits the limitation of the mathematical framework to the consideration of linear magnetic behavior. A second consideration is that the torque ripple of interest is related to the spatial harmonics of inductance variations of the phases. Therefore, a useful model must include some flexibility in specifying the inductance variations. Thirdly, since the proposed technique will amplitude modulate the magnitudes of the three-phase currents, it is necessary to have the ability to incorporate this into the model. With these considerations in mind, it is found that an expression relating torque to the partial derivative of the magnetic-field coenergy suffices, providing flexibility in specifying the inductances and the phase currents.

The magnetic-field coenergy approach begins with a specification of the machine's inductances and currents. It proceeds by calculating the coenergy and then the torque. From an expression for the torque it is possible to calculate a "correction factor" with which to scale the magnitudes of the currents. Using these modulated currents, the motor produces constant torque versus position.

3.2.1 The Inductance Model of Machine Behavior

Before delving into the calculation of torque, a model of the motor must be set up. A few assumptions of the model must be decided up front. First, the model assumes the inductances do not vary with the flux density. That is, the effects of saturation are neglected. The validity of this assumption is verified in Section 4.4.1. This also implies

that the inductances are independent of all the currents in the system. The second assumption is that the inductances vary only with position. Using these assumptions, the desired result of this chapter is worked out in a general form applicable to many motors.

Now to define some variables. The vector $\mathbf{i}$ functions as a vector containing the currents through each winding. If there are N windings with currents $\{i_1, i_2, \ldots, i_N\}$, then $\mathbf{i}$ is given by Equation 3.9.

$$\mathbf{i} = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_N \end{bmatrix} \tag{3.9}$$

The vector $\boldsymbol{\lambda}$ serves as the flux linkage vector. The flux linkages for each winding make up the entries for $\boldsymbol{\lambda}$ as shown in Equation 3.10.

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{bmatrix} \tag{3.10}$$

Using the inductance matrix of an arbitrary system of windings, the flux linkage in each winding can be related to the currents. This relation is given in Equation 3.11 and the definition of the inductance matrix is shown in Equation 3.12. Each entry of this matrix, L_{jk}, specifies the inductance between winding j and winding k. If $j \neq k$ then the inductance is a mutual inductance, otherwise it is a self-inductance.

$$\boldsymbol{\lambda} = \mathbf{L}(\theta)\,\mathbf{i} \tag{3.11}$$

$$\mathbf{L}(\theta) = \begin{bmatrix} L_{11} & L_{12} & \dots & L_{1N} \\ L_{21} & L_{22} & \dots & L_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & L_{N2} & \dots & L_{NN} \end{bmatrix} \tag{3.12}$$

3.2.2 Torque Production

Now that the flux linkages and currents have been defined, torque can be calculated. One of the most elegant ways of calculating torque is by way of the magnetic field coenergy, W'_{fld}. The book *Electric Machinery* [5, Chapter 3] contains an introduction to this method. This presentation follows the format of that book.

Since the magnetic field energy is conserved in quasi-statics, it must equal the net sum over time of the power flows into or out of the system, which is the sum of the power flows into each winding minus any mechanical power out of the system. The power into each winding equals the voltage across the winding, caused by back EMF, multiplied by the current through the winding. The voltage drops across the windings are given by Faraday's Law. This finds expression in a vector form in Equation 3.13, where the e_i are the back EMF's of each winding.

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} = \frac{\mathrm{d}\boldsymbol{\lambda}}{\mathrm{d}t} \tag{3.13}$$

The net power flow into the system from the windings is then given by Equation 3.14.

$$P_{electrical} = \mathbf{e}^T \mathbf{i} \tag{3.14}$$

The total power entering the system is then the electrical input power minus any

work the system might be doing. For the case of a motor, this work is the torque, T, times the angular velocity, ω.

$$P_{total} = \mathbf{e}^T \mathbf{i} - T\omega \tag{3.15}$$

This can be written in differential form by utilizing Faraday's Law, Equation 3.13, and by noting that the angular velocity is the derivative of angular position.

$$\omega = \frac{\mathrm{d}\theta}{\mathrm{d}t} \tag{3.16}$$

Also the realization that the derivative of the field energy with respect to time is equal to the net power into the system, allows Equation 3.17 to be written. In this equation, the variable $W_{fld}(\theta, \boldsymbol{\lambda})$ is the field energy.

$$\mathrm{d}W_{fld}(\theta, \boldsymbol{\lambda}) = (\mathrm{d}\boldsymbol{\lambda}^T)\,\mathbf{i} - T\,\mathrm{d}\theta \tag{3.17}$$

This relation indicates that the field energy is a function of the flux linkage and the position. This allows for an easy calculation of torque versus flux linkage; however, it is more desirable to express torque as a function of the currents in the windings. To this end, the magnetic coenergy, $W'_{fld}(\theta, \mathbf{i})$, is defined in Equation 3.18.

$$W'_{fld}(\theta, \mathbf{i}) = \boldsymbol{\lambda}^T \mathbf{i} - W_{fld}(\theta, \boldsymbol{\lambda}) \tag{3.18}$$

Taking the differential of the coenergy yields Equation 3.19. It can readily be seen that the coenergy is a function of the position and currents.

$$\begin{aligned} \mathrm{d}W'_{fld}(\theta, \mathbf{i}) &= \mathrm{d}(\boldsymbol{\lambda}^T \mathbf{i}) - \mathrm{d}W_{fld}(\theta, \boldsymbol{\lambda}) \\ &= (\mathrm{d}\boldsymbol{\lambda}^T)\,\mathbf{i} + \boldsymbol{\lambda}^T\,\mathrm{d}\mathbf{i} - (\mathrm{d}\boldsymbol{\lambda}^T)\,\mathbf{i} + T\,\mathrm{d}\theta \\ &= \boldsymbol{\lambda}^T\,\mathrm{d}\mathbf{i} + T\,\mathrm{d}\theta \end{aligned} \tag{3.19}$$

This expression can be written without reference to the flux linkage by substituting in Equation 3.11. Also, it should be noted that the inductance matrix is symmetric as a result of conservation of energy in the magnetic field. This allows the matrix $\mathbf{L}^T(\theta)$ to be equated with $\mathbf{L}(\theta)$. This yields the equation below.

$$\mathrm{d}W'_{fld}(\theta, \mathbf{i}) = \mathbf{i}^T\,\mathbf{L}(\theta)\,\mathrm{d}\mathbf{i} + T\,\mathrm{d}\theta \tag{3.20}$$

The magnetic field coenergy at a particular state $(\theta, \mathbf{i})$ can then be found by performing a path integral from the origin to that state. In general, this integration is not independent of path, so a path must be chosen. It turns out that if the path is chosen so as to integrate along the dummy position variable, $\hat{\theta}$, until θ is reached, the remainder of the integral will be independent of path. Thus, the coenergy can be expressed in integral form as shown in Equation 3.21.

$$W'_{fld}(\theta, \mathbf{i}) = \int_0^{\theta} T\,\mathrm{d}\hat{\theta}\,\Big|_{\hat{\mathbf{i}}=\mathbf{0}} + \int_{\mathbf{0}}^{\mathbf{i}} \hat{\mathbf{i}}^T\,\mathbf{L}(\hat{\theta})\,\mathrm{d}\hat{\mathbf{i}}\,\Big|_{\hat{\theta}=\theta} \tag{3.21}$$

Examining the first integral, it should be noted that in the absence of external forces and permanent magnets the torque is zero since the currents are zero. If the torque is zero for the entire path, then its contribution to the coenergy will be zero. Equation 3.21 then simplifies to Equation 3.22.

$$W'_{fld}(\theta, \mathbf{i}) = \int_{\mathbf{0}}^{\mathbf{i}} \hat{\mathbf{i}}^T \mathbf{L}(\theta)\, \mathrm{d}\hat{\mathbf{i}} \tag{3.22}$$

Now it can be observed that because of conservation of energy in the quasi-static magnetic field, the integrand can be written as the gradient with respect to current, $\nabla_{\hat{\mathbf{i}}}$, of a scalar function. Equation 3.23 expresses the scalar function that satisfies the necessary relationships. It is unique up to the addition of a scalar constant.

$$\hat{\mathbf{i}}^T \mathbf{L}(\theta) = \nabla_{\hat{\mathbf{i}}} \left(\frac{1}{2} \hat{\mathbf{i}}^T \mathbf{L}(\theta)\, \hat{\mathbf{i}} \right) \tag{3.23}$$

Using Equation 3.23, Equation 3.22 reduces to Equation 3.24. Returning to what remains of the path integral that was started, it can be seen that the integrand is a gradient and thus the integral is independent of path (also a result of conservation of energy in general). Performing the integral, the field coenergy is given by Equation 3.25.

$$W'_{fld}(\theta, \mathbf{i}) = \int_{\mathbf{0}}^{\mathbf{i}} \nabla_{\hat{\mathbf{i}}} \left(\frac{1}{2} \hat{\mathbf{i}}^T \mathbf{L}(\theta)\, \hat{\mathbf{i}} \right) \mathrm{d}\hat{\mathbf{i}} \tag{3.24}$$

$$W'_{fld}(\theta, \mathbf{i}) = \frac{1}{2} \mathbf{i}^T \mathbf{L}(\theta)\, \mathbf{i} \tag{3.25}$$

From this coenergy expression, the torque can be calculated.

$$T(\theta, \mathbf{i}) = \frac{\partial W'_{fld}(\theta, \mathbf{i})}{\partial \theta} = \frac{1}{2} \mathbf{i}^T \frac{\partial \mathbf{L}(\theta)}{\partial \theta} \mathbf{i} \tag{3.26}$$

In the introduction to this chapter, the torque is expressed as function of position.

Furthermore, the torque is shown to scale quadratically with a linear scaling of the currents. This formulation can be obtained by expressing the current as the product of a magnitude, I_{mag}, and a waveform shape vector, $\mathbf{f}(\theta)$, as explained in Section 3.1. This is expressed mathematically in Equation 3.27.

$$\mathbf{i} = I_{mag}\,\mathbf{f}(\theta) \tag{3.27}$$

Equation 3.27 can be substituted into Equation 3.26 and the magnitude factor can be pulled out. This manipulation is shown in Equation 3.28.

$$\begin{aligned} T(\theta, I_{mag}, \mathbf{f}(\theta)) &= \frac{1}{2} I_{mag}\,\mathbf{f}^T(\theta) \frac{\partial \mathbf{L}(\theta)}{\partial \theta} I_{mag}\,\mathbf{f}(\theta) \\ &= \left(\frac{1}{2} \mathbf{f}^T(\theta) \frac{\partial \mathbf{L}(\theta)}{\partial \theta} \mathbf{f}(\theta) \right) I_{mag}^2 \end{aligned} \tag{3.28}$$

This model of torque production is useful for two reasons. First, it demonstrates that a linear scaling of the magnitude of the currents scales the torque quadratically. Second, it shows that the torque is a function of position, waveform shape, and the magnitude of the current. Thus if torque measurements are taken versus position using known waveform shapes and a known magnitude of current, the torque can be set to any desired level versus position by using the same waveform shapes and by scaling the magnitude appropriately. The following section exploits this feature as a starting place for the ripple-correction technique.

3.2.3 The Derivation of the Correction Factor

This section derives a position-dependent "correction factor" which when used to scale the phase currents results in reduced-ripple torque output. The calculation of a correction factor requires that the torque output first be measured versus position using known current waveform shapes and a known magnitude. The derivation pro-

ceeds as follows. First the effect of scaling the currents on the torque is examined mathematically. From this, the appropriate scaling can be chosen at each position to yield the reduced-ripple torque. These scaling factors, taken over position, form the correction factor.

To begin the derivation, assume a measurement of the torque, $T(\theta, I_{mag}, \mathbf{f}(\theta))$, is made with a known current $\mathbf{i}(\theta)$, $\mathbf{i}(\theta) = I_{mag}\,\mathbf{f}(\theta)$. If the magnitude of the current, I_{mag}, is scaled by a factor of $\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta))$, as defined in Equation 3.29, then the resultant torque can be related to the value obtained without scaling by Equation 3.30. Note that the scaling may be a function of position, making the scaled magnitude of the current, $\tilde{I}_{mag}(\theta)$, a function of position.

$$\tilde{I}_{mag}(\theta) = \tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta))\;\; I_{mag} \tag{3.29}$$

$$\begin{aligned} T(\theta, \tilde{I}_{mag}(\theta), \mathbf{f}(\theta)) &= \left(\frac{1}{2}\mathbf{f}^T(\theta)\frac{\partial \mathbf{L}(\theta)}{\partial \theta}\mathbf{f}(\theta)\right)\left(\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta))\;\; I_{mag}\right)^2 \\ &= \tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta))^2\;\; T(\theta, I_{mag}, \mathbf{f}(\theta)) \end{aligned} \tag{3.30}$$

To keep the torque constant, $T(\theta, \tilde{I}_{mag}(\theta), \mathbf{f}(\theta)) = T_0$, $\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta))$ should be chosen as governed by Equation 3.31.

$$\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta)) = \sqrt{\frac{T_0}{T(\theta, I_{mag}, \mathbf{f}(\theta))}} \tag{3.31}$$

Using the correction factor, $\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta))$, to amplitude modulate the currents, theoretically yields a constant torque over position. Thus, modulation of the magnitude of the currents forms the basis for a simple ripple-correction method.

This chapter has developed the theory of operation for a ripple-reduction technique. It first demonstrates that torque can be expressed as function of position.

Furthermore, the torque scale quadratically with a linear scaling of the phase currents. Using this information, this chapter derives an expression for a correction factor which when used to scale the currents will reduce the torque ripple of the motor. However, in order to calculate the correction factor, it is necessary to characterize the motor's torque production versus position. The next chapter describes the method used in this thesis for obtaining the characterization, and proceeds from the characterization to calculate correction factors and apply them to the currents. The following chapter concludes with the results of applying the correction factors.

Chapter 4

Ripple Reduction

4.1 Introduction

As discussed in Chapter 3, the torque-ripple-reduction technique modulates the magnitude of the currents in the motor. The modulation requires prior knowledge of the position-dependent variations in torque for given currents in the motor. Acquiring knowledge about the position dependence through characterization becomes the first step in the ripple-reduction algorithm. With this data the algorithm can compute the necessary modulations, in the form of "correction factors," and then apply these correction factors to the currents in the motor to obtain a reduction in torque ripple.

Following the general outline of the algorithm given above, this chapter documents the specific implementation of the algorithm used in this thesis. Figure 4-1 graphically portrays the steps of the algorithm. As can be seen in the figure, during the characterization phase raw data is collected. This data is then processed to account for the aperiodic sampling of the data. The "de-jittered" data then undergoes a correction for baseline torque, that is, torque present when there are no currents in the motor. Following the characterization phase, the correction factors are calculated from the baseline-corrected data. The final phase is the actual application of the correction factors and a measurement of the results. The measurement procedure used during verification repeats the procedure used during characterization.

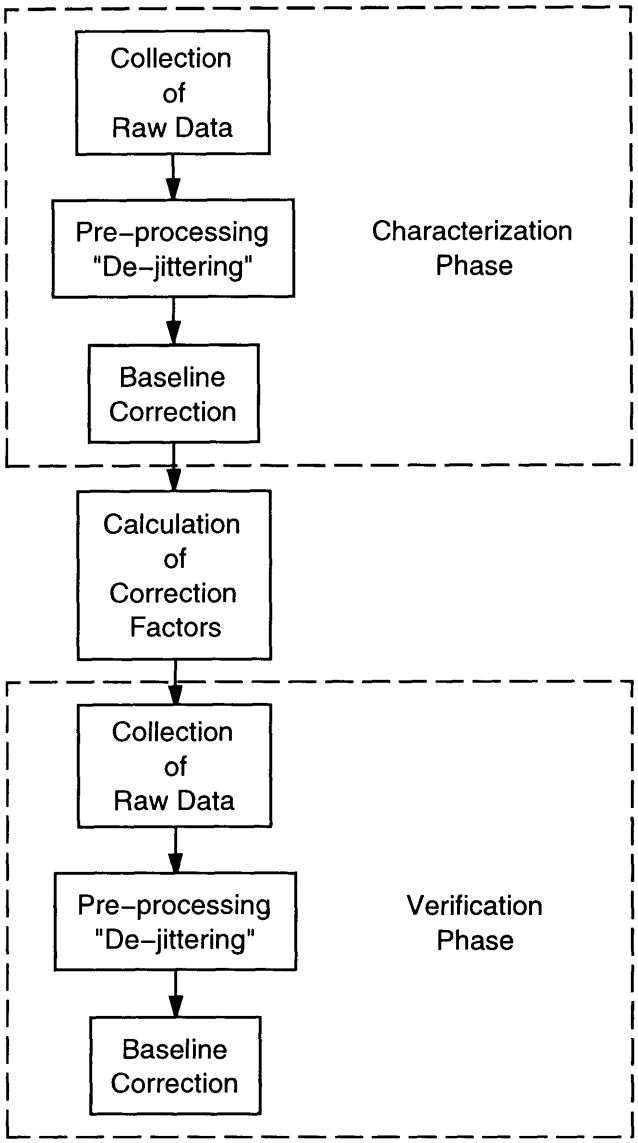

Figure 4-1: Flowchart of Characterization, Calculation and Verification.

The sections of this chapter document the individual steps just mentioned. Section 4.2 details the characterization dataset to be collected as well as the conditions for the data collection. It also provides a glimpse at the raw data that is gathered. Section 4.3 explains the next step in the data-flow which is the pre-processing of the raw characterization data. This section implicitly documents the processing done on the raw data collected during the verification phase as well, since the processing is the same. Once the raw data is pre-processed, it is used in the calculation of the correction

factors as described in Section 4.4. Once the correction factors are calculated, they are applied and verification data is gathered and processed. Section 4.5 comments on this step of the algorithm. The data demonstrates that the torque-ripple-reduction algorithm reduces torque ripple. Section 4.6 examines the performance of the technique in regions where the motor is not explicitly characterized. The performance is on par with that found in well characterized regions.

4.2 The Characterization Dataset

As a preliminary step in the ripple-reduction technique presented in this thesis, it is necessary to characterize the motor's torque production versus position and current. For reasons mentioned in Section 3.1, the currents are chosen to be three-phase sinusoids on both the rotor and the stator. The currents can be described mathematically as expressed in Equations 4.1–4.6. i_{AR}, i_{BR}, and i_{CR} are the rotor A, B and C phase currents respectively, and i_{AS}, i_{BS}, and i_{CS} are the stator A, B and C phase currents respectively. δ_R and δ_S are constants used to set the relative phase angle of the rotor and stator magnetic poles. This choice is made in such a way as to maximize torque output [17][18]. The thermal management technique of Chapter 5 utilizes the freedom in choosing the absolute phase angles of δ_R and δ_S.

$$i_{AR} = I_{mag}\cos(\delta_R) \tag{4.1}$$

$$i_{BR} = I_{mag}\cos(\delta_R - \frac{2\pi}{3}) \tag{4.2}$$

$$i_{CR} = I_{mag}\cos(\delta_R - \frac{4\pi}{3}) \tag{4.3}$$

$$i_{AS} = I_{mag}\cos(9\,\theta + \delta_S) \tag{4.4}$$

$$i_{BS} = I_{mag}\cos(9\,\theta + \delta_S - \frac{2\pi}{3}) \tag{4.5}$$

$$i_{CS} = I_{mag}\cos(9\,\theta + \delta_S - \frac{4\pi}{3}) \tag{4.6}$$

For the characterization phase of the algorithm, torque measurements are taken

for one complete revolution of the rotor, while stepping the amplitude of the phase currents, I_{mag}, after each revolution. I_{mag} ranges from zero to eleven amperes in one ampere increments. The process is repeated to acquire two sets of measurements at each current level (except at ten and eleven amperes). For a graphical portrayal of the dataset, Figure 4-2 plots the data from one trial at each current. During each revolution, the motor is rotated at a rate of approximately 0.05 RPM. The rotor position is divided into 2049 bins per revolution, and one measurement is taken per bin. For more technical information on the measurement apparatus, Appendix G documents the position measurement subsystem and Appendix E documents the dynamometer subsystem.

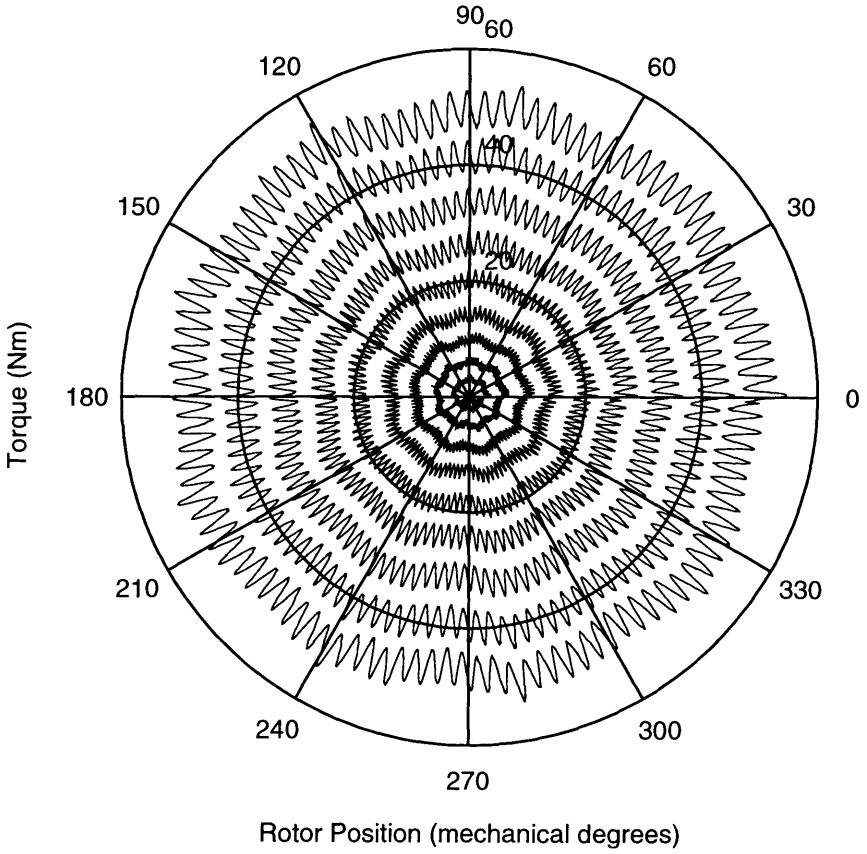

Figure 4-2: Raw Characterization Data Portraying Measured Torque versus Position and Current.

4.3 Pre-processing of Collected Data

This section details the pre-processing of data which is done in both the characterization and verification phases (see Figure 4-1). The primary goal of the data collection procedure is to ensure that the data is actually descriptive of the torque produced by the motor and not other factors. Investigation has revealed three areas for concern. The first area is the finite resolution of the torque transducer, which introduces quantization errors into the torque measurements. Section 4.3.1 provides details regarding the accuracy of the instrument and the impact that its limitations have on the experiments to be conducted. The second area for concern is the accuracy of the position measurements. The position is measured nearly periodically but each measurement has some amount of measurable deviation from the ideal periodic sampling. Section 4.3.2 details the steps taken to ensure that the position measurement is utilized as best as possible. The final area for concern is the existence of baseline torque, that is, torque present when there are no currents in the motor. It is possible to subtract out this effect. Section 4.3.4 explains the specifics of this step.

4.3.1 Quantization Errors in the Torque Measurements

The resolution of the torque measurement system limits the accuracy of the data gathered during the collection procedure. The resolution of the torque transducer is 0.311 Nm. With additional electrical noise due to the switching in the phase-current drive electronics, the measurement accuracy further diminishes. As an idea of the accuracy of the measurements, the RMS sample-to-sample difference between two characterization runs at eight amps yields a standard deviation of 0.4088 Nm. This is seen in subsequent sections to be about equal to the RMS torque ripple after correction. Thus, the accuracy of the torque measurement critically hinders the effectiveness of the technique. Future experiments need to put more emphasis on the resolution of the measurement system. Details of the implementation and problems of the torque measurement system and related systems can be found in Appendices E, H and K. With a more accurate measurement system, it should be possible to better

probe the limits of this ripple-reduction technique.

4.3.2 Removing the Effect of Sampling Jitter on the Data

As can be seen in Figure 4-2 which plots measured torque versus position versus current in polar format, the torque ripple displays a periodic structure. This is confirmed by viewing a DFT of the data as shown in Figure 4-3. As a result of the periodicity, the analysis of torque ripple in the spatial-frequency domain simplifies the problem and allows for a more compact representation of the dominant features of the problem. However, the data collection system is not optimized for taking the periodic samples necessary for an accurate transformation to the frequency domain. Appendix K discusses the flaws in more depth. The following discussion brings out some precautionary measures that have been developed to ensure an accurate frequency-domain representation of the problem.

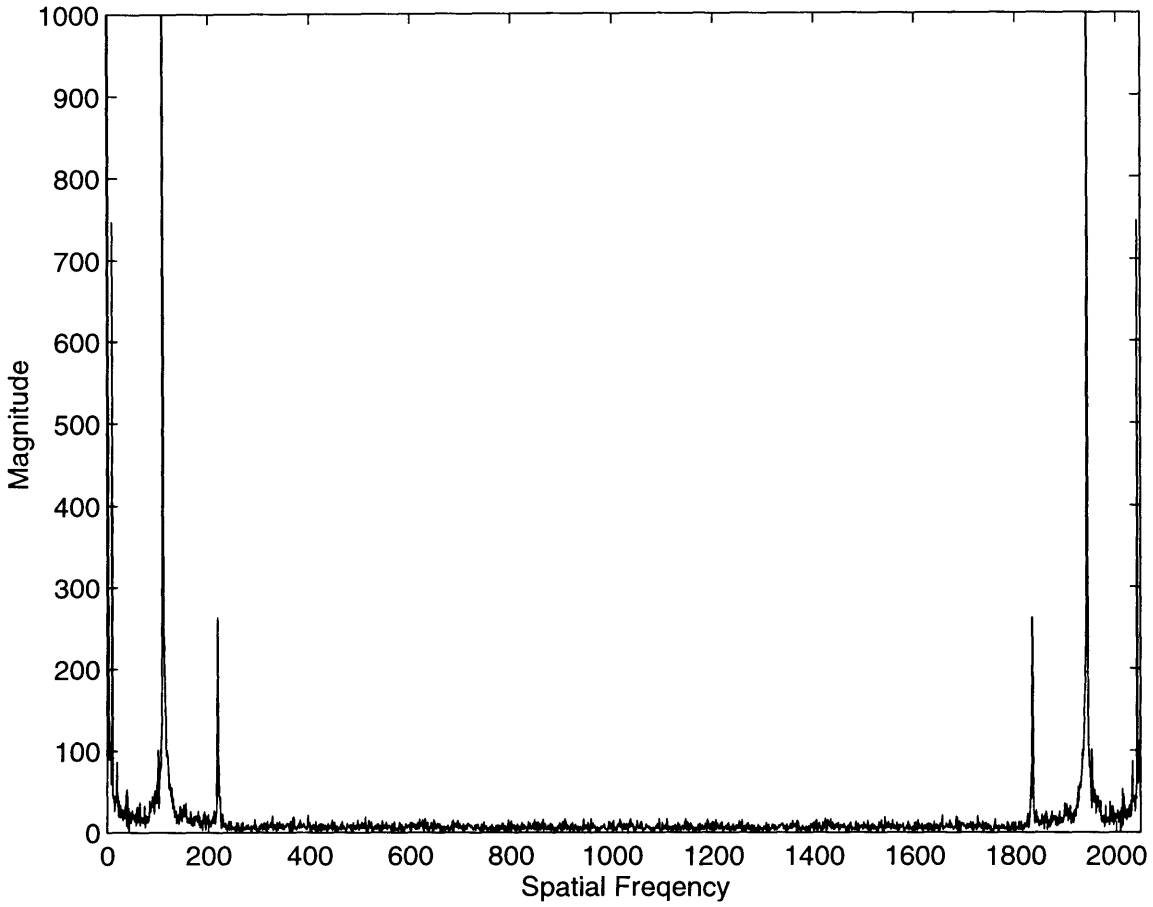

Figure 4-3: Magnitude of the DFT of Measured Torque Data at $I_{mag} = 8$ Amps.

Because the rotary encoder has a resolution of 65536 counts per revolution, it is possible to resolve the position to a greater accuracy than the number of measurement bins would indicate. However, since the bandwidth and storage capability of the measurement system are limited, it is not possible to take a data point at every count of the encoder, and actually is not necessary. It proves more than sufficient to take 2049 data points per revolution, as the highest spatial harmonic of torque is the 216th harmonic. Taking 2049 samples per revolution exceeds the minimum necessary to avoid aliasing; that is, the sample rate exceeds the Nyquist rate. Although the system takes enough samples to accurately resolve the frequency content, the measurement apparatus generates aperiodically spaced samples which introduce

noise into the spectrum of the signal. Appendices D, F, G, J and K document the measurement apparatus and the reasons for the aperiodic sampling.

Irregularly spaced measurements of torque versus position can cause inaccurate resolution of the torque's spatial harmonics if a Discrete Fourier Transform (DFT) is applied to the data. This can be best visualized by examining the effect of performing a 512-point DFT on an aperiodically sampled sum of two sinusoids. Figure 4-4 depicts the result of performing a DFT on a periodically sampled sum of sinusoids and Figure 4-5 depicts the result of performing a DFT on an aperiodically sampled version of the same waveform. Clearly the DFT of the aperiodically sampled waveform shows some additional noise and loss of spectral resolution. This is the problem to be addressed.

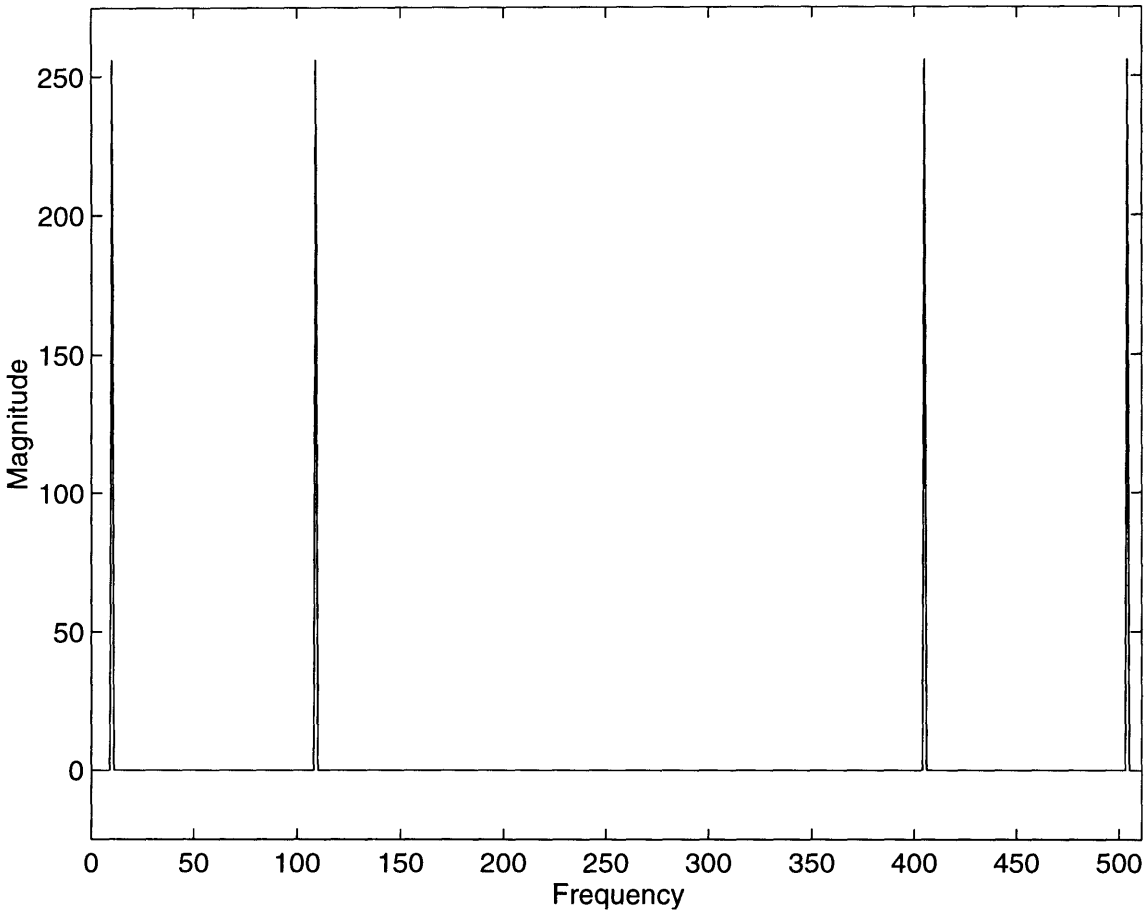

Figure 4-4: 512-point DFT of a Periodically Sampled Signal.

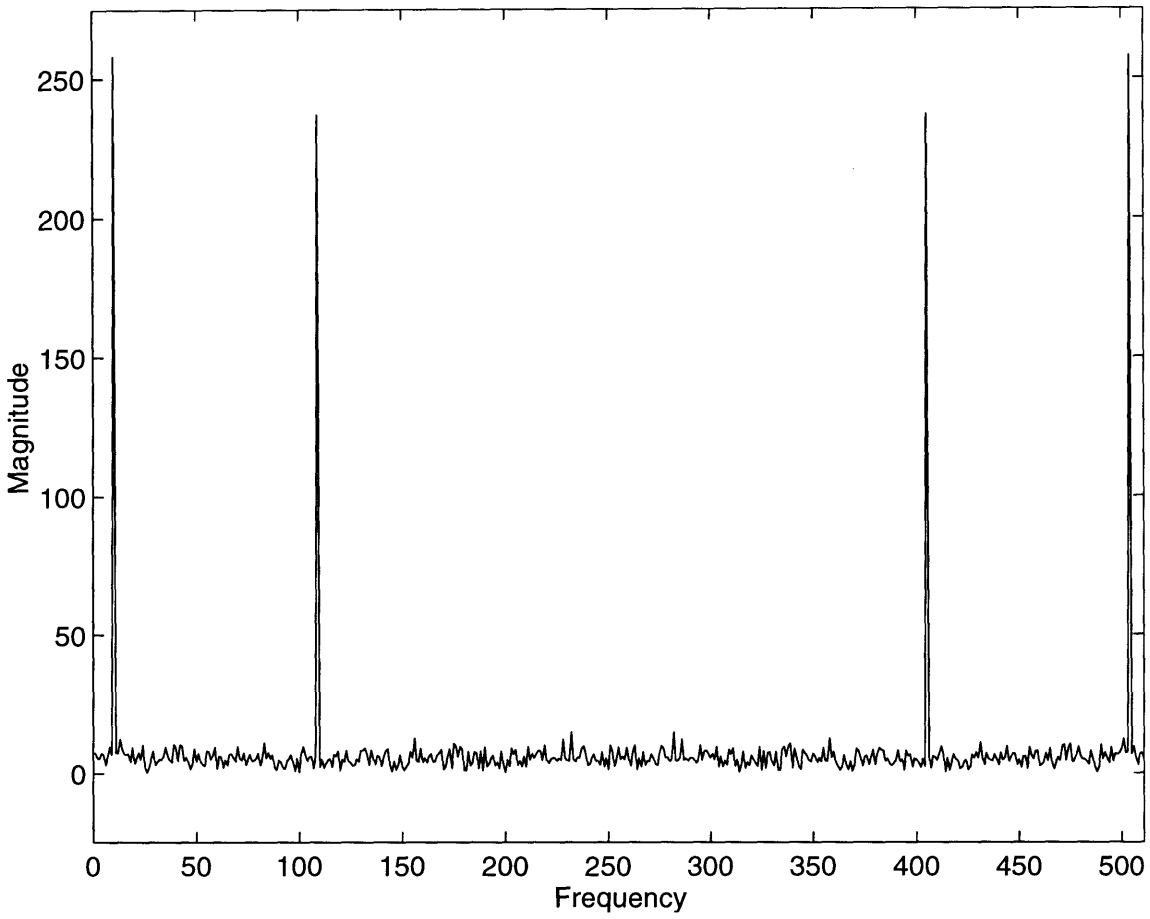

Figure 4-5: 512-point DFT of an Aperiodically Sampled Signal.

To ensure that the irregular spacing does not pose a problem, a more general approach to calculating the DFT has been constructed and used as a pre-processing step on all raw data. This approach to the DFT has the effect of "de-jittering" the data. This terminology seems appropriate because the calculation compensates for the jitter in the sampling. It is not entirely clear if the "de-jittering" is necessary because at a sampling rate of over eight times the Nyquist rate, the effects of aperiodic sampling tend to get averaged out. However, if the sampling rate were closer to the Nyquist rate, this technique would be a powerful tool.

The more general form of the DFT works as follows. A bandwidth-limited, periodic input signal, $x(t)$, is taken as an input. This signal is aperiodically sampled at

known times, $t[k]$, over one period. Since $x(t)$ is periodic, its Fourier transform should only have harmonics of its fundamental frequency, which implies that it should be representable by a Fourier series. Using the condition that the signal is bandwidth-limited, the Fourier series can then be written with a finite number of terms. It is possible to set up a set of linear equations relating the Fourier series coefficients to the aperiodically spaced samples of the original waveform. These equations can then be solved. This finite sequence of coefficients comprises the sequence of coefficients for the DFT, however it is arrived at by solving the linear equations as opposed to performing a transform. This framework provides the flexibility to handle the aperiodic spacing of the samples. Thus, for an aperiodic sampling of a bandwidth-limited periodic waveform, it is possible to obtain the DFT coefficients.

The above description can be written mathematically as follows. Given that the input signal is periodic with period T, the time series $x(t)$ can be expressed as a Fourier series.

$$x(t) = \sum_{n=-\infty}^{\infty} X[n] e^{j2\pi n\, t/T} \tag{4.7}$$

Since the signal is also band-limited, it is possible to put constraints on the coefficients, $X[n]$. If the signal has a finite bandwidth, $\frac{N}{T}$, then the coefficients obey Equation 4.8.

$$X[n] = 0, \quad |n| > N \tag{4.8}$$

The application of Equation 4.8 to Equation 4.7 simplifies Equation 4.7 to Equation 4.9. It is also possible to write the time representation only at the sample times, $t[k]$.

$$x(t[k]) = \sum_{n=-N}^{N} X[n] e^{j2\pi n\, t[k]/T} \tag{4.9}$$

This Fourier series representation of $x(t[k])$ now defines a set of equations in $2N+1$ unknowns, the $X[n]$. It is a relatively simple matter to solve these equations using a computer. Once the coefficients, $X[n]$, are obtained they can be manipulated into the conventional DFT format by zero-padding and re-ordering. Also, the terms $X[N]$ and $X[-N]$ must be summed; this additional constraint properly accounts for the loss of phase information for signals at half the sampling frequency. Appendix A lists Matlab code that can be used to perform the de-jittered DFT. Using the Fourier series form for the $X[n]$, it is possible to construct a periodically sampled version of the original continuous signal by employing Equation 4.9 with uniformly spaced values of $t[k]$. Similarly, taking an inverse DFT of a DFT representation constructs a periodically sampled representation of the original signal.

To test the de-jittered version of the DFT, it can be applied to the aperiodically sampled sum of sinusoids examined earlier. If a traditional and a de-jittered DFT are applied to the data, the DFT's displayed in Figures 4-6 and 4-7 result. The de-jittered results show increased spectral resolution and a lower (zero) noise floor.

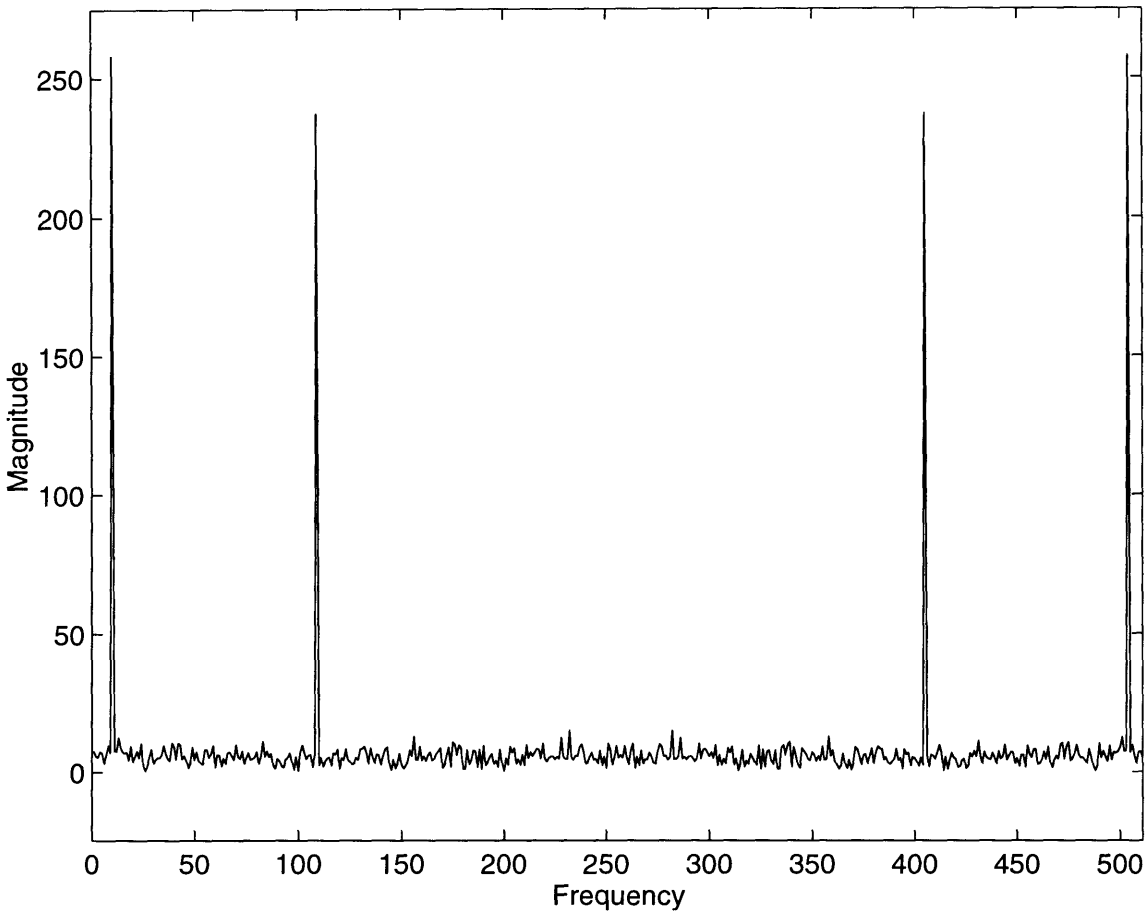

Figure 4-6: 512-point DFT of an Aperiodically Sampled Signal.

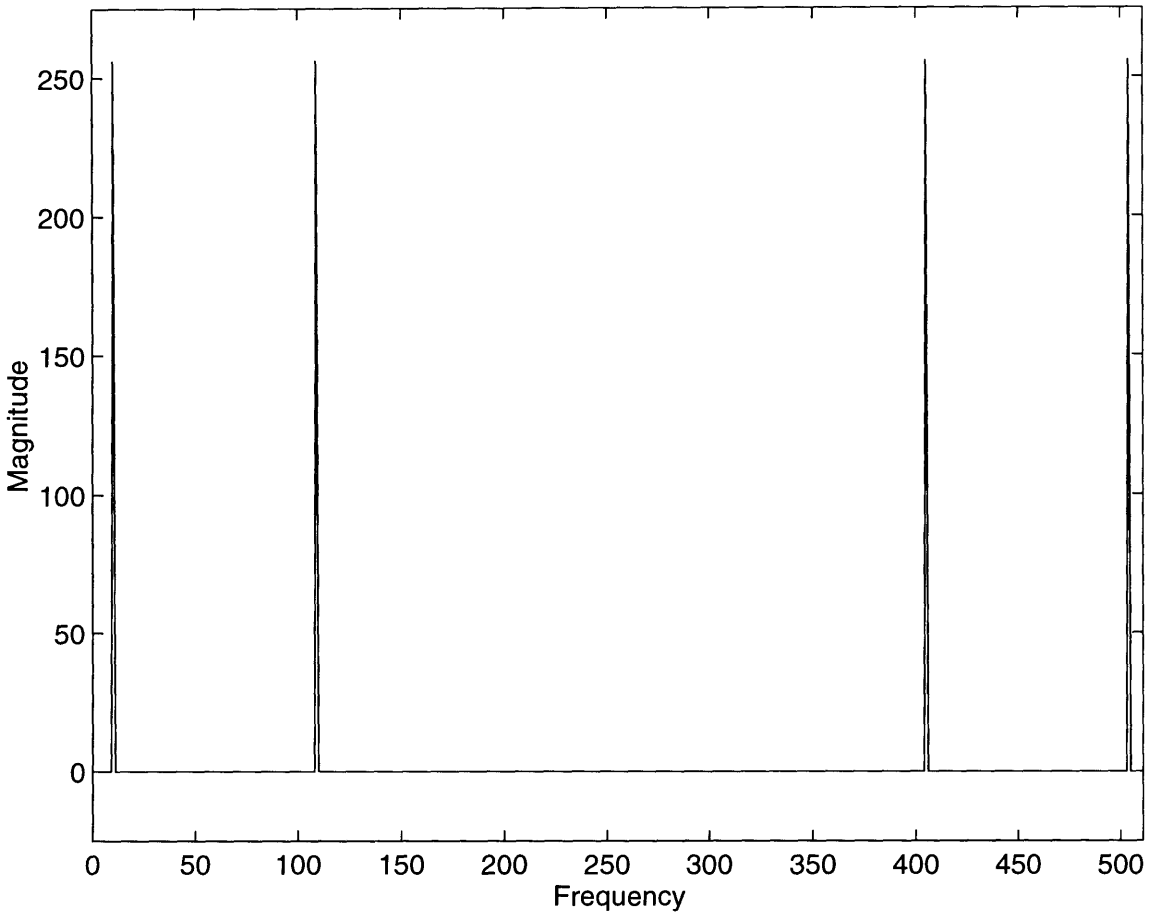

Figure 4-7: 513-point De-Jittered DFT of an Aperiodically Sampled Signal.

For this thesis the data set consists of 2049 sampled measurements, $T[\theta[n], I_{mag}]$, of the continuous periodic torque waveform, $T(\theta, I_{mag})$. For such large sample sequences, the array size to do the de-jittered DFT computation is so large that it is not possible to do it on any computer with a standard math package. To get around this obstacle, the problem can be broken down into four smaller sequences of length 513. The four sample sequences are chosen simply by beginning with the first through fourth data points and taking every fourth point after that, as well as the 2049th point for each sequence. This length of sequence is permissible because the highest harmonic of any importance is the 216th spatial harmonic, and thus Nyquist's criteria is satisfied. 513 points are chosen so that when the de-jittered DFT is applied, the result is a DFT

of length 512, which is amenable to Fast Fourier Transform (FFT) algorithms. The four data sets are used to construct four de-jittered DFT's which are then averaged. This averaging reduces the noise floor of the measurement, as opposed to doing just one de-jittered DFT on 513 points.

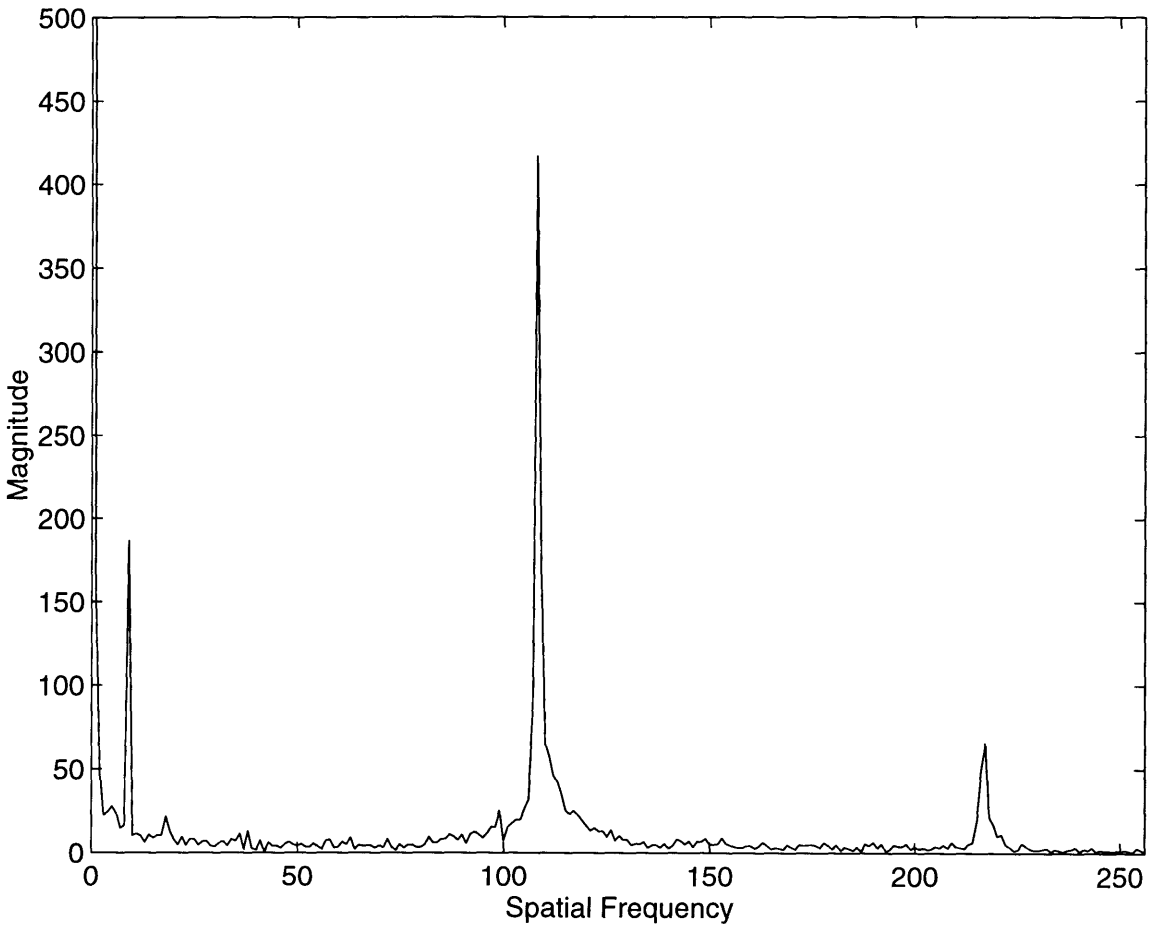

Figure 4-8: DFT on 2049 Points.

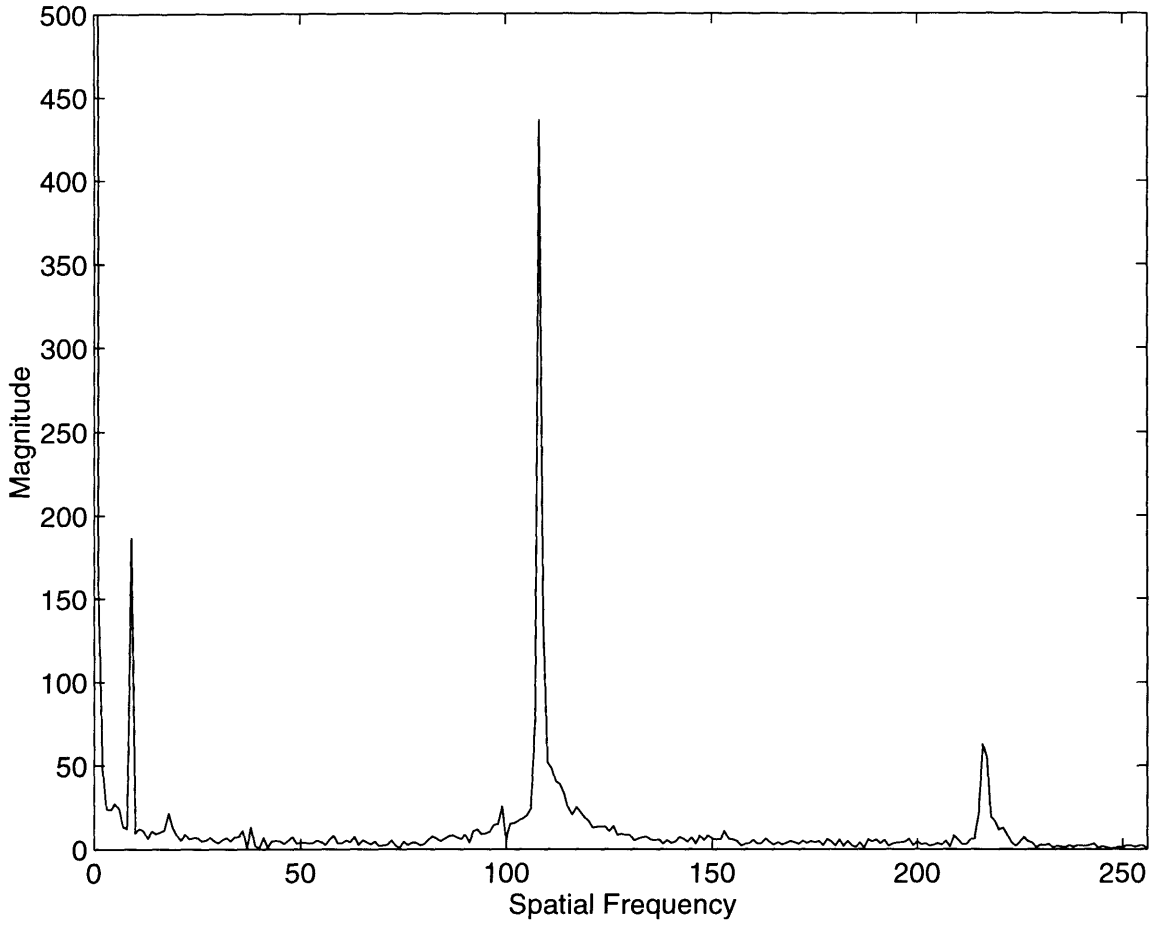

Figure 4-9: De-Jittered DFT on 2049 Points.

To examine the results of this procedure, compare Figure 4-8 and Figure 4-9, showing DFT's of the motor's torque output versus position. Figure 4-8 shows the results of a traditional DFT done on 2049 points. Figure 4-9 shows the results of averaging the four de-jittered DFT's constructed from 2049 points. It should be noted that the axes are cropped for the purpose of creating a more illustrative scaling. The de-jittered DFT again demonstrates sharper spectral resolution of the harmonics. It even moves the peak of the 216th spatial harmonic from 217 to 216 where it belongs, since it is the second harmonic of the slot ripple. The amplitudes of the harmonics are larger in the de-jittered DFT and there is less sideband energy. The noise floor is reduced as well. However, these improvements are subtle and at times slight. To

more graphically see this, an overlay of the two DFT's is shown in Figure 4-10. The lack of a large difference calls into question the need for doing the de-jittered DFT. The best recommendation is that it should be decided for each particular system, taking into consideration the desired accuracy of the results.

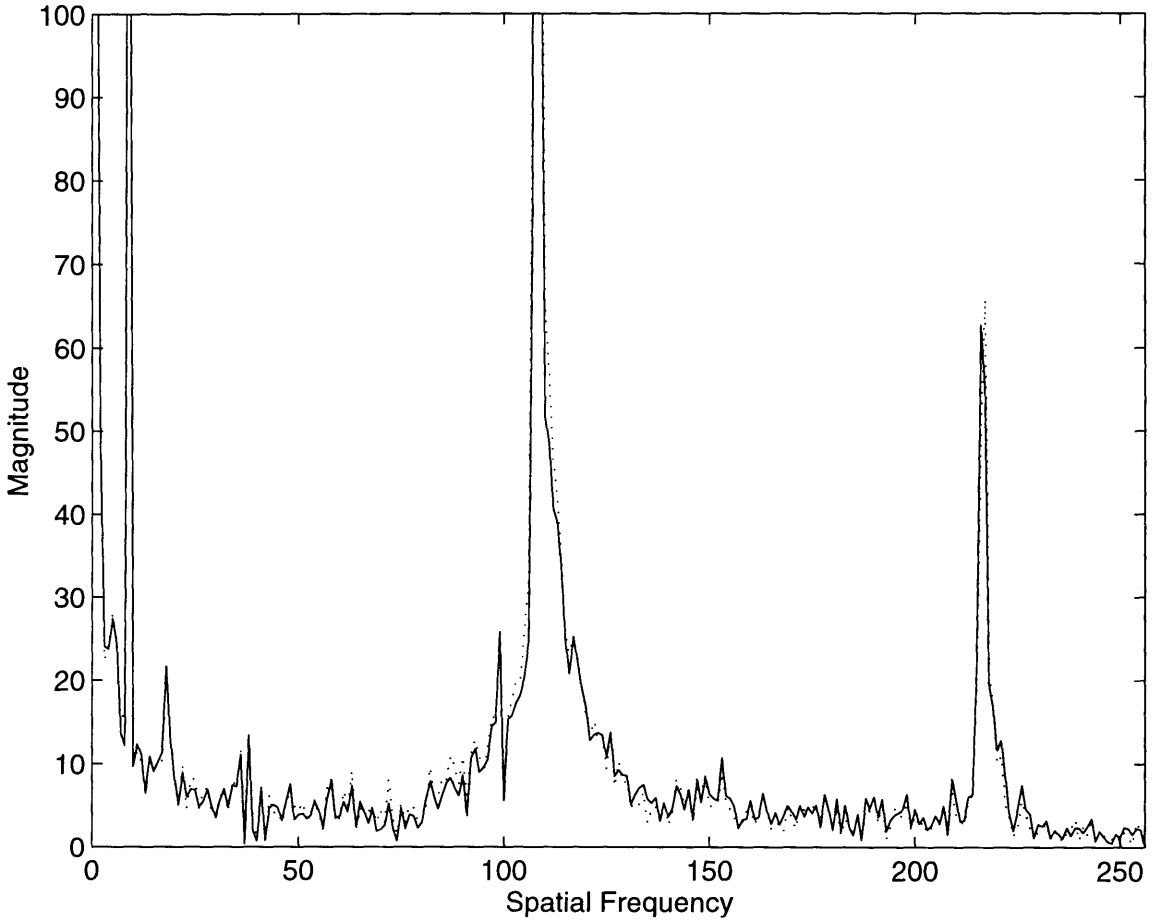

Figure 4-10: A Comparison of the Standard DFT (dotted) and the De-Jittered DFT (solid).

4.3.3 De-Jittered Data

Once the jitter elimination is done, it is possible to take a look at torque versus position versus current. It is helpful to plot this data in polar format, as it graphically reveals more of the harmonic structure of the ripple. A polar plot of the de-jittered

characterization data is shown in Figure 4-11. The plot shows torque versus spatial angle for currents ranging from three to eleven amps.

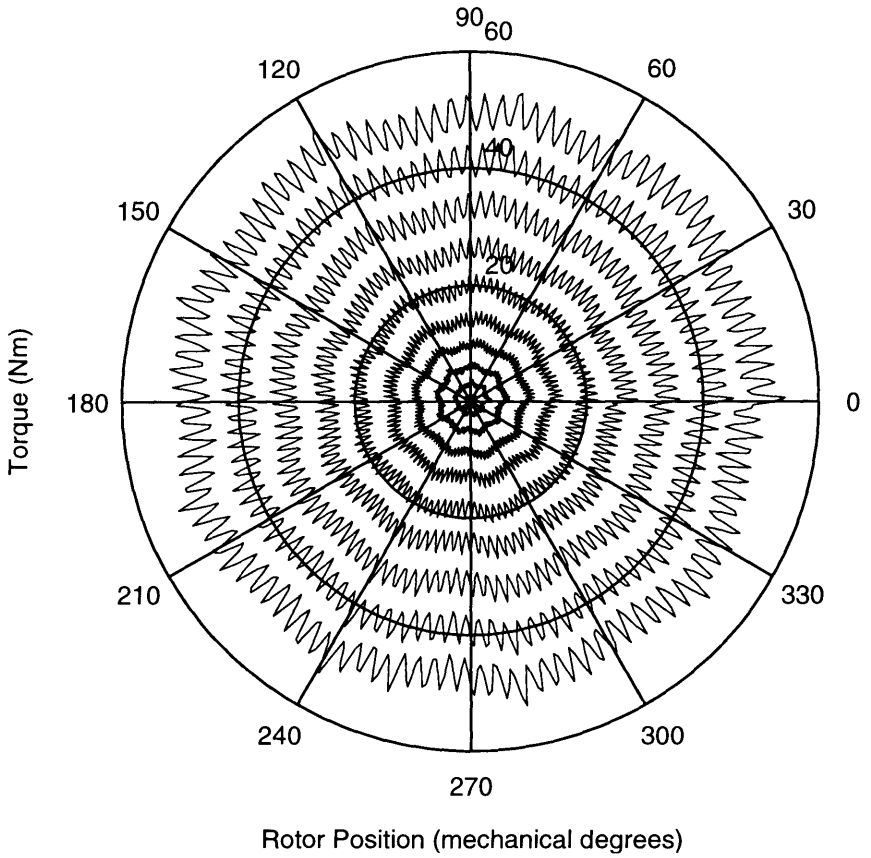

Figure 4-11: De-Jittered Torque Data versus Position and Current.

Several characteristics of the data should be noted. There is a prominent 108th spatial harmonic, which is about 5% of the mean torque value, as well as a significant ninth harmonic. There is also a noticeable reduction in torque as the rotor rotates in the counter-clockwise direction. This reduction in torque can be traced to a zero-current torque caused by the pulling of cooling lines, described in Section 1.2, attached to the motor. The cooling lines exert a spring-like force on the motor. This calls attention to the need for a zero-current, or baseline, correction of the torque.

4.3.4 Zero-Current Torque Correction

The failure to account for a torque present on the motor in the absence of currents can introduce significant error into the correction of torque ripple. Since the ripple-correction technique depends on scaling the currents to generate a constant torque output, any confusion over current-dependent and current-independent torque causes inaccuracies in the correction. The significance of this problem depends on the particular motor in question. Because of the mechanical construction of the motor used in this thesis, the zero-current torque is considerable and thus it is necessary to account for it.

There are two methods to account for the zero-current torque and the choice of method depends on the end objective. If it is desired to command a torque and have that torque be generated, it is necessary to subtract the zero-current torque from the desired torque to calculate the torque the motor must produce. In this way, currents can be commanded so as to cause the net sum of baseline torque and current-dependent torque to equal the desired torque. If measurements of the current-dependent torque are the only objective, then it suffices to subtract the zero-current torque from any measured results. This is the approach used in this thesis.

The existence of zero-current torque with the motor used in this thesis reveals a need to account for the mechanical forces acting on the motor. To limit the scope of the problem, an emphasis has been placed on the forces exerted during the first counter-clockwise revolution starting from position zero, where the ripple-correction algorithm is primarily tested. Yet, to get a more general picture of the scenario, data is taken over the entire positional range of the motor. An effect of the rotational direction on the baseline torque has also been discovered, calling attention to the need to characterize the motor during both clockwise and counter-clockwise rotation. This directional effect is not a major concern in this work since the emphasis is on the counter-clockwise revolution; however, future examinations of torque control should bear this phenomenon in mind.

To characterize the baseline torque, a series of experiments are conducted. First,

the rotor is turned in the counter-clockwise direction for two full revolutions, its entire operating range, while measuring torque. Similarly, data is collected for two revolutions in the clockwise direction. Two runs of data are collected in each direction. Furthermore, an additional three runs of data are collected over the 0 to 360 degree range in the counter-clockwise direction. The collected data is then de-jittered as described in Section 4.3.2 and the data is averaged among runs. The averaging is necessary to reduce the measurement error caused by measurement uncertainty, especially since the zero-current torque is on the order of the resolution of the dynamometer, 0.311 Nm. Figure 4-12 and Figure 4-13 portray the results of a characterization experiment, indicating magnitudes versus position in polar coordinates. Figure 4-14 plots the values of the torque versus position in Cartesian coordinates.

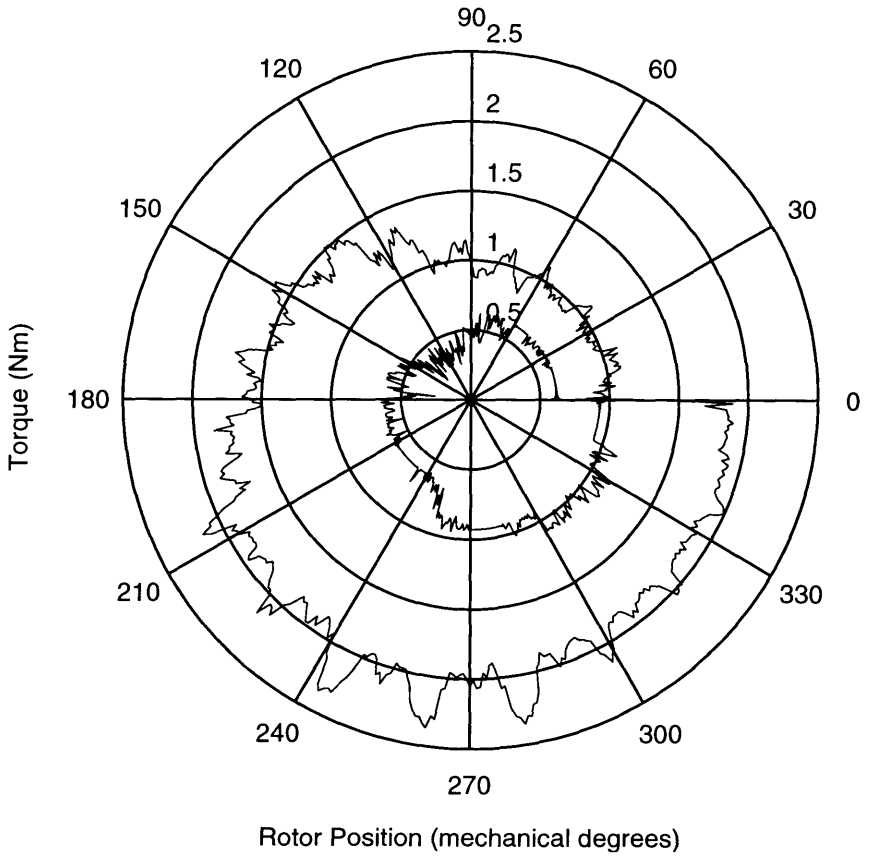

Figure 4-12: Magnitude of Baseline Torque for Counter-clockwise Revolutions.

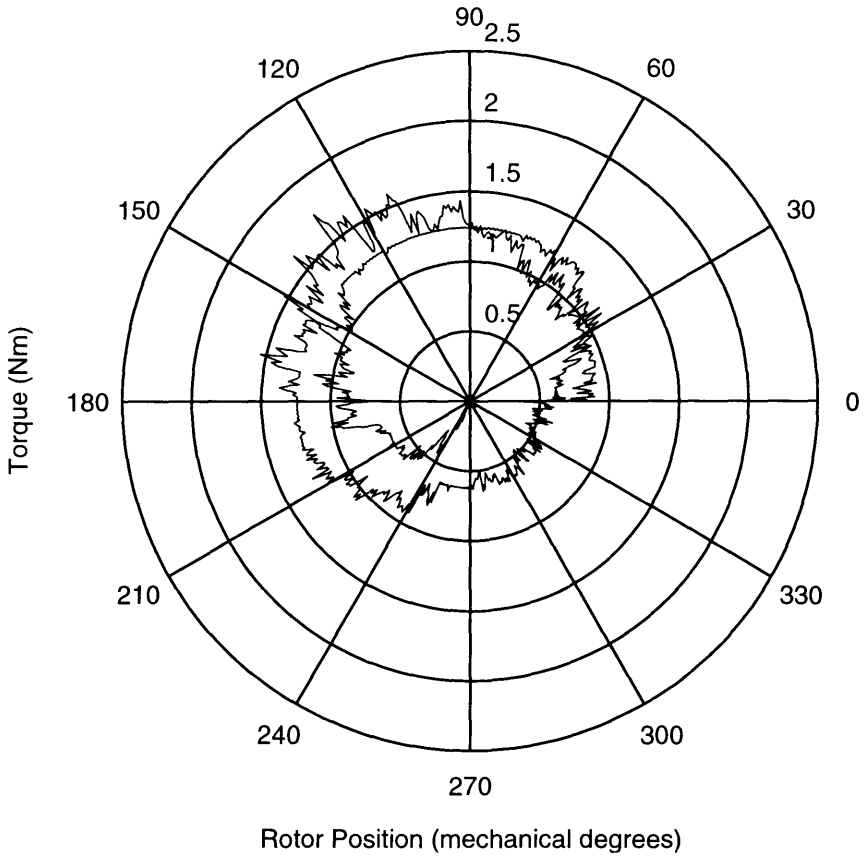

Figure 4-13: Magnitude of Baseline Torque for Clockwise Revolutions.

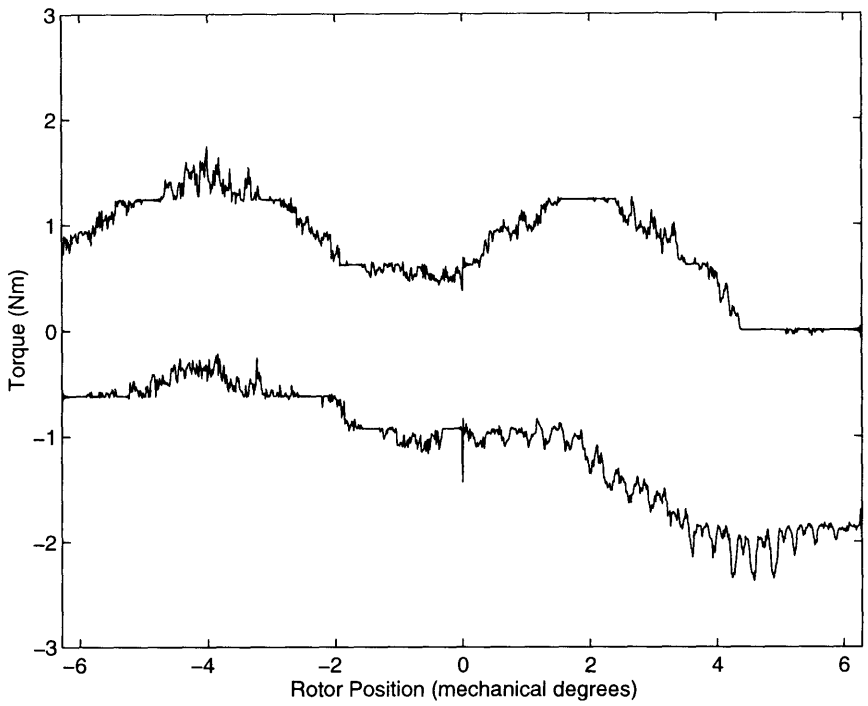

Figure 4-14: Composite of Baseline Torque Showing Counter-clockwise and Clockwise Data.

Upon examining the data, a number of points become obvious. First, the Cartesian plot exhibits a difference in zero-current torque between clockwise and counter-clockwise rotation. It is believed this difference is caused by bearing friction which opposes the rotation. The Cartesian plot also demonstrates a noticeable first-harmonic torque caused by gravity's action on the asymmetric weight distribution of the motor. The polar plots indicate a considerable spiral in the torque for both the counter-clockwise and clockwise sets of data. This is caused by the spring-like action of the cooling lines. This force is proportional to the displacement from the spring's zero-force point.

As the experiments later in the thesis place emphasis on the first counter-clockwise rotation from the zero point, a model of baseline torque is now made solely for that range of operation. Since the baseline torques act in an additive way, a least-squares fit to the torque can be made assuming the form of Equation 4.10. This smoothes the noisy measurement data and extracts the dominant features.

$$T_{baseline} = a_1 \cos(\theta) + a_2 \sin(\theta) + a_3\theta + a_4 \tag{4.10}$$

The resultant model is super-imposed on the original baseline measurements in Figure 4-15. Table 4.1 lists the coefficients for the curve fit. This model for baseline torque can be used to account for torques not related to current. This ability is absolutely necessary in order to control torque by commanding the phase currents in an environment where feedback is not available. It is also necessary in order to more accurately measure the current-dependent torque.

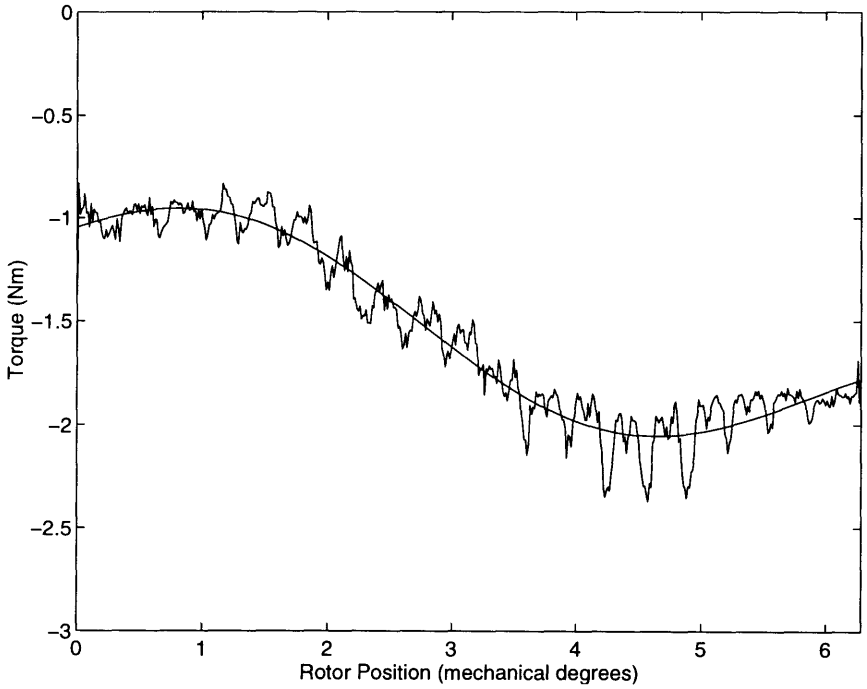

Figure 4-15: Magnitude of Baseline Torque for Counter-clockwise Revolution with Baseline Model Super-imposed.

Coefficient	Value
a_1	0.1384
a_2	0.3177
a_3	-0.1169
a_4	-1.1845

Table 4.1: Coefficients for the Baseline Torque Model.

4.3.5 Baseline-Corrected Data

In this section, the baseline correction is applied to the de-jittered characterization data gathered earlier in this chapter. A plot of the baseline-corrected data is shown in Figure 4-16, and a DFT of the data is given in Figure 4-17. Observe that the decreasing values of torque versus rotor angle seen previously are no longer present at low current and torque levels. However, at higher current levels, the decreasing of torque versus rotor angle still occurs, as evidenced by the discontinuity at the 0° position, for reasons that are not known. Yet, even in these cases the baseline correction does reduce the error.

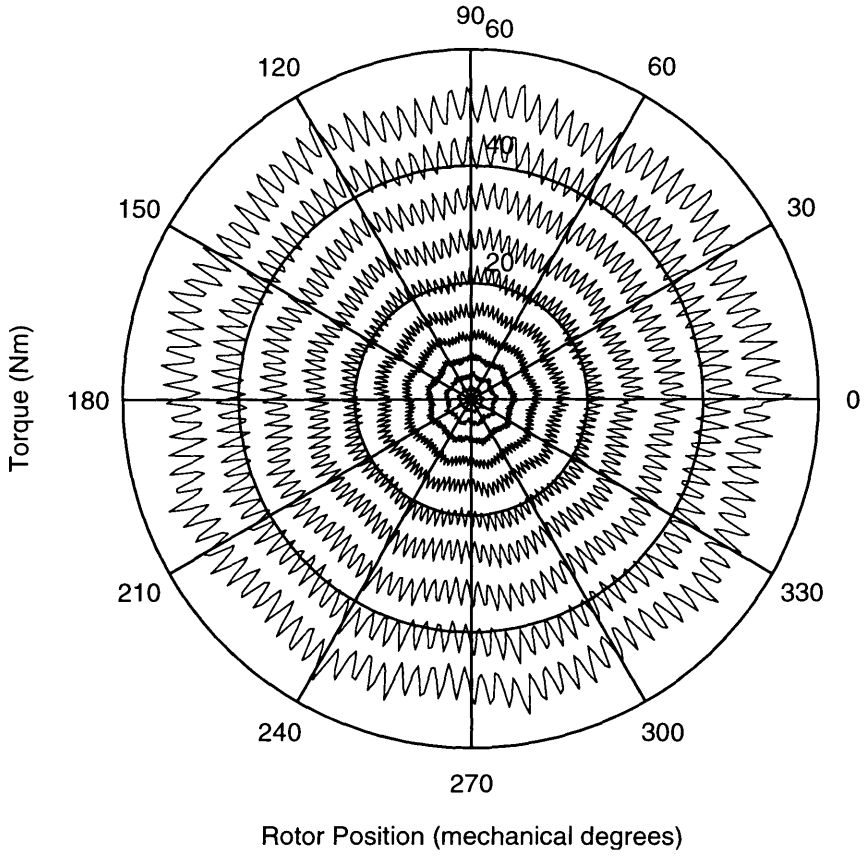

Figure 4-16: Baseline Corrected Torque Data.

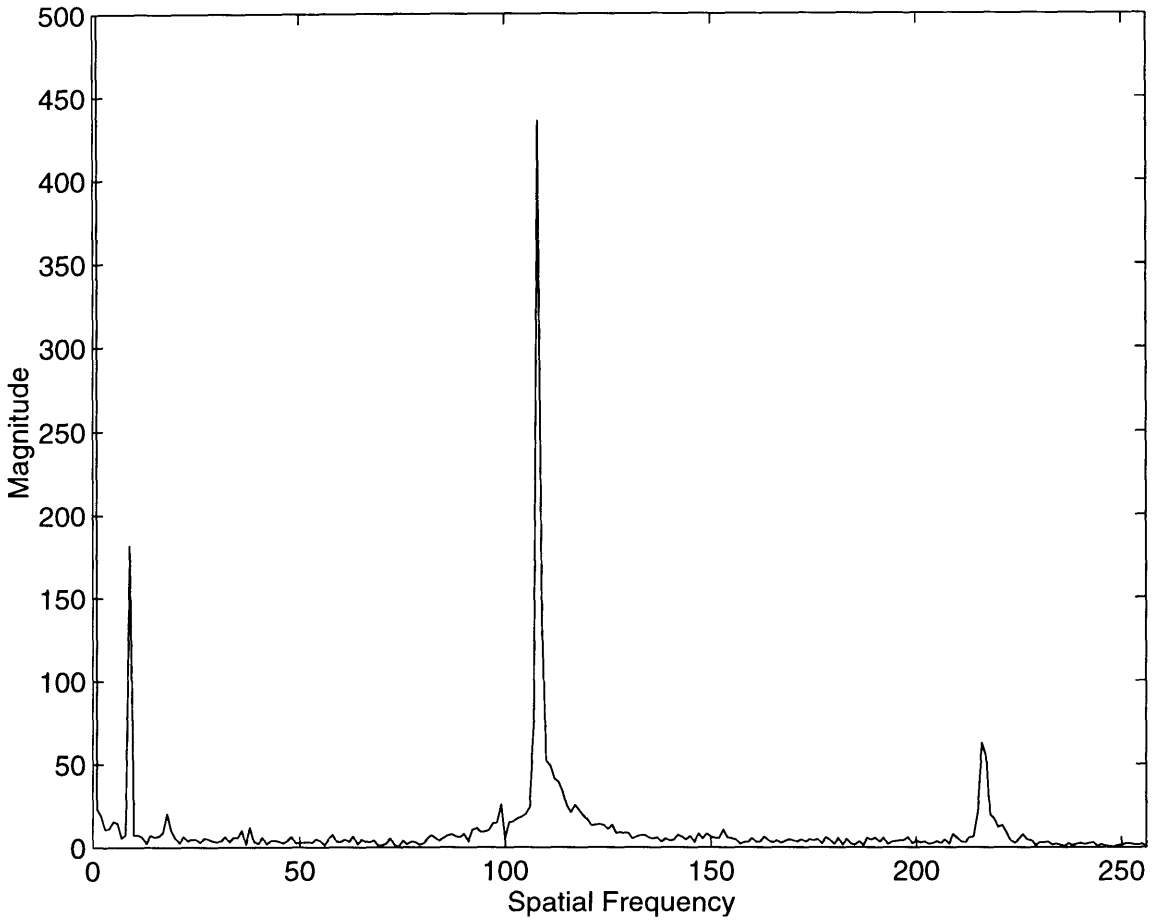

Figure 4-17: Magnitude of DFT of Baseline Corrected Torque Data at 8 amps.

4.3.6 Comments on the Characterization Data

The baseline-corrected characterization data of the previous section can be analyzed to find the significant contributors to the torque ripple. The most dominant features are the 108th and 216th harmonics, which are the fundamental and second harmonic of the slot ripple. These produce an RMS ripple approximately equal to about 5% of the average torque. A prominent ninth harmonic of torque ripple also exists. The cause of this ripple had eluded explanation during the experimentation period of this thesis. In subsequent research, Jackson [7] discovered that this harmonic resulted from an unintended harmonic distortion in one of the phase currents. Nevertheless,

as will be demonstrated, the relatively small amount of distortion does not preclude the success of the ripple-correction technique, although it does diminish it slightly.

The ensemble of data gathered suffices to characterize the motor's torque production versus position and current. The data is used to determine correction factors for the magnitudes of the rotor and stator currents as discussed in the following section. With this data, the possibility to perform open-loop correction opens up.

4.4 The Correction Factor

Once the characterization data is gathered, de-jittered and baseline corrected, Equation 3.31 can be employed to calculate the correction factors. Equation 3.31 is repeated below in Equation 4.11. In this equation, T_0 is the desired constant output torque and $T(\theta, I_{mag}, \mathbf{f}(\theta))$ is the measured torque data. Before performing any calculations, Section 4.4.1 modifies Equation 4.11 slightly to account for a deviation from the ideal quadratic relationship between current and torque. Then, Section 4.4.2 details the procedure for calculating the correction factors. Section 4.5 applies the correction factors and gathers verification data. It also analyzes the performance of the ripple-reduction algorithm. Section 4.6 then examines the performance of the technique in regions that have not been explicitly characterized.

$$\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta)) = \sqrt{\frac{T_0}{T(\theta, I_{mag}, \mathbf{f}(\theta))}} \tag{4.11}$$

4.4.1 A Deviation from the Ideal Motor Behavior

Looking at the plot of mean torque versus current shown in Figure 4-18, the torque appears to obey the quadratic law given in Equation 3.26. The nearly quadratic relationship between torque and current verifies the assumption, made in Chapter 3, that the motor is operating out of saturation. However, when the logarithm of mean torque is displayed versus the logarithm of current as in Figure 4-19, a slight variation from the ideal quadratic law is seen, namely the slope is not as steep as it should

be (the dotted line shows the proper slope). At low currents the deviation can be attributed to the lack of accuracy in the torque measurements. Most probably, at high currents magnetic saturation is acting to reduce the torque. The paragraph below documents the attempt made to account for this slight deviation from ideal behavior. If the deviation is not taken into account, the amount of correction necessary might be underestimated since the torque will not change in relation to the square of the current but some lesser exponent.

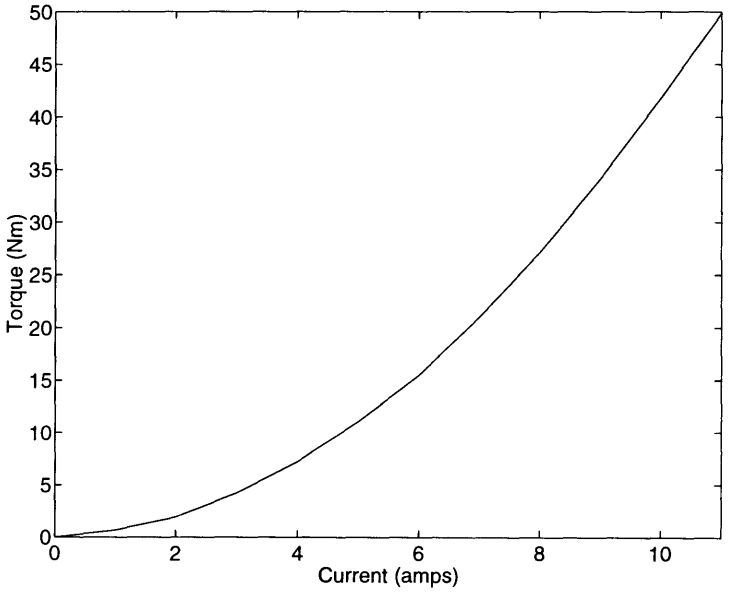

Figure 4-18: Mean Torque versus I_{mag}.

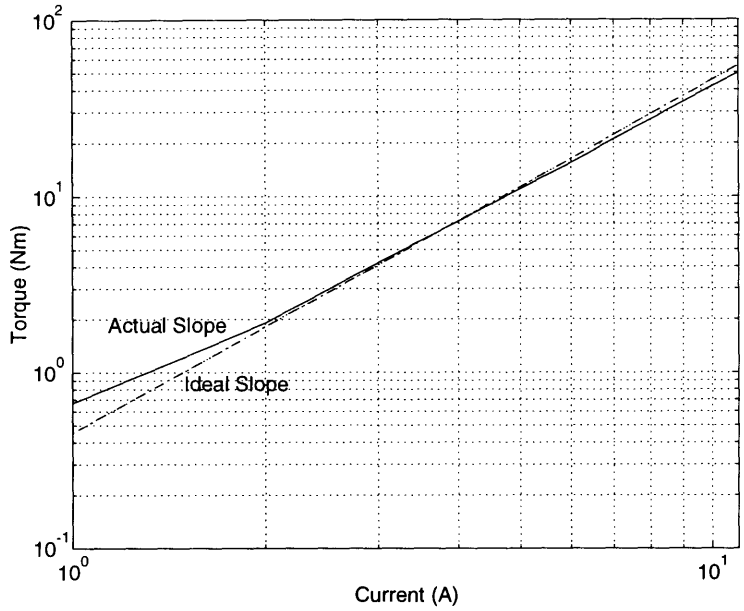

Figure 4-19: Logarithm of Mean Torque Versus the Logarithm of I_{mag}.

To more accurately relate torque to current, Equation 3.26 is modified slightly to reflect the role of saturation. As saturation is neared, the ideal quadratic relationship between current and torque approaches a linear behavior. With this in mind, Equation 3.30 can be modified so that the torque will be a function of the magnitude of the current raised to the power of α. This results in a new torque equation, Equation 4.12. It might appear that this causes a discrepancy in the units of the equation. However, the hidden dependence of the inductance on the current, which causes the non-ideal behavior, also causes the equation to have the proper units.

$$T(\theta, I_{mag}, \mathbf{f}(\theta)) = \left(\frac{1}{2}\mathbf{f}^T(\theta)\frac{\partial \mathbf{L}(\theta)}{\partial \theta}\mathbf{f}(\theta)\right) I_{mag}^{\alpha} \tag{4.12}$$

The variable α is a weak function of current, meaning that in a neighborhood of some current, $\mathbf{i}(\theta) = I_{mag}\,\mathbf{f}(\theta)$, α is constant. Thus when calculating the correction factor it is enough to use the value of α that is valid in that neighborhood. Using elementary calculus, α can be found to be equal to the derivative of the logarithm

of the average torque over the derivative of the logarithm of the magnitude of the current; this is merely the slope of the log-log plot of torque versus current. In this thesis, a fifth-order polynomial fit of torque versus current is made and the values of α are calculated from the fitted curve. At currents ranging from 8 amps to 11 amps, α fell from a peak value of 1.98 to 1.84. Table 4.2 lists the calculated values of alpha versus I_{mag}.

I_{mag}	α
1A	1.7266
2A	1.8266
3A	1.8555
4A	1.8825
5A	1.9135
6A	1.9440
7A	1.9674
8A	1.9762
9A	1.9628
10A	1.9193
11A	1.8367

Table 4.2: α versus Current Level.

Using an analysis similar to that shown in deriving Equation 3.31, a new correction term can be derived. The new correction factor is stated in Equation 4.13, where T_0 is the desired torque. This is the equation for the correction factor that is used in the ripple-reduction algorithm.

$$\tilde{k}_{T_0}(\theta, I_{mag}, \mathbf{f}(\theta)) = \sqrt[\alpha]{\frac{T_0}{T(\theta, I_{mag}, \mathbf{f}(\theta))}} \tag{4.13}$$

4.4.2 Calculating the Correction Factor

Using the de-jittered, baseline-corrected characterization data, the correction factors, $\tilde{k}_{T_0}(\theta, I_{mag})$, can be calculated using Equation 4.13. (When notating the correction factor as $\tilde{k}_{T_0}(\theta, I_{mag})$, it is implied that the current vector is formed by sinusoids of amplitude I_{mag}.) The correction factors can then transformed to the frequency-domain via the DFT. Because of the periodic nature of the torque ripple, the DFT of the correction factor has only a few large terms. An approximation of the correction factors, $\tilde{k}_{T_0}(\theta, I_{mag})$, using the dominant terms, can be created. These dominant terms are the 0, 9, 108 and 216 terms. This approximation saves tremendously on storage space as only a few coefficients need to be stored. To store the entire spatial-domain waveform, at least 432 floating point values need to be stored. Using a dominant-term approach, only seven floating point values need to be stored. This storage savings comes at the cost of additional computation at run-time, however. The form for the correction factor is given by Equation 4.14. Table 4.3 lists the coefficients for each current level.

$$\tilde{k}_{T_0}(\theta, I_{mag}) = m_0 + m_9 \cos(9\,\theta + \phi_9) + m_{108} \cos(108\,\theta + \phi_{108}) + m_{216} \cos(216\,\theta + \phi_{216}) \tag{4.14}$$

	Coefficients						
I_{mag}	m_0	m_9	ϕ_9	m_{108}	ϕ_{108}	m_{216}	ϕ_{216}
4A	1.0006	0.0252	2.2953	0.0384	2.4790	0.0074	0.5083
5A	1.0005	0.0199	2.2879	0.0366	2.4011	0.0068	0.4127
6A	1.0005	0.0173	2.4733	0.0377	-2.5309	0.0066	-3.0037
7A	1.0004	0.0146	2.3560	0.0353	2.4347	0.0064	0.4515
8A	1.0004	0.0128	2.3953	0.0344	2.4515	0.0062	0.5062
9A	1.0003	0.0120	2.4192	0.0334	2.4526	0.0060	0.5394

Table 4.3: Correction Coefficients versus Current Level.

4.5 Verification

To verify the ripple-correction technique, the following experiment is conducted. Using the characterization data, the mean torque is calculated at current levels ranging from four to nine amperes. These mean torques are then selected as the desired constant output torque level to be produced by the motor. The experiment then attempts to produce the desired constant torques using the correction factors.

In order to verify that the motor can be commanded to produce a constant torque, the correction factors calculated in Section 4.4.2 are used to amplitude modulate the phase currents as compared to the original characterization runs. The torque is measured as the motor rotates. The measured torque is then de-jittered and corrected for zero-current torque. This result is the torque produced solely by the action of the motor. The desired result at this point should be a constant torque output. It is observed that modulating the three-phase currents on rotor and stator achieves significantly reduced torque ripple. Figure 4-20 plots the ripple-corrected data.

Table 4.4 lists many results of this experiment. The first column lists the desired torque level. The second column lists the mean torque level that has been achieved. The third column lists the original root-mean-square (RMS) ripple. The fourth col-

umn lists the RMS error from the desired mean torque level. The fifth column lists the RMS ripple in the reduced ripple output. It should first be noted that the technique has a significant error in the achieved value of mean torque relative to the desired value of torque. The cause of this error is not yet explained. The effect of the error in the achieved mean torque is to create a difference between the RMS error from the desired value and the RMS ripple. If the achieved mean torque value equaled the desired torque value, then the error and the ripple would be the same.

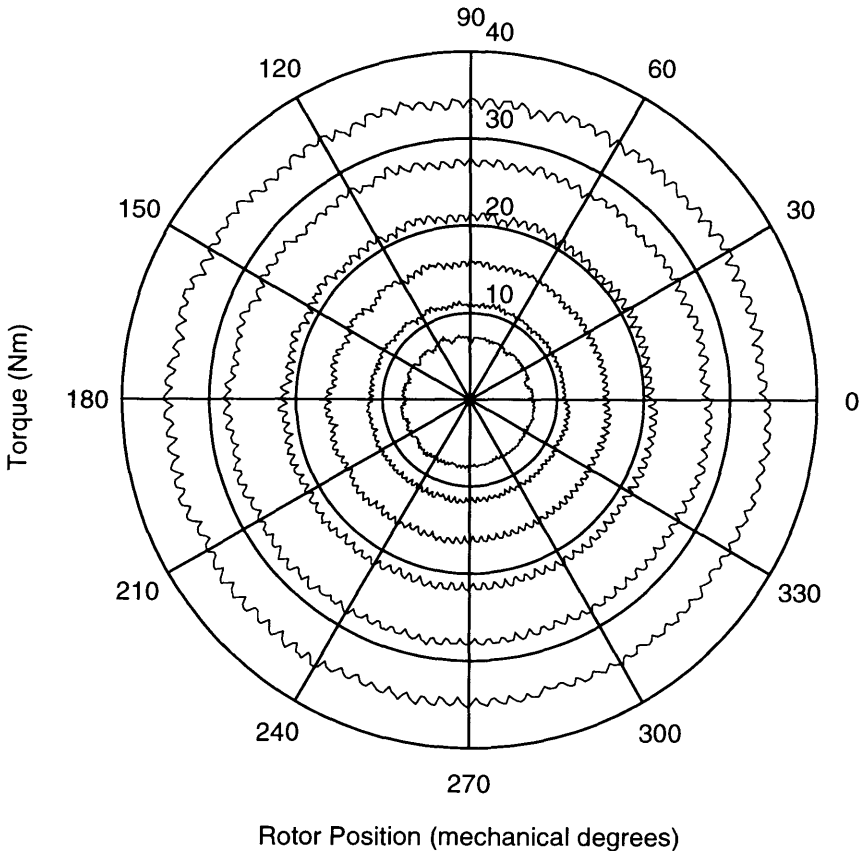

Figure 4-20: Ripple-reduced Torque Data.

Desired Mean Torque	Achieved Torque	Original RMS Ripple	Achieved RMS Error	Achieved RMS Ripple
7.218	7.432	0.485	0.392	0.329
11.058	11.302	0.664	0.436	0.361
15.507	15.962	0.971	0.590	0.376
21.045	21.259	1.151	0.494	0.445
27.097	27.613	1.484	0.665	0.419
34.108	34.450	1.720	0.617	0.515

Table 4.4: Results of Ripple Reduction.

A brief comment can be made about the choice of the metric for comparison. A number of metrics are possible: RMS ripple, RMS ripple as a percentage of mean torque, and peak-to-peak ripple. In this thesis, the choice is made to look at RMS ripple and RMS ripple as a percentage of mean torque. There are at least two reasons for this decision. Since the technique operates near the quantization limit of the measurement system, a peak-to-peak metric would inherit the considerable granularity of the measurement, and the results would be obscured. A second reason is that since it is empirical data that is being dealt with, it makes sense to talk about the distribution. The RMS error is the standard deviation of the measurement and so it is a natural metric for analyzing the data. For these reasons, the RMS ripple and RMS ripple as a percentage of mean torque are chosen as the performance metrics.

Expressing the RMS error of the results above as a percentage of mean torque, the improvement becomes more tangible. Table 4.5 lists the original ripple as percentage of desired torque and the ripple-reduced ripple as a percentage of achieved torque. At higher torque values, there is an improvement of 3.5%, which translates into slightly more than a factor of three reduction. At lower torque values, the improvement is not tremendous. In agreement with the comments of Section 4.3.1, any real improvement is obscured by the measurement error. By scanning the fifth column of Table 4.4 it can be seen that the RMS ripple is of the same magnitude as the

errors in the measurement system. The effectiveness of the technique is being limited by the measurement system. This also accounts for the apparent improvement in the percentage ripple as the mean torque value increases. It is conceivable that with a more accurate measurement system or at higher torque values, the technique would be able to achieve a ripple of less than 1%.

Desired Torque	Original RMS Ripple	Achieved Torque	Achieved RMS Ripple
7.218	6.7%	7.432	4.4%
11.058	6.0%	11.302	3.2%
15.507	6.3%	15.962	2.4%
21.045	5.5%	21.259	2.1%
27.097	5.5%	27.613	1.5%
34.108	5.0%	34.450	1.5%

Table 4.5: Torque Ripple as a Percentage of Mean Torque.

4.6 Testing the Practicality

Seeing that the technique is reasonably successful at reducing torque ripple in well characterized regions of operation, it is desirable to see how well it performs in areas of operation that are not characterized. In order to quantify this, an attempt can be made to produce constant torque at what should be the mean torque values with current amplitudes of 4.5, 5.5, 6.5, 7.5, and 8.5 amps: 8.817, 12.943, 17.907, 23.728 and 30.383 Nm. To accomplish this, the correction factors are chosen to be the average of the two nearest known correction factors. For example, to generate a correction factor around 4.5 amps, the correction factors for 4 and 5 amps are averaged. Data is then taken for a single revolution in the counter-clockwise direction, at each current level.

Figure 4-21 plots the results of the experiment described above. Table 4.6 tabu-

lates the metrics of performance. The results look quite good and are on par with those obtained in regions where the torque production is characterized. There is an unexplained error in the mean value again. The first column of the table lists the desired torque, and the second column lists the achieved mean torque. The third column lists the RMS error from desired torque value and the fourth column lists the RMS ripple around the achieved torque value. The fifth column lists the ripple as a percentage of the achieved mean torque.

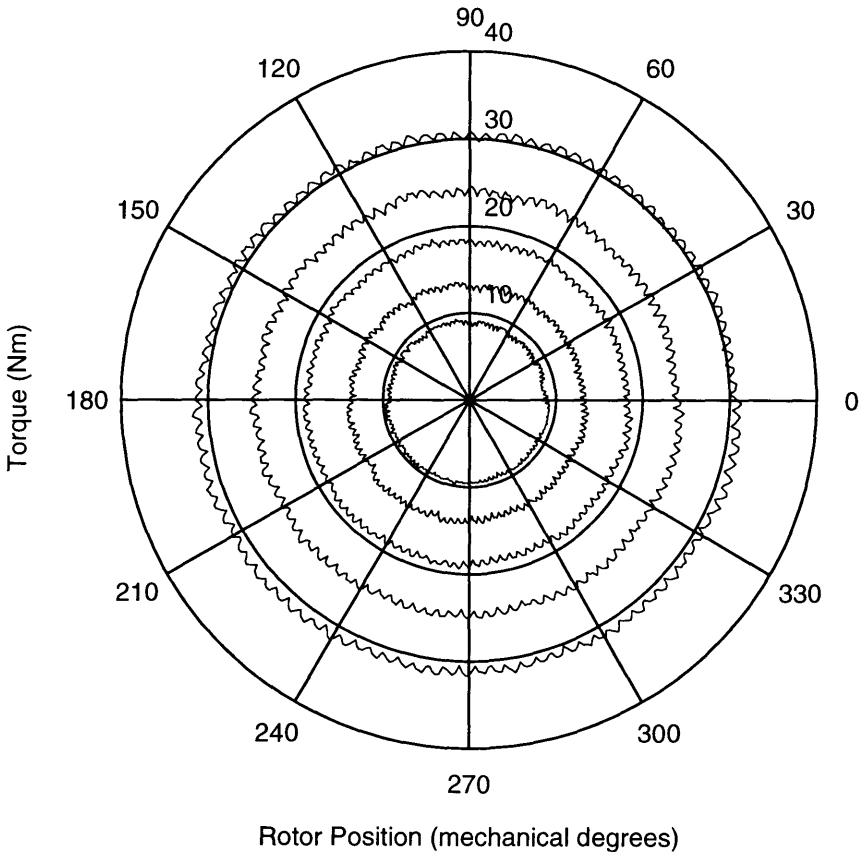

Figure 4-21: Ripple-reduced Torque Data.

Desired Torque	Achieved Torque	Achieved RMS Error	Achieved RMS Ripple	Achieved RMS Ripple
8.817	9.206	0.502	0.318	3.4%
12.943	13.544	0.729	0.412	3.0%
17.907	18.543	0.736	0.370	2.0%
23.728	24.355	0.777	0.458	1.9%
30.383	30.854	0.670	0.476	1.5%

Table 4.6: Results of Ripple Reduction in Uncharacterized Regions.

4.7 Summary of Results for the Ripple-Reduction Algorithm

This chapter has documented the ripple-reduction algorithm. The basic data flow is depicted in Figure 4-1. The data-flow begins with the collection and processing of characterization data. This chapter documents the data gathered as well as the processing steps which include a compensation for aperiodic sampling and a removal of zero-current torque components. Once the characterization phase is completed, the correction factors are calculated. Section 4.4.1 develops a modification to the basic equation for calculating the correction factor, including an extra parameter to account for a non-ideal torque versus current relationship. Section 4.4.2 then documents the actual method used for calculating the correction factor and how to implement the correction factor with a minimum amount of data storage. Sections 4.5 and 4.6 document the final phase of the data flow which is the verification phase. This phase applies the correction factors and measures the resulting data. This data is then processed to account for aperiodic sampling and zero-current torque.

The results of the ripple-reduction technique demonstrate a best-case reduction of torque ripple from 1.72 Nm to .51 Nm. Relative to the mean torque values this is a reduction from a level of 5.0% to 1.5%. The magnitude of the residual ripple is com-

parable to the resolution of the torque measurement system. The ripple-reduction technique is also applied via interpolation to areas where the characterization has not been done. Results in these areas demonstrate ripple as low as 1.5%, which is equal to the best value obtained in the characterized region. Thus, an effective ripple-reduction technique has been demonstrated. While depending upon characterization, the technique operates successfully in regions where the characterization is not explicitly carried out.

The following chapter investigates the possibility of rotating the magnetic fields in the motor in such a way as to maintain constant torque at a fixed position while spreading power dissipation among the windings. The technique of that chapter is combined with the ripple-reduction technique of this chapter to obtain the benefits of both.

Chapter 5

Thermal Management

5.1 Introduction

Often a limitation on torque output comes from the inability to radiate or conduct heat away from a motor quickly enough. In particular there is the concern of creating hot spots in the motor while providing constant torque. These hot spots can cause insulation breakdown or mechanical failure. Normally, to provide constant torque at a fixed position, the windings must conduct constant currents, and the winding handling the most current will heat up more than the rest. This heating condition provides a limit on how much current can be delivered without overheating the motor. The problem is not easily solved since a reduction in current normally implies a reduction in torque.

However, with the motor used in this thesis, there exists the possibility of providing constant torque without using constant currents. Since the motor is constructed of two symmetrical "pancake" halves, each with a set of windings, it is possible to electrically commutate the stator and rotor currents while maintaining a constant torque output at a fixed position. This can be done by maintaining a fixed electrical angle between the rotor and stator phase currents in such a way that the rotor and stator fields are separated by a constant angle. If the period of the rotor and stator current commutation is much less than the thermal time constants, the peak temperature in each location in the motor will be only a function of the average power dissipation in

that region as opposed to the peak dissipation. This rotation of the phase currents can increase the allowable power dissipation in the motor by some amount, which translates into a proportional increase of the torque production capability.

This chapter proceeds by detailing a thermal model of the motor in Section 5.2. Using this model, Section 5.3 examines the difference between constant and time-varying currents in the windings in regard to peak temperature production. Section 5.4 verifies the assumptions made in formulating the thermal-management technique. Section 5.5 provides a characterization of the motor's thermal properties and Section 5.6 provides data supporting the success of the thermal-management technique. In Section 5.7, the technique is combined with the ripple-reduction technique and supporting data for their combined effectiveness is provided.

5.2 A Thermal Model

The motor used for this thesis, a cross-section of which is shown in Figure 5-1, has two symmetrical halves, one acting as a rotor and one as a stator. Windings in either half dissipate power proportional to the winding resistance and the square of the current through the winding. The motor has two mechanisms for removing generated heat. The first mechanism is conduction to a cooling channel in the back-iron which has cool water flowing in it. The second method for cooling is conduction through the epoxy encasing the windings and then convection to the ambient air.

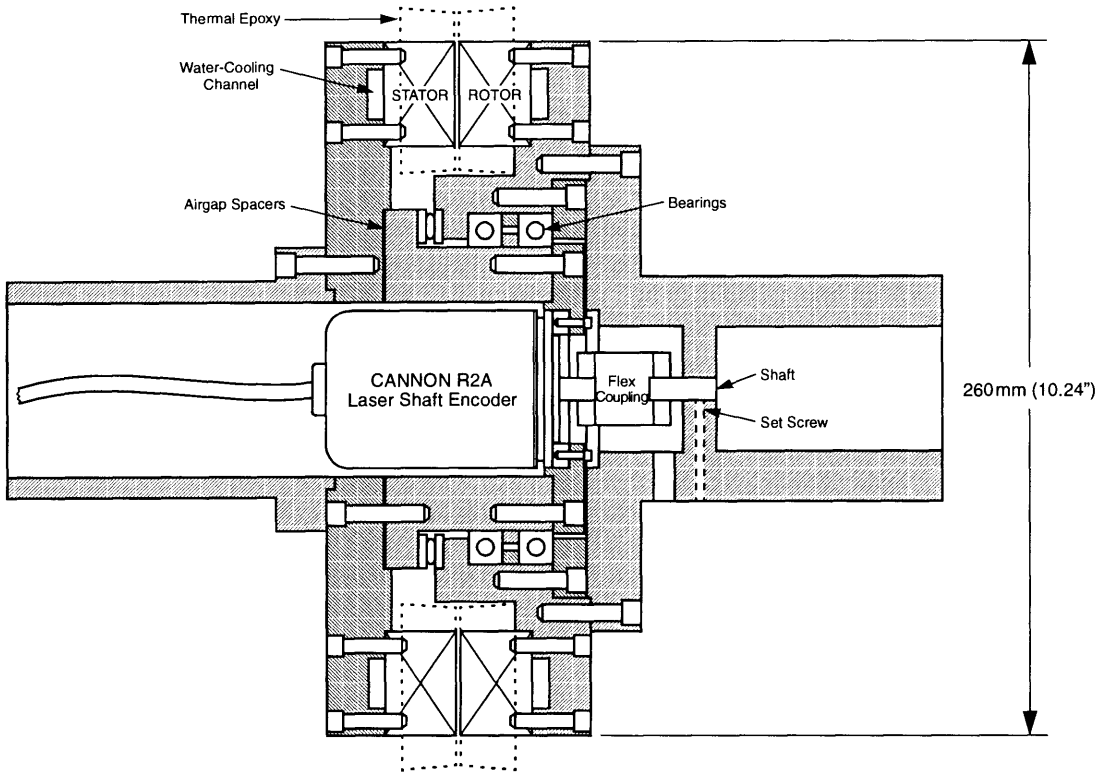

Figure 5-1: A Cross-section of the Axial Flux Motor.

This section will model the thermal behavior at any location in the motor using thermal impedances and power sources. In this form of modeling, temperatures correspond to voltages and power sources to currents. The impedances relate temperature differences to power flow. For this model, it is assumed that there is no heat flux between the rotor and stator and this is verified in Section 5.4. Once this assumption is made, the thermal model be used to consider the thermal properties of the rotor or stator, depending on the location of the point of interest. It can be noted that some fraction of the power dissipated by each winding arrives at the location of interest. Thus, the power flow into the point is equal to a linear combination of the power dissipations in the windings. Equation 5.1 states this mathematically, where P_A, P_B and P_C are the dissipations in windings A,B, and C respectively. The coefficients a,b, and c depend on the location in the motor. If the power dissipation is time-varying, the coefficients may also be dependent on the frequency of the power dissipation.

$$P_{in} = a\,P_A + b\,P_B + c\,P_C \tag{5.1}$$

The power flow into the point of study must also flow out, eventually to the cooling channel or to the ambient air. This power flow is supported by a temperature gradient between the point and the cooling mechanisms. The relationship between this temperature gradient and the power flow can be described by adding thermal impedances to the model between the point and both the cooling channel and the ambient air. Figure 5-2 provides a pictorial representation of the model. In the figure, T_{air} is the temperature of the ambient air, T_{water} is the temperature of the water in the cooling channel, and T_{point} is the temperature at the location of interest. θ_{air} and θ_{water} are the thermal resistances from the point to the ambient air and cooling channel, respectively. Similarly, C_{air} and C_{water} are the thermal capacitances from the point to the ambient air and cooling channel.

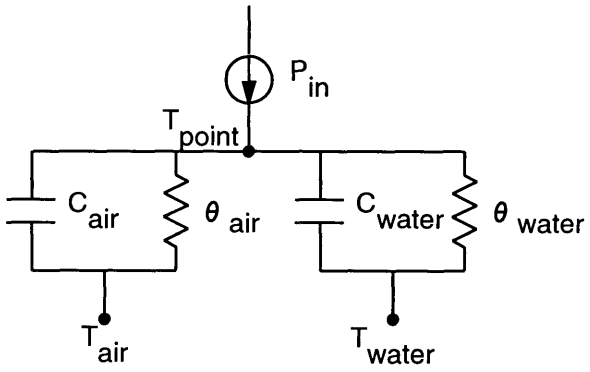

Figure 5-2: A Pictorial Representation of the Thermal Model.

In this model, the thermal impedances are formed by a lumped thermal resistor and a lumped thermal capacitor. A few qualitative comments can be made as supporting evidence for the validity of this choice. The use of a thermal resistance asserts that the steady-state temperature differential from the point to the temperature

reservoir is proportional to the heat flow between these two points. A capacitive term enters in when the heat capacity of the material is considered. The effect of heat capacity is that the material conducting the heat away from the dissipation source will itself absorb some of the heat to effect a temperature change. A small digression is necessary to describe exactly how the heat capacity behaves as a capacitive element.

Elementary thermodynamics defines the heat capacity of an object as the ratio of its change in heat to its change in temperature. Equation 5.2 provides a mathematical definition, where $\mathrm{d}Q$ is an incremental amount of heat, $\mathrm{d}T$ is an incremental temperature and C is the heat capacity.

$$\mathrm{d}Q = C\,\mathrm{d}T \tag{5.2}$$

The incremental addition of heat can be expressed as a flow of power, P, into the substance for an incremental amount of time, dt. Substituting this into Equation 5.2 and re-arranging terms, yields Equation 5.3.

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{P}{C} \tag{5.3}$$

The form of this equation lends itself to immediate analysis of the effective thermal impedance of the substance, which looks capacitive. Equation 5.4 provides the Laplace-transform form of the thermal impedance resulting from the heat capacity.

$$Z = \frac{1}{Cs} \tag{5.4}$$

Since some of the power flow will divert into heating the conductive material, the heat capacity should be modeled as an impedance in parallel with the resistive impedance of the substance. Appropriate impedances to the cooling channel and the ambient air have been included in the thermal model, as depicted in Figure 5-2. This

completes the thermal model of the motor. The following section uses this model to analyze the temperature at the point of study for various phase currents.

5.3 Employing the Thermal Model

Using the model set up in the previous section, this section studies the temperature of an arbitrary point in the motor when the currents in the windings are three-phase sinusoids. It is found that as the frequency of the sinusoids increases, the peak temperature at every point in the motor decreases. This fact gives motivation for the thermal-management technique, which at a fixed rotor position provides time-varying three-phase currents to both the rotor and the stator.

The currents in the windings are chosen to be three-phase sinusoids as used in previous chapters. Equations 4.1–4.6 dictated these currents earlier. Since the problem now under consideration is equivalent for the rotor and stator, the equations for the phase currents can be chosen to be either the rotor or the stator currents. For simplicity the equations for the rotor are used. These are repeated below in Equations 5.5–5.7. It should be remembered that the variable δ is a free variable. I_{mag} is the amplitude of the phase current sinusoid.

$$i_A = I_{mag}\cos(\delta) \tag{5.5}$$

$$i_B = I_{mag}\cos(\delta - \frac{2\pi}{3}) \tag{5.6}$$

$$i_C = I_{mag}\cos(\delta - \frac{4\pi}{3}) \tag{5.7}$$

The phase currents defined above yield power dissipations for each winding as listed in Equations 5.8–5.10. R is the electrical resistance of each winding. For these equations, δ is chosen to have a frequency-dependent term and a phase angle, $\delta = \omega t + \gamma$.

$$P_A = I_{mag}^2 R \left[\frac{1}{2} + \frac{1}{2}\cos(2\omega t + 2\gamma)\right] \tag{5.8}$$

$$P_B = I_{mag}^2 R \left[\frac{1}{2} + \frac{1}{2}\cos(2\omega t + 2\gamma - \frac{4\pi}{3})\right] \tag{5.9}$$

$$P_C = I_{mag}^2 R \left[\frac{1}{2} + \frac{1}{2}\cos(2\omega t + 2\gamma - \frac{2\pi}{3})\right] \tag{5.10}$$

Using the expressions for the power dissipations in the windings, the power flow into the point of interest is written out in Equation 5.11. In this equation, the frequency dependencies of the coefficients a, b, and c are made explicit.

$$\begin{aligned} P_{in} =& \frac{1}{2} I_{mag}^2 R \left[a(0) + b(0) + c(0)\right] \\ &+ \frac{1}{2} I_{mag}^2 R \left[a(2\omega)\cos(2\omega t + 2\gamma)\right. \\ &\qquad + b(2\omega)\cos(2\omega t + 2\gamma - \frac{4\pi}{3}) \\ &\qquad \left. + c(2\omega)\cos(2\omega t + 2\gamma - \frac{2\pi}{3})\right] \end{aligned} \tag{5.11}$$

With Equation 5.11 it is possible to find the maximum power flux into the point with a static distribution of currents, that is, $\omega = 0$. This can be done using standard maximization methods from calculus as shown in Appendix B. Equation 5.12 expresses this maximum power flux. This power flux will produce a temperature given by Equation 5.13.

$$\begin{aligned} P_{in} =& \frac{1}{2} I_{mag}^2 R \left[a(0) + b(0) + c(0)\right] \\ &+ \frac{1}{2} I_{mag}^2 R \sqrt{a(0)^2 + b(0)^2 + c(0)^2 - a(0)b(0) - b(0)c(0) - a(0)c(0)} \end{aligned} \tag{5.12}$$

$$T_{point} = T_{air}\frac{\theta_{water}}{\theta_{air}+\theta_{water}} + T_{water}\frac{\theta_{air}}{\theta_{air}+\theta_{water}} + \frac{1}{2}I_{mag}^2 R\,[a(0)+b(0)+c(0)]\frac{\theta_{air}\theta_{water}}{\theta_{air}+\theta_{water}} + \frac{1}{2}I_{mag}^2 R\sqrt{a(0)^2+b(0)^2+c(0)^2-a(0)b(0)-b(0)c(0)-a(0)c(0)}]\frac{\theta_{air}\theta_{water}}{\theta_{air}+\theta_{water}} \quad (5.13)$$

It is also possible to solve for the temperature change when the frequency of the sinusoidal currents becomes large relative to the thermal time constants of the motor. In this case, the thermal impedances to the cooling sources approach zero, while the power flow into the point remains bounded. In this way the sinusoidal power does not contribute to the temperature at the point of interest. Only the direct current component of the power flow creates a temperature change. Equation 5.14 expresses the temperature at the point of interest when the frequency, ω, is much larger than the thermal time constants.

$$T_{point} = T_{air}\frac{\theta_{water}}{\theta_{air}+\theta_{water}} + T_{water}\frac{\theta_{air}}{\theta_{air}+\theta_{water}} + \frac{1}{2}I_{mag}^2 R\,[a(0)+b(0)+c(0)]\frac{\theta_{air}\theta_{water}}{\theta_{air}+\theta_{water}} \quad (5.14)$$

Taking the difference between Equations 5.13 and 5.14 yields the temperature difference between the worst-case static distribution of currents and the high-frequency, time-varying distribution of currents. This is given in Equation 5.15. For a typical motor, the current distributions are static at a fixed position. This implies that for over the range of possible rotor positions, each point experiences a peak temperature change. For the motor used in this thesis however, it is possible to produce torque at a fixed position while maintaining time-varying currents in the rotor and the stator. If the frequency of commutation is sufficiently high, the temperature peaks can be eliminated, and the temperature experienced corresponds to that generated by the

average power flow. Because this result has been derived for an arbitrary location in the motor, it is valid for every location. Thus time-varying currents can be used to eliminate temperature peaks while still producing torque at a fixed position.

$$\Delta T = \frac{1}{2} I_{mag}^2 R \sqrt{a(0)^2 + b(0)^2 + c(0)^2 - a(0)b(0) - b(0)c(0) - a(0)c(0)} \frac{\theta_{air}\theta_{water}}{\theta_{air} + \theta_{water}} \tag{5.15}$$

5.4 Assumptions about Thermal Behavior

Before proceeding any further, the assumptions that underlie this thermal-management technique should be verified. First, it has been assumed that the rotor and stator heat independently. To verify this, an experiment is created in which the amplitude of the three-phase sinusoidal stator current is fixed at 8 amps. The stator-current phase angle is swept from 0° to 330° in 30° increments, allowing the motor to reach thermal equilibrium at each angle. Temperature is measured at four thermocouples. Appendix I gives more information on the temperature measurement system. The temperature data is presented in Table 5.1. Ambient temperature is approximately 22°C. The rotor thermocouples are notated as T1 and T2 and the stator thermocouples are T3 and T4. It is clear from the data that there is no phase angle at which the thermal coupling is significant. Thus, for the purposes of this thesis, the rotor and stator heating occur independently. However, it should be noted that the coupling could increase if the air-gap distance were reduced.

Phase Angle(°)	T1(°C)	T2(°C)	T3(°C)	T4(°C)
0	23	22	44	50
30	22	22	46	50
60	23	23	49	55
90	23	22	50	58
120	23	22	49	59
150	23	22	46	56
180	23	22	46	53
210	22	22	46	51
240	22	22	47	53
270	22	22	48	56
300	23	23	47	57
330	23	22	44	53

Table 5.1: Thermocouple Temperatures versus Stator-Current Phase Angle.

Another assumption of the work so far has been that the phase currents can be rotated much faster than the thermal time constant. Figure 5-3 presents temperatures at the rotor thermocouple locations when all windings are driven with 6 amps. The temperatures are measured versus time. It can be seen that the thermal time constant is on the order of 3 minutes. Thus, if the phase currents can be driven at a frequency much greater than .0056 Hz, the thermal-management technique should be successful in avoiding temperature peaks.

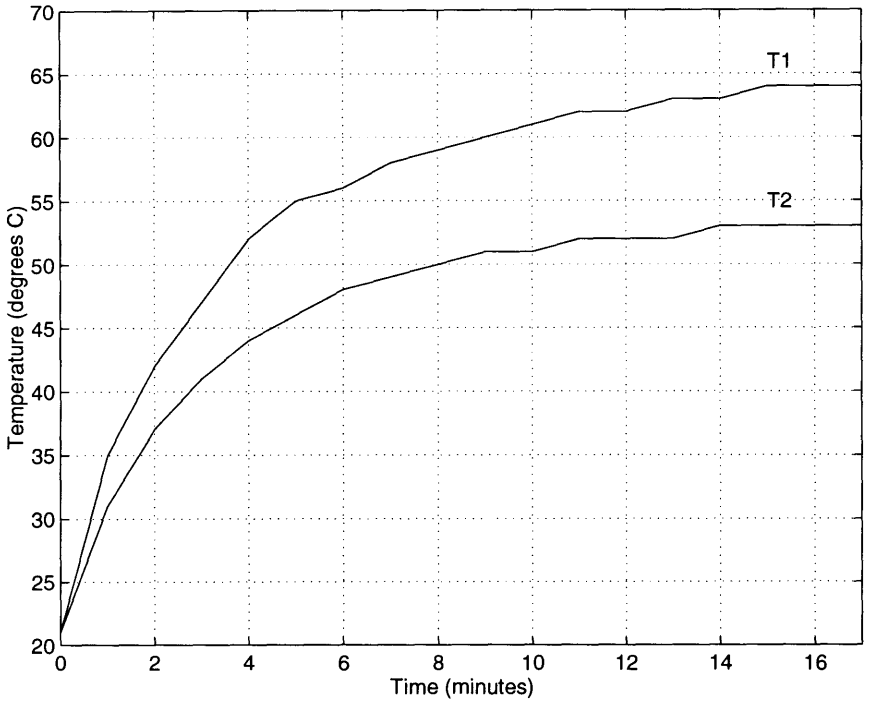

Figure 5-3: Thermocouple Temperatures versus Time.

5.5 Thermal Characterization

In order to anticipate what type of gains can be achieved by rotating the phase currents, it is necessary to arrive at some estimate of the difference between the peak temperatures and the average temperatures. This can be done for any one of the four thermal couples in the motor. Figure 5-4 provides temperature data from the T1 thermocouple on the rotor when the phase current magnitude is set to 8 amps and the phase angle is swept. The best-fit sinusoidal approximation to this data yields an average value of 50.76°C and an amplitude of 2.98°C. From this data, it would be expected that rotating the 8 amp phase currents at a frequency much faster than the thermal time constants would yield a constant temperature of 50.76°C at thermocouple T1. This eliminates about 3°C of temperature increase, or about 10% of the temperature rise above ambient, 22°C. Using the linear model of the previous sections, it would be expected that at higher currents, the same percentage

reduction in the temperature rise would occur, provided the non-linearities of non-linear phenomena, such as convection, remain negligible. Furthermore, the thermal model predicts that if the motor had points with strong coupling to the dissipations of only one or two phases, the effectiveness of the technique would be enhanced. As it is, it appears that there is fairly uniform coupling to each phase.

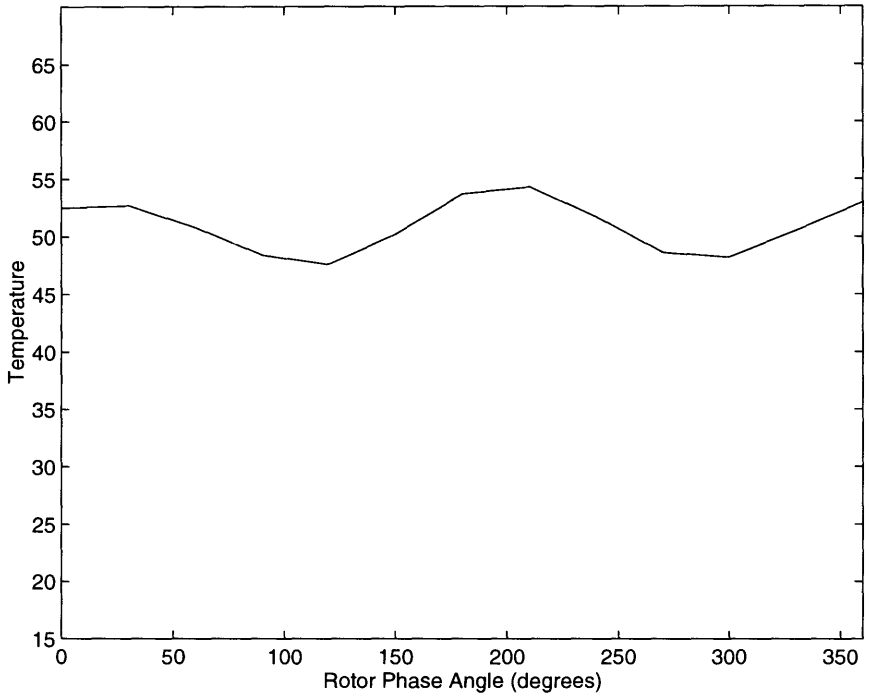

Figure 5-4: Rotor Temperature versus Phase Angle.

5.6 The Thermal-Management Technique

As discussed in Section 5.3, if the phase currents are rotated quickly enough, temperature peaks are avoided. This section provides data supporting the validity of the technique. An experiment is run in which the rotor and stator are driven with 10 amp magnitude three-phase currents. Two scenarios are compared, similar to the scenarios discussed in Section 5.3. In the first scenario, the motor position is locked at the zero position. The rotor-current phase angle is fixed at zero degrees, which is close to the angle which causes the peak dissipation as seen at the rotor thermocouple, T1.

The stator-current phase angle is set to maximize torque output. The steady-state temperature at each thermocouple is then recorded. The second scenario still locks the motor position at zero, however, the rotor and stator-current phase angles rotate at approximately 18 Hz while maintaining a constant phase angle between them. The steady-state temperatures are then recorded.

Table 5.2 compares the steady-state temperatures for each experiment. The third data column shows that the steady-state temperature goes down by a few degrees at each thermocouple. But more telling is that the temperature is reduced at thermocouple T1, where the heating would be at its maximum if the currents were not rotating. In Section 5.5, it has been found that at 8 amps and a rotor current phase angle of zero degrees, the temperature at thermocouple T1 was slightly less than 3°C above the mean temperature taken when the phase currents rotate. Translating this into an equivalent expected reduction at 10 amps, one could anticipate a change of slightly less than 4.7°C. Experimental data shows a change of 4.2°C, confirming the expectations. Thus, the thermal-management technique successfully reduces the temperature and does so in a predictable way. This technique can be folded into the ripple-correction algorithm as discussed in the next section.

Thermocouple	Non-Rotating Temp. (°C)	Rotating(°C)	ΔT(°C)
T1	80.7	76.5	-4.2
T2	67.1	63.1	-4.0
T3	66.5	61.7	-4.8
T4	80.0	74.6	-5.4

Table 5.2: Thermocouple Temperatures with Non-Rotating and Rotating Phase Currents.

5.7 A Unification of Ripple Reduction and Thermal Management

It is possible to combine the ripple-reduction and thermal-management techniques. The idea is that the phase currents should rotate to minimize peak temperature while the magnitudes are modulated to correct for ripple. At the outset, it is not clear whether ripple depends on the absolute phase-current angles or not. To get any idea whether combining techniques would work, it is first necessary to characterize the torque ripple versus position and phase angle.

Data has been taken for torque versus position and phase angle with fixed rotor and stator-current magnitudes of 8 amps. The magnitude and phase of significant ripple harmonics are listed in Table 5.3 versus the phase angle of the currents. It can be seen that all but the ninth harmonic of torque ripple are of fixed magnitude and phase. The ninth harmonic however, rotates with the phase angle of the currents. It has been discovered that the rotation of the ninth harmonic is caused by harmonic distortion in the current waveforms produced by the power electronics. Therefore, it makes perfect sense that this ripple component rotates along with the phase angle of the currents. There have been no efforts to re-do the experiment, as the distortion can be compensated for quite successfully, as the results demonstrate.

	9th Harmonic		108th Harmonic		216th Harmonic	
Phase Angle	Magnitude(Nm)	Phase	Magnitude	Phase	Magnitude	Phase
0	0.343	133	0.926	41	0.180	144
36	0.352	167	0.931	38	0.187	139
72	0.352	206	0.930	41	0.178	143
108	0.335	244	0.928	42	0.180	144
144	0.358	280	0.908	40	0.181	142
180	0.384	318	0.928	32	0.182	127
216	0.377	353	0.945	42	0.182	145
252	0.385	29	0.923	42	0.176	146
288	0.350	63	0.906	41	0.175	142
324	0.353	98	0.891	40	0.181	142

Table 5.3: Magnitude and Phase of Ripple Harmonics versus Current Phase Angle.

Using the information in the table above, a ripple-compensation algorithm is constructed in which the magnitude and phase angle of the 108th and 216th harmonic components of the correction factor are fixed. The 9th harmonic component is fixed in magnitude with a phase that tracks the phase current rotation appropriately. This correction algorithm is applied and ripple data is collected versus current phase angle. The corrected ripple waveform is spectrally decomposed and the results are reported in Table 5.4. These results show an appreciable reduction in ripple, on par with results reported in Chapter 4. Thermally, there is a slight temperature dependence on position that is caused by the modulating magnitudes of the phase currents, yet the advantage of phase current rotation is not diminished. Thus, the thermal-management technique of rotating the fields can be merged with the ripple-reduction technique to achieve the advantages of both. Torque ripple and peak temperatures are reduced.

	9th Harmonic		108th Harmonic		216th Harmonic	
Phase Angle	Magnitude(Nm)	Phase	Magnitude	Phase	Magnitude	Phase
0	0.044	331	0.202	334	0.078	85
36	0.044	4	0.235	327	0.093	74
72	0.041	46	0.207	329	0.080	79
108	0.034	72	0.215	331	0.078	81
144	0.042	105	0.185	333	0.072	86
180	0.042	136	0.259	328	0.094	81
216	0.046	182	0.221	329	0.075	79
252	0.044	206	0.195	333	0.077	86
288	0.052	246	0.209	330	0.083	81
324	0.042	288	0.224	326	0.086	81

Table 5.4: Magnitude and Phase of Corrected Ripple Harmonics versus Phase Angle of Current.

Chapter 6

Summary, Conclusions, and Suggestions for Future Work

This thesis presents a method for torque-ripple compensation which can be seamlessly folded into a thermal-management technique. The torque-ripple reduction is performed without the aid of torque feedback. Experimentation is done using a synchronous axial-air-gap motor with a three-phase rotor and three-phase stator. The ripple-correction method uses a magnitude modulation of balanced three-phase currents to achieve a 70% reduction of the Root-Mean-Square (RMS) torque ripple. The proper magnitude modulation is determined from empirical measurement of the ripple harmonics. The thermal-management technique uses the idea of rotating the rotor and stator phase currents versus time, while maintaining a constant angle between the fields. This distributes the generated heat throughout the motor, even while operating at a fixed position. This technique proves to work as expected; however, the gains are minimal, resulting in about a 5°C temperature reduction. Finally, these two techniques are combined. The results of the combined algorithm demonstrates ripple reduction on par with results achieved using only ripple reduction. The paragraphs below outline the achievements of each chapter. Following this are some ideas for continuing research.

6.1 Summary

Chapter 3 presents an inductance-based model of torque production. Using the assumption that the magnetics behavior linearly, the model torque is shown to be a function of rotor position when the phase currents are functions of position. More importantly, the model torque is shown to scale quadratically with a linear scaling of the phase currents. This equation for torque production is given in Equation 3.28. It is manipulated to show that if the phase currents are modulated proportionally to the square root of the ratio of the desired torque to the measured torque at a position, then the torque ripple is eliminated. Equation 3.31 provides an expression for this correction factor. The technique requires a characterization of torque production versus position before the ripple-reduction technique can be employed.

Chapter 4 documents the measurement procedure and provides data supporting the success of the ripple-reduction method. Section 4.3 documents the possible sources of error in the measurement technique which include quantization error in the dynamometer, aperiodic sampling of the torque data, and zero-current forces on the motor. Section 4.5 reports the success of the ripple-reduction technique in reducing the RMS torque ripple. Table 4.4 lists ripple levels versus mean torque for the corrected and uncorrected cases. The best results are at the highest mean torque level, 34.1 Nm, where the torque ripple is reduced from 1.72 Nm to 0.51 Nm. Taking the ripples as percentages of mean torque, the ripple is reduced from 5.0% to 1.5%. At lower torque levels, the improvements are less significant, being bounded by the dynamometer resolution of about 0.4088 Nm as documented in Section 4.3.1. Section 4.6 concludes Chapter 4 with an examination of the technique in regions where the torque production is not characterized. The results listed in Table 4.6 indicate that the technique performs as well in characterized regions as it does in uncharacterized regions. For a mean torque of approximately 30 Nm, the ripple is reduced to a level of 0.4757 Nm.

Chapter 5 concerns itself with the issue of hot-spot creation during static loading conditions. Taking advantage of the unique design of the motor used in this thesis,

a technique is developed which rotates both the rotor and the stator currents while holding the rotor position fixed. This prevents all locations in the motor from experiencing the peak temperatures that would occur if the currents did not rotate. However, the gains are minimal. Sections 5.2 and 5.3 develop a model of the temperature of an arbitrary location in the motor. When the windings conduct constant currents, Equation 5.13 dictates the peak temperature of an arbitrary point in the motor. When the windings conduct sinusoidal currents, Equation 5.14 expresses the peak temperature. For the case in which the phase currents are constant, the peak temperature is higher than for the case in which they conduct sinusoidal currents. Thus the rotation of the phase currents while holding a static position reduces the peak temperature experienced at every point in the motor. For phase-current magnitudes of 10 amps, Section 5.6 predicts a temperature drop of 4.7°C as measured by a certain thermocouple. The section documents an actual decrease of 4.2°C. Thus the thermal-management technique reduces temperature peaks in a predictable manner.

Finally Chapter 5 combines the ripple-reduction technique and the thermal-management technique into one algorithm. Both techniques are compatible with each other. Data is presented in Section 5.7 to demonstrate that the thermal-management technique does not compromise the ripple-reduction technique.

6.2 Conclusions

The ripple-reduction technique provides a 3.5% decrease in the RMS torque ripple. A distinguishing feature of the technique is that it does not depend on torque feedback, but only on position feedback. This eliminates the need for a torque sensor during normal operation. Eliminating the torque sensor reduces cost, and avoids the introduction of additional system dynamics. The lack of a torque sensor does not seem to be an enormous handicap, provided that accurate characterization measurements can be made. Compared to this technique, other researchers using open-loop techniques have observed similar reductions in torque ripple [8][11][12]; however, these gains have been on other types of machines.

The thermal-management technique demonstrates only minimal reductions in peak temperatures. Because of the distributed nature of the windings and the multi-pole design, there is not the significant localization of the heating which causes hot spots to form. Thus, rotation of the phase currents does not cause a large change in zero-frequency power flow into a given point, as would be hoped. It is expected that the technique would show much greater success with a motor having fewer poles. However, it is note-worthy that the slight changes that are observed in peak temperatures do follow the predictions of the thermal model developed in this thesis.

6.3 Areas for Future Research

The full potential of the ripple-reduction technique has not yet been explored because of limitations in the experimentation system. The resolution of the dynamometer poses the greatest limitation. This is further hindered by the coupling of switching noise from the servo amplifier into the measurement and control systems. Future experimentation systems can be improved by selecting a more accurate dynamometer and by eliminating switching noise and its effects.

The techniques discussed in this thesis allot some flexibility to the motor designer. Future motors should be designed with these techniques in mind. The ripple-reduction technique may make it feasible to obtain low ripple without using distributed windings, or at least fewer of them. The benefits of this could be an increase in torque-to-mass ratio, a decrease in volume, a reduction in manufacturing cost and complexity, and most likely an increase in the reliability of the motor. Related to this, the thermal-management technique could be used to minimize the localization of heating that would occur with a less distributed set of windings. The thermal-management technique also allows for other relaxations in the motor design, again with the benefits listed above. One possible design change may be to eliminate the cooling channel in the back-iron. An interesting topic for research would be the study of the simplifications that could be made in the motor design when employing the techniques discussed in this thesis.

Extending the techniques of this thesis to other regions of operation is a natural progression in the research. To some extent, this has already been carried out by Jackson [7] at MIT. He has had success in extending the ripple-reduction technique to the saturation region. He uses the same motor and apparatus as is used in this thesis. His static results are somewhat better because of improvements in the experimentation apparatus; however, the limitations are not completely overcome. Jackson's thesis provides more data on the techniques discussed in this thesis and it examines other aspects of the techniques, the motor, and the experimentation system. Jackson also examines what would be necessary to extend the techniques to a dynamic situation, but does not implement such a system. Ultimately, the techniques used in this thesis must to be implemented in a dynamic situation.

Appendix A

Matlab Code

This code performs the de-jittered DFT developed in Section 4.3.2. The code runs in Matlab.

```
function [y]=djdft(index,data)

% [y]=djdft(index,data) This function does a de-jittered DFT
% The output is y in the canonical DFT representation

% To put the de-jittered DFT in a DFT format, the coefficients at
% half the sampling frequency are added.  This properly accounts for
% the loss of phase information that results from aliasing.
% For real signals the sum must be a real number, so taking the
% real part does not affect operation, but cleans up finite-precision
% errors in the math, which could cause a probelm when doing an
% inverse FFT.

nopts=length(index)-1;
summa=(exp(index*[(0:(nopts/2)) ((-nopts/2):-1)]*i)\data)*nopts;
y=[summa(1:(nopts/2));...
real(summa(nopts/2+1)+summa(nopts/2+2));...
summa((nopts/2+3):(nopts+1))];
```

Appendix B

Calculating the Maximum Power Flow

Section 5.3 requires the calculation of the maximum power flow into a point when the power flow, P, is in the form given by Equation B.1. The parameters A, B and C are arbitrary, but range between 0 and 1. The maximization must be done over the angle, α.

$$P = A\cos(\alpha) + B\cos(\alpha - \frac{2\pi}{3}) + C\cos(\alpha - \frac{4\pi}{3}) \tag{B.1}$$

The maximization can be done using standard methods of calculus. First, the extrema are found by taking a derivative with respect to the free variable, setting the derivative equal to zero, and solving the resulting equation. This is carried out below. Once the extrema are found, each one must be scrutinized to find if it is a maximum or a minimum.

$$\frac{\mathrm{d}P}{\mathrm{d}\alpha} = -A\sin(\alpha) - B\sin(\alpha - \frac{2\pi}{3}) - C\sin(\alpha - \frac{4\pi}{3}) = 0 \tag{B.2}$$

Using the trigonometric identity, $\sin(\alpha - \beta) = \sin(\alpha)\cos(\beta) - \cos(\alpha)\sin(\beta)$, Equa-

tion B.2 can be written as in Equation B.3.

$$\frac{\mathrm{d}P}{\mathrm{d}\alpha} = -\left(A - \frac{1}{2}B - \frac{1}{2}C\right)\sin\alpha + (B - C)\frac{\sqrt{3}}{2}\cos\alpha = 0 \tag{B.3}$$

This equation can be solved for the quantity $\tan(\alpha)$.

$$\tan\alpha = \frac{(B - C)\frac{\sqrt{3}}{2}}{A - \frac{1}{2}B - \frac{1}{2}C} \tag{B.4}$$

Once the tangent is known, the sine and cosine can be determined.

$$\sin\alpha = \pm\frac{(B - C)\frac{\sqrt{3}}{2}}{\sqrt{A^2 + B^2 + C^2 - AB - BC - AC}} \tag{B.5}$$

$$\cos\alpha = \pm\frac{A - \frac{1}{2}B - \frac{1}{2}C}{\sqrt{A^2 + B^2 + C^2 - AB - BC - AC}} \tag{B.6}$$

At this point the original expression for P should be expanded in terms of $\sin(\alpha)$ and $\cos(\alpha)$ and substitutions should be done using Equations B.5 and B.6. This results in the final expression for the extremum of P, P_{max}.

$$\begin{aligned} P_{max} &= A\cos\alpha + B\cos\alpha\cos(\frac{2\pi}{3}) + B\sin\alpha\sin(\frac{2\pi}{3}) \\ &\quad + C\cos\alpha\cos(\frac{4\pi}{3}) + C\sin\alpha\sin(\frac{4\pi}{3}) \\ &= \left(A - \frac{1}{2}B - \frac{1}{2}C\right)\cos\alpha + \frac{\sqrt{3}}{2}(B - C)\sin\alpha \\ &= \pm\sqrt{A^2 + B^2 + C^2 - AB - BC - AC} \end{aligned} \tag{B.7}$$

Once the extrema have been found it must be determined whether each one represents a point where a maximum or a minimum resides. It is clear that the positive

root of P_{max} represents the maximum. Thus, a function in the form of Equation B.1 has a maximum represented by Equation B.8.

$$P_{max} = \sqrt{A^2 + B^2 + C^2 - AB - BC - AC} \tag{B.8}$$

Appendix C

The Data Collection System

All the elements necessary to control the motor and take measurements on it compose the data collection system. The system can be broken down into three subsystems: control, drive, and measurement. The control subsystem coordinates all system activities. It carries out the tasks of receiving input from the operator, commanding the drive subsystem, commanding the measurement subsystem and logging data received from the measurement subsystem. The drive electronics subsystem translates the commands of the control subsystem into electrical currents which power the motor. The measurement subsystem accepts commands from the control subsystem and reacts to these commands by taking the requested measurement and transmitting the data back to the control subsystem. It has the ability to take data regarding motor temperature, phase currents, rotor position, and torque. A system block diagram, Figure C-1, illustrates the interconnections of the system. These blocks are all described in more detail in the following sections and in Appendixes D through J. After briefly outlining the operation of the system, each subsystem is discussed in more detail. Finally, the data collection system description ends in Appendix K with a mention of some practical issues of system implementation.

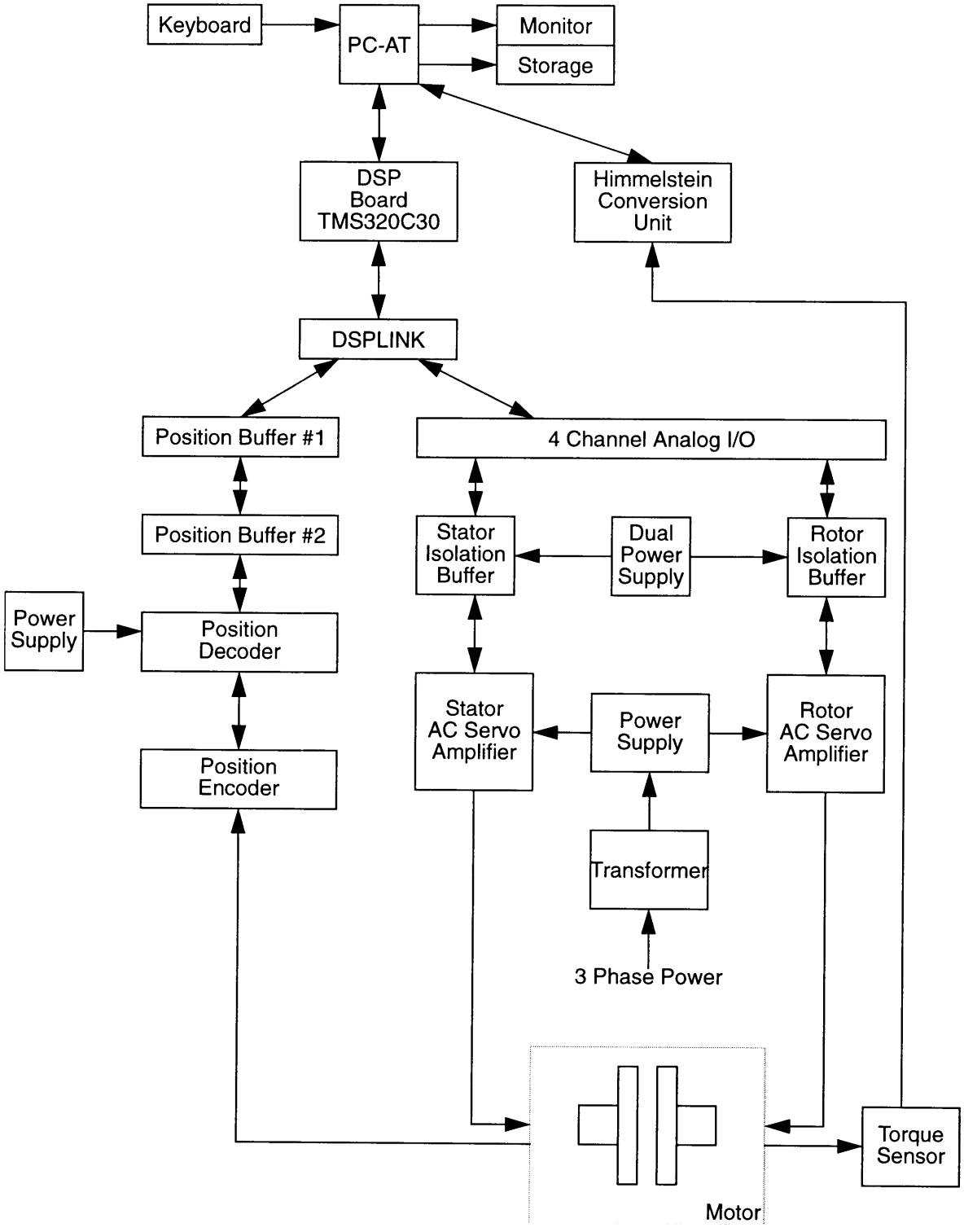

Figure C-1: System Block Diagram

C.1 The Control Subsystem

At the top level of the control subsystem is an IBM PC-AT compatible computer. This computer coordinates all aspects of system operation. It is responsible for requesting information from the operator and passing experiment parameters to the Spectrum TMS320C30 System Board, an add-in DSP card inside the personal computer. The personal computer (PC) also uploads information from the DSP board and displays it on the monitor or logs it on the hard disk as appropriate.

Many pieces of information are passed between the PC and the DSP board. Typically, at the beginning of an experiment, the PC loads a machine language program onto the DSP board. This code is primarily responsible for reading the rotor position and coordinating phase currents with this position. Before the code begins operation however, the PC queries the operator for any parameters for the experiment and passes these as arguments to the DSP code. After these steps are complete, the DSP board runs the code, thereby conducting the experiment. At this point, the DSP runs freely, without any control intervention from the PC-AT, yet, the PC is capable of reading measurements from the DSP board in a non-obtrusive way.

Once the DSP is executing experiment code, it performs many functions. It controls its two on-board Digital-to-Analog (D/A) converters as well as two other D/A converters on a 4 Channel Analog I/O daughter-card. These four analog outputs are fed, via galvanic isolation, to the drive subsystem. The drive subsystem uses these analog voltages as command inputs to generate the phase currents. This is described in more detail in Section C.2 and Appendix H. By putting phase currents under direct control of the DSP, various motor control algorithms can be implemented and evaluated.

The DSP also reads in the drive systems internal measurements of the phase currents. This is done using the analog-to-digital converter (ADC) on the 4 Channel Analog I/O daughter-card. This read-in process is under the control of the DSP and the results are stored in registers on the DSP for a non-obtrusive read by the PC control computer.

One piece of information does not travel through the DSP board to get to the personal computer. The PC directly communicates with the dynamometer, a Himmelstein Model 66032 Power Instrument, via an RS-232-C interface established through a "COM" port on the PC. The interface is interrupt driven, so there are no problem with data loss as a result of the asynchronous communications protocol. Using the Model 66032 command language, the PC requests that measurements be taken and then reads the values reported by the Model 66032. Once values are read in by the PC, they can be stored on a hard disk and displayed on the PC's monitor.

C.2 The Drive Subsystem

The core of the drive subsystem is formed around a modified Allen-Bradley Bulletin 1389 AC Servo Amplifier System. The servo system has two three-phase current drivers. Normally, the servo would commutate the three-phase currents based on inputs to it but this commutation feature has been bypassed, turning the servo system into a set of current drivers. Each three-phase driver has two analog voltage inputs. The driver generates two currents proportional to the input voltages, and the third current is calculated to balance the other two. This is done for both sets of three-phase currents.

The four command voltages to the servo system originate at the output of the DSP System Board and the 4 Channel Analog I/O daughter-card, two from each board, as described in the previous section. From there each signal passes through a galvanic isolation amplifier, which is configured to provide offset and scale trimming. This calibration is carried out by hand, and only needs to be done periodically to correct for drift. It should be noted that all drive system offsets and scale errors can be corrected at this single calibration point and that it is irrelevant where a scale or offset error originates.

The Allen-Bradley servo system has also been modified to provide feedback proportional to the output currents. The servo system internally generates feedback voltages roughly proportional to the current measured at the outputs. As part of the

modifications to the Allen-Bradley servo, the four feedback voltages are brought out of the servo system. These voltages are galvanically isolated and provided with an offset and scale trim. Again, the feedback path can be manually calibrated to eliminate offset and scale errors. These voltages are then input to the 4 Channel Analog I/O daughter-card's ADC, where they are measured and registered by the DSP. The PC-AT is then able to read these representations of the voltages. Also, the DSP uses these feedback voltages to provide a feedback control loop on the output currents — enhancing the accuracy of the not so well regulated outputs of the servo system.

Other elements of the drive system are a Allen-Bradley Bulletin 1389–T100DA 10kVA Isolation Transformer and an Allen-Bradley Bulletin 1389–PAT10 Power Supply Module. The isolation transformer provides a slight step up from three-phase 208 VAC input to three-phase 220 VAC input, as well as effecting galvanic isolation. The power supply module rectifies and regulates the three-phase input to provide power to the servo system.

C.3 The Measurement Subsystem

The measurement subsystem is composed of a position encoder/decoder, a Himmelstein dynamometer, temperature sensors and feedback voltages proportional to phase currents. Although properly belonging to the measurement subsystem, the feedback voltages have already been discussed in Section C.2, so further treatment will not be given to them.

The rotor position is encoded using a Canon R-2A Laser Rotary Encoder. Originally it was supplied with a Canon CI16–1 Encoder Interpolator. With this interpolator the resolution of the decoder is 1,048,576 counts per revolution, which gives 20 bits of resolution per revolution. However, the interpolator board was subject to ambient electrical noise and would not function in the operating environment of the motor. As a result, a more reliable decoder was designed which provided 16 bits of resolution per revolution. This decoder is not as subject to environmental noise. Furthermore, the original decoder had only 8 bits of absolute position information

and the more reliable decoder was designed to support 16 bits of absolute position.

The position encoder interfaces to the DSP System Board by way of a daughter-card, called the DSPLINK, which buffers the data. The DSP can read the position into its memory as often as it needs. This is vital to proper commutation of the motor requires an indication of position, regardless of the sophistication of the commutation algorithm. The PC-AT can read the position from the memory of the DSP board, using it as an index to log each data point. The monitor generally displays the position as well so that the operator knows the position of the rotor.

The dynamometer comprises another important element of the measurement subsystem. The Himmelstein Model 66032 Power Instrument communicates with the control subsystem, specifically the personal computer, by way of an RS-232-C serial connection. The Himmelstein receives commands from the personal computer as to what data to measure: torque, velocity or power. Once the measurement is complete, the Model 66032 sends the information back to the PC via the RS-232-C link, where each receipt of a character triggers a service interrupt in the PC-AT computer thereby insuring against data loss. Upon receipt of the information the personal computer typically logs the data on the hard disk and displays it on the monitor.

Four thermocouples distributed throughout the motor, two on the rotor and two on the stator, provide the basis for measuring the motor temperature. The thermocouples are standard T-type, Copper-Constantan, thermocouples. They are switched by a K-type thermocouple switch box (at room temperature). The switched output is fed into a K-type thermocouple amplifier. There is no method for computer assisted temperature logging, so all temperatures are logged by hand. The temperatures are recorded without any conversion between thermocouple types and a conversion accounting for the thermocouple type is done later.

This completes the discussion of the measurement system and with that the discussion of the three subsystems: control, drive and measurement. These three subsystems are well coordinated and are mostly under software control, providing a basis for flexible experimentation in this thesis and for subsequent work. It can serve as a model and starting place for future commercial and experimental systems.

The next six appendices provide detailed information on the operation of the hardware. For ease of organization, each appendix will address all issues related to one major system block: the PC-AT, the dynamometer, the DSP subsystem, the position encoder/decoder, the drive subsystem, the temperature sensing subsystem. Appendix J contains complete listings of sample code. The details of what goes on in that code is described in Appendices D–I. Finally, the last appendix provides insights into some of the major obstacles encountered in system integration.

Appendix D

The AT-compatible PC

This appendix provides information which may be helpful in understanding Section 4.3.2.

The AT-compatible PC serves as the top level control for the data collection and control system. The computer is an AT-compatible with a 30 Megabyte hard disk drive. It is configured with a Spectrum DSP board, floppy and hard disk drives, an RS232-C port and a monitor with an appropriate video interface. There is no mouse or printer connected to the computer. The computer uses MS-DOS as its disk operating system.

The CPU on the AT is a 80286 which is sufficiently fast to provide reasonable system performance. Some increase in speed would be desirable especially for compiling large programs.

D.1 Disk Management

The disk system is used to capture experimental data and transfer it to a permanent backup. The permanent backup can then used to create a temporary copy which can be ported to a workstation where it is processed. This porting to a workstation is done, only because of the limited configuration of the PC and because of the wide variety of software available on the workstation.

The PC-AT is configured with a 30 Megabyte hard drive. This disk drive size is

barely sufficient for the system. It must store operating system files and raw data from experiments. It also stores the Texas Instruments compiler for the TMS320C30 processor on the DSP board. Ideally the Microsoft C compiler used for code writing for the PC should also be on the PC. The combined size of both compilers however does not permit this, so a second PC is used to develop software for the PC. With all the system files and compilers, there is about 5 Megabytes of free storage space left for experimental data capture.

The hard disk drive puts some limitations on the data collection system's performance. Because the computer searches around for empty storage space when it can find it, as the disk drive fills up, the system slows down. This can present a problem during data taking as the motor position might be changing faster than the PC can store the data. This problem can be alleviated in the future by increasing the hard drive storage space.

The computer also has two floppy drives, one capable of reading and writing 1.2 MB disks (as well as 360 KB) and the other capable of reading and writing 360 KB disks. The 360 KB drive is not used. The 1.2 MB disk drive is used to copy the temporary experimental data from the hard drive onto a permanent backup floppy. After every few experiments, the data should be transferred to the 1.2 MB disks. These disks should then carried over to another PC where they can be temporarily copied to a 3.5 inch disk (1.44 MB). Most workstations then have a facility for reading 3.5 inch IBM format disks.

Appendix E

The Dynamometer Subsystem

The dynamometer subsystem is responsible for measuring and logging torque output upon request from an experiment. It is also possible for the dynamometer to report angular velocity and power, but this data is generally is not requested during the current experimental work. This appendix is referenced by Sections 4.2 and 4.3.1.

The following paragraph briefly summarizes the dynamometer operation and subsequent sections will elaborate on the details. The PC computer is connected via an RS-232-C standard connection to the Himmelstein Power Instrument as described in Section E.1. Since a protocol transfer is involved, it is necessary to set up the protocol on both the PC and on the dynamometer as reported in Sections E.3.4 and E.4, respectively. The PC has further set-up to be done since it uses interrupt driven communication to ensure against data loss. The set-up of the interrupt handler is documented in Section E.3.3. Once the set-up is completed, the PC can transmit and receive messages with the dynamometer as described in Sections E.3.5 and E.3.6. The actual format of the messages between the PC and the dynamometer is given in Section E.5.

The detailed description of system operation will take place in much the same order as outlined above. However, to be able to understand the code that implements what is described above, first it is necessary to describe all the definitions and declarations made in the headers which assist in the code writing. This is done in Section E.2.

E.1 The RS-232-C Link

An RS-232-C serial link provides a mechanism for communication between the PC and the Himmelstein torque sensor. The connector for this link is hand-fashioned with a DB-25 type connector for connection to the back of the PC-AT and a card edge connector for connection to the Himmelstein Power Instrument. The wiring is given in Figures E-1 and E-2.

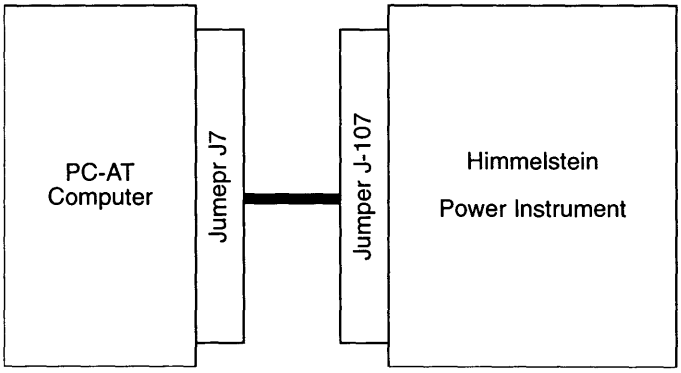

Figure E-1: Dynamometer System Interconnect.

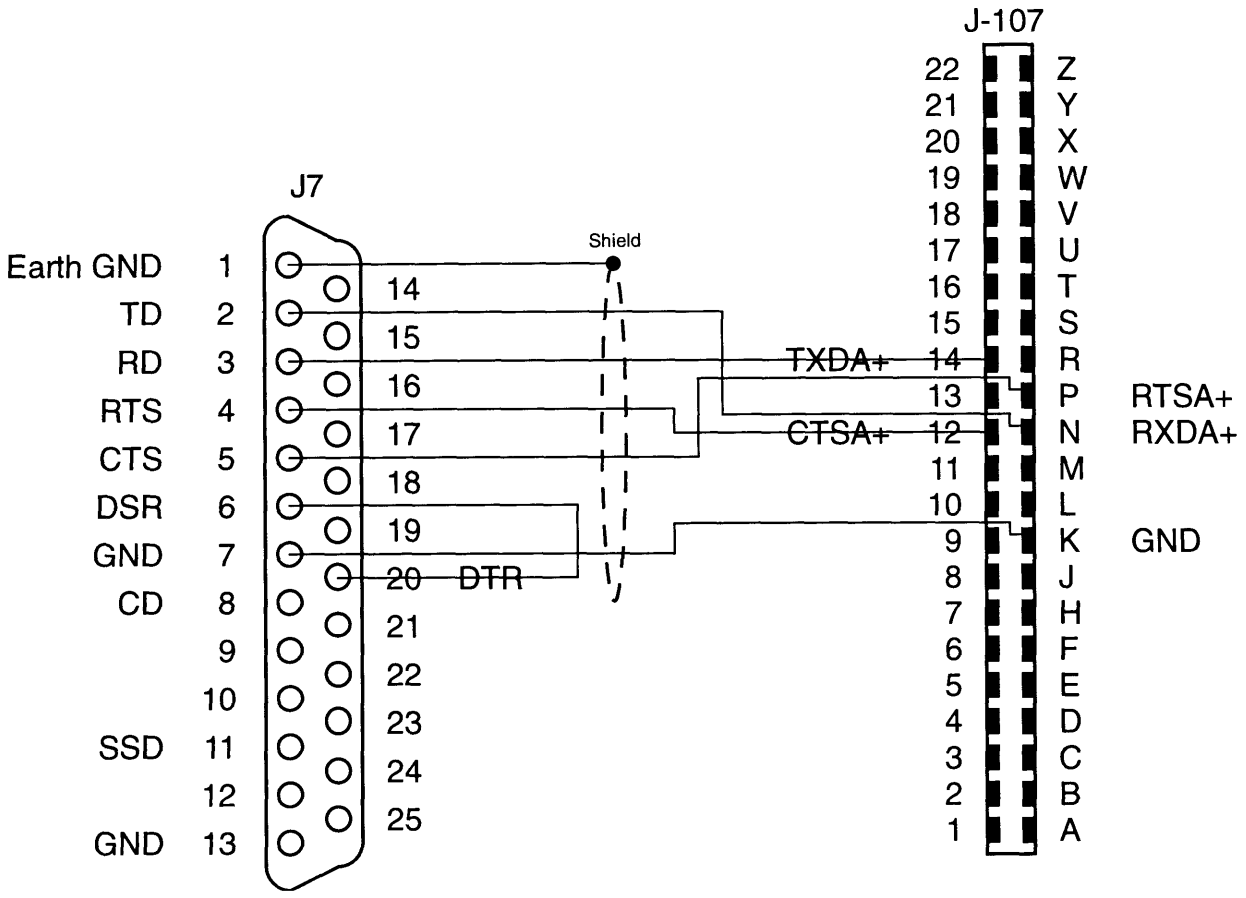

Figure E-2: Jumper J4 Pin Assignments and Wiring Diagram.

RS-232-C is a protocol transfer. The proper protocol in this system is 9600 baud, 8 bit, 1 Stop Bit, No Parity. This information must be programmed into both the PC's RS-232 port and the Himmelstein's port.

AT compatible PC's have several serial ports. Typically serial port COM1 is used for a mouse interface. The next serial port, COM2, is generally used by modem or other such device. It has been decided to use this serial port to communicate with the dynamometer. The COM2 serial port utilizes I/O port address `0x2f8`.

E.2 PC Code Definitions and Declarations

Coding for the RS-232 communication on the PC has been done in Microsoft C. The code has been adapted from the book *Advanced C Programming* [14], which originally coded in Turbo C. Before compilation the code defines and declares many variables for the aid of the compiler. The main code file first inputs a header file, *serial.h*, to define some I/O register locations and bit masks. The function of *serial.h* is described in more detail below in Section E.2.1. After *serial.h* is loaded, the main code file defines and declares many global references completing the definitions and declarations. This is described in Section E.2.2. More information on RS-232-C communications can be found in *Advanced C Programming* [14] and *RS-232 Simplified, Everything You Need to Know About Connecting, Interfacing, & Troubleshooting Peripheral Devices* [15].

E.2.1 Definitions and Declarations in *serial.h*

The header file *serial.h* is loaded from the main code file. This accomplished with the following line of code.

```
#include "serial.h"
```

The file *serial.h* first defines a structure type, `sio`, corresponding to the register structure of the I/O port. The definition is shown below.

```
/*
 * define the register structure for the serial i/o
 */
struct sio {
    char    data;           /* data register */
    char    interrupt_enable;/* interrupt enable register */
    char    interrupt_id;   /* what kind of interrupt is going on */
    char    format;         /* communications format */
    char    out_control;    /* modem control lines */
    char    status;         /* status byte */
    char    i_status;       /* input status */
    char    scratch;        /* extra pad */
};
```

The header file also defines two instantiations of this structure named `COM1` and `COM2` and puts them at addresses `0x3f8` and `0x2f8` respectively. This definition is done in the code fragment below. By assigning the addresses as such, a write to the structure using the `outp` (port write) command becomes a cosmetically attractive way of writing to the serial port registers.

```
/*
 * The location of the i/o registers on the IBM PC
 */
#define COM1    ((struct sio near *)0x3f8)
#define COM2    ((struct sio near *)0x2f8)

/*
 * Use COM2 for this program
 */
#define COM      COM2
```

Each successive byte of the structure `sio` corresponds to the next port address. The first byte of the structure `sio` is the **Data Register** to and from which data is written and read. The second byte is the **Interrupt Enable Register** which defines when interrupts to the PC will be generated, if they are at all. The possibilities for interrupts are when: the modem status flag changes, the receive status flag changes, the transmit buffer is empty, a character is received.

The **Data Register** and **Interrupt Enable Register** are also used under certain circumstances for transmitting the low and high bytes of the baud rate. This condition is initiated by setting the **Divisor Latch Access Bit** in the **Line Control Register** as discussed later. The following lines define the bit masks for easy programming of interrupt selects via the **Interrupt Enable Register**.

```
/*
 * Defines for Interrupt Enable Register (interrupt_enable)
 */
#define I_STATUS (1 << 3) /* interrupt on modem status changed */
#define I_REC_STATUS (1 << 2) /* interrupt on rec. status changed */
#define I_TRANS_EMPTY (1 << 1) /* interrupt on trans. empty */
#define I_CHAR_IN (1 << 0) /* interrupt on character input */
```

The third byte in the structure `sio` relates what type of interrupt occurs. The register is named the **Interrupt Identification Register**. It is necessary to read

this register to indicate that any interrupt present has been processed. The reading action clears the interrupt flags.

The fourth byte of the structure sets the communication format. This register is commonly known as the **Line Control Register** (**LCR**). Setting bit 7 of the **LCR**, the **Divisor Latch Access Bit** (**DLAB**), to a '1' enables the baud rate to be set on the next writes to the **Data Register** and **Interrupt Enable Register**. Setting **DLAB** to a '0' resets the functions of the **Data** and **Interrupt Enable Registers** to their normal modes. This type of register usage is often referred to as over-loading. The header file *serial.h* defines a bit mask for these functions.

```
/*
 * Defines for Line control register  (format)
 */
#define F_BAUD_LATCH    (1 << 7)     /* enable baud rate registers */
#define F_NORMAL        (0 << 7)     /* normal registers enabled */
```

Bit 6 of the **LCR** sets the "break" or "no break" condition. The next three bits, bits 3 through 5, set the parity either as "none", "odd", "even", "mark", or "space". The next bit, bit 2, specifies the number of stop bits as being either one or two. Finally the last two bits, bits 1 and 0, set the number of data bits for the transmission. Bit mask are defined for all these bits as shown below.

```
#define F_BREAK         (1 << 6)        /* set a break condition */
#define F_NO_BREAK      (0 << 6)        /* no break condition */

#define F_PARITY_NONE   (0 << 3)        /* no parity on output */
#define F_PARITY_ODD    (1 << 3)        /* odd parity on output */
#define F_PARITY_EVEN   (3 << 3)        /* even parity on output*/
#define F_PARITY_MARK   (5 << 3)        /* parity bit is always 1 */
#define F_PARITY_SPACE  (7 << 3)        /* parity bit is always 0 */

#define F_STOP1         (0 << 2)        /* Use one stop bit */
#define F_STOP2         (1 << 2)        /* Use two stop bits */

#define F_DATA5         (0)             /* 5 data bits on output */
#define F_DATA6         (1)             /* 6 data bits on output */
#define F_DATA7         (2)             /* 7 data bits on output */
#define F_DATA8         (3)             /* 8 data bits on output */
```

The next register is the **Modem Control Register**. This register takes direct control of some of the RS-232 signal lines. It also permits loop-back test mode to be entered. Setting bit 4 forces the modem into loop-back test mode. Bit 1 is the **Request To Send(RTS)** bit, which tells the modem to indicate that data is ready to be transmitted. Bit 0 is the **Data Terminal Ready(DTR)** bit which tells the modem to indicate its presence to the remote device. The code below shows the bit masks for the **Modem Control Register**.

```
/*
 * Defines for the MODEM control register (out_control)
 */
#define O_LOOP          (1<<4)          /* loopback test */
#define O_OUT1          (1<<3)          /* Extra signal \#1 */
#define O_OUT2          (1<<2)          /* Extra signal \#2 */
#define O_RTS           (1<<1)          /* Request to send */
#define O_DTR           (1<<0)          /* Data terminal ready */
```

The next register is the **Line Status Register** whose 8-bit value stores various flags. Bit 6 stores the **Transmitter Shift Register Empty** flag and Bit 5 the **Transmitter Holding Register Empty** flag. Bit 4 flags a **Break Interrupt** and bit 3 flags a **Framing Error**. Bit 2 signals a **Parity Error** and bit 1 that the input has overrun the buffer, an **Overrun Error**. Bit 0 indicates **Data Ready**, that the receiver has a character in its buffer. The code below defines the appropriate bit masks.

```
/*
 * Line Status register (Status)
 */
#define S_TXE           (1 << 6)
#define S_TBE           (1 << 5)    /* Transmitter buffer empty */
#define S_BREAK         (1 << 4)    /* Break detected on input */
#define S_FR_ERROR      (1 << 3)    /* Framing error on input */
#define S_PARITY_ERROR  (1 << 2)    /* Input parity error */
#define S_OVERRUN       (1 << 1)    /* Input overrun */
#define S_RxRDY         (1 << 0)    /* Receiver has character ready */
```

Finally, there is the **Modem Status Register**. This register reports the status of the **Data Carrier Detect(DCD)** flag, the **Ring Indicator(RI)** flag, **Data Set**

Ready(DSR) flag, and the **Clear To Send(CTS)** flag. It also reports the "delta" of these signals, that is, whether or not they have changed. This register is not currently being used for any purpose.

```
/*
 * Modem Status Register (i_status)
 */
#define I_DCD           (1 << 7)        /* DCD control line is on */
#define I_RI            (1 << 6)        /* RI control line is on */
#define I_DSR           (1 << 5)        /* DSR control line is on */
#define I_CTS           (1 << 4)        /* CTS control line is on */
#define I_DEL_DCD       (1 << 3)        /* DCD line changed */
#define I_DEL_RI        (1 << 2)        /* RI line changed */
#define I_DEL_DSR       (1 << 1)        /* DSR line changed */
#define I_DEL_CTS       (1 << 0)        /* CTS line changed */
```

The last part of the header defines a few constants to assist in setting the baud rate to be used on the PC. The value of the variable `SPEED` will hold a constant indicating the baud rate.

```
/*
 * constants are used to define the
 * baud rate for the serial i/o chip
 * (Selected entries from Table-III of the National 8250
 *  data sheet)
 */
#define B1200   96
#define B2400   48
#define B9600   12

⋮

#define SPEED  B9600
```

E.2.2 Definitions and Declarations in the Main File

After the definitions contained in *serial.h* are loaded into the compiler, the main file makes several definitions and declarations. This section will handle the definitions and declarations in the order in which they occur in the main file.

The first define sets the buffer size for the RS-232 reception to be 1024 characters.

```
/* size of the buffer to use for string incoming characters */
#define COM_BUF_SIZE (1 * 1024)
```

The next lines of code relating to the RS-232 interface are the following.

```
static char *buffer_start;      /* beginning of the buffer */
static char *buffer_end;        /* end of the buffer */

/* pointer to next place to put character in */
static char *buffer_in;
static char *buffer_out;        /* place to get next character from */
```

The first two statement lines of code declare pointers to the start and end of the buffer. The next statement declare a pointer to the current location to write incoming data into the buffer. The line after that declares a pointer to the current location in the buffer from which to read data. Handling of the buffer is cyclic. As characters are read from the RS-232 they are put into the location `buffer_in`. When this pointer reaches `buffer_end`, the pointer is reset to `buffer_start`. Similarly, once data has been read from the RS-232 and put in the buffer, it is read out of the buffer from location `buffer_out`. When `buffer_out` reaches the end of the buffer, `buffer_end`, it is reset to the beginning of the buffer, `buffer_start`.

The next two lines of RS-232 related code declare a counter, `count`, containing the number of characters to be read remaining in the buffer, and a flag, `EOT`, indicating the end of a transmission from the dynamometer. These lines of code are shown below.

```
static int count = 0;           /* number of characters in buffer */
volatile int EOT = 0;
```

The following lines of code declare the presence of a number of pointers. The first two lines declare pointers that will be used to store the locations for the serial interrupt handler and break handler present when the program begins. The locations will be stored so that upon completion of the code, the old original handlers can be restored and the code will terminate without any side effects. The break handler is the interrupt service routine activated by the pressing of the **<CTRL–BREAK>**

key sequence. It is desirable to circumvent this handler because its normal operation would allow jumping out of the program while the serial interrupt handler is still in place.

```
static void (_interrupt _far *old_serial_interrupt)();
static void (_interrupt _far *old_break_interrupt)();
```

Following the pointer declarations are four declarations which declare routines to initialize the buffer, to initialize the serial port, to empty the buffer, and to write a string to the RS-232 output.

```
void init_buf(void);
void init(void);
void empty_buffer(void);
void write_port(char *s);
```

E.3 The Main Code Body for the PC

Before the PC-AT can transmit or receive anything, the serial port must be initialized to the proper protocol. Furthermore, reception of characters by the PC is interrupt driven, thereby assuring complete reception of the Himmelstein's transmission. This approach has been used in place of the simpler polling method which does not guarantee reception of the complete transmission. The interrupt approach to communications also requires some initialization.

The main code segment is responsible for initializing a character storage buffer, initializing the serial port, and initializing the interrupt handlers. It is also where the end goal of transmitting data and receiving data is accomplished.

The main code body operates as follows. First, the main code makes a few more declarations. Second, the code initializes and empties the buffer. Third, it initializes the interrupt handlers. Fourth, the serial port is initialized to use the proper RS-232 protocol. Then it proceeds to write a command to the Himmelstein and read the response back. The write/read cycle can be done as many times as desired. When it is desired for the program to exit, the code removes the interrupt handlers.

E.3.1 Main Code Body Declarations and Definitions

Even after all the previous declarations and definitions discussed in previous sections, a few last variables are declared in the main code body. The variable `status` is declared in the line shown below. This variable is used for storing any reports of the status of the RS-232 port. Also, the flag variable `EOT` is set to zero, indicating that the transmission from the Himmelstein has not been completed. This will be the state of the variable before any transmission to the Himmelstein is ever made. The above declaration and define are accomplished in the following lines.

```
int status;

EOT=0;
```

E.3.2 Initializing the RS-232 Buffer

Initialization of the RS-232 is accomplished by calling the `init_buf` routine with the following program line.

```
init_buf();
```

The buffer initialization code is given below. First, a section of memory of length `COM_BUF_SIZE` is allocated as the buffer and the pointer `buffer_start` is pointed to the beginning of the buffer. Then the pointers `buffer_in`, the current location to which to write incoming data, and `buffer_out`, the current location to read from the buffer, are set to the beginning of the buffer. Finally the end of the buffer, `buffer_end`, is assigned a value pointing ten characters before the end of the allocated memory area.

```
/*******************************************************
 * init_buf -- initialize the buffer pointers          *
 *******************************************************/
void init_buf(void)
{
        buffer_start = malloc(COM_BUF_SIZE);
        buffer_in = buffer_start;
        buffer_out = buffer_start;
        buffer_end = buffer_start + COM_BUF_SIZE - 10;
}
```

In the book *Advanced C Programming* [14] the code flushes the buffer after initializing it. This seems to be an unnecessary step, however the incredulous may want to do it anyway. It should be noted that the code used for experiments in this thesis accidentally flushed the buffer before creating/initializing it. This has only just been realized during the documentation of the system. This bug did not adversely affect system performance as flushing the buffer does not cause anything to be written to memory.

E.3.3 RS-232 Interrupt Initialization

Three things must be done to initialize the interrupt handler. First, the old interrupt code location must be stored. This makes it possible to restore the old interrupt when the main code completes. Also, in order ensure that the old interrupt vector is restored, it is necessary to disable the <**CTRL–BREAK**> interrupt, which happens whenever <**CTRL–BREAK**> is pressed. Failure to do this could cause the main code to terminate without restoring the old interrupt vector. The old interrupt handler locations are stored using the following lines of code.

```
old_serial_interrupt=_dos_getvect(0x0b);
old_break_interrupt=_dos_getvect(0x1b);
```

Second, after the old interrupt handler locations are stored, new interrupt vectors are stored with the following lines of code. While the new handler locations are stored in the vector table, it is necessary to disable all interrupts. Failure to do this could cause the system to look up a partially updated interrupt vector, if an interrupt were to occur during the update, and jump to that erroneous location. Thus, it is necessary to disable interrupts while updating the vector table. After completing the update, it is permissible to enable all interrupts.

```
_disable();
_dos_setvect(0x0b, new_serial_interrupt);
_dos_setvect(0x1b, new_break_interrupt);
_enable();
```

Finally, the serial interrupt must be enabled. This is distinct from the enabling and disabling of interrupts as done above, but rather refers to the specific interrupt dedicated to serial I/O. This enable is accomplished by writing to two different port locations as shown below.

```
/* enable interrupts */
outp(0x21, inp(0x21) & 0xF7);
outp(0x20, 0x20);
```

These steps complete the interrupt initialization.

E.3.4 RS-232 Port Initialization

The RS-232 port needs to be initialized before using it. This initialization is achieved by calling the subroutine `init` using the program line below.

```
init();
```

The first initialization step is to disable interrupts to prevent an interrupt from occurring before the settings are set. Next, the serial port interrupt generation is enabled. Then, the data format is sent to the **Line Control Register**. The format is set to 8 bit, no parity, 1 stop bit, no breaks. Also, the **Divisor Latch Access Bit** is set, which indicates that the next byte sent to the **Data Register** will be the lower eight bits necessary to set the correct baud rate, and the next byte sent to the **Interrupt Enable Register** will be the higher eight bits necessary to set the baud. After the baud rate is set, the **Line Control Register** is again sent the correct format, but this time the **Divisor Latch Access Bit** is not set. At this point, the serial port is instructed to indicate its readiness for data transmission and reception; **Request To Send(RTS)** and **Data Terminal Ready(DTR)** are asserted. Finally, the serial port input registers, the **Data Register**, the **Interrupt Enable Register**, the **Interrupt ID Register**, and the **Line Status Register** are read, thereby clearing their flags indicating they have data. Finally, interrupts are cleared and enabled, and the subroutine returns.

```
/*******************************************************
 * init -- initialize the port                         *
 *******************************************************/
void init(void)
{
        /* don't allow interrupts while we do this */
        _disable();
        /* receive interrupts */
        outp((int)&COM->interrupt_enable,   I_CHAR_IN);

        outp((int)&COM->format,
            F_BAUD_LATCH|F_NO_BREAK|F_PARITY_NONE|F_STOP1|F_DATA8);

        /* now that we have the baud latch set, send baud */
        outp((int)&COM->baud_l, SPEED & 0xFF);
        outp((int)&COM->baud_h, SPEED >> 8);

        outp((int)&COM->format,
            F_NORMAL|F_NO_BREAK|F_PARITY_NONE|F_STOP1|F_DATA8);

        outp((int)&COM->out_control, O_OUT1|O_OUT2|O_RTS|O_DTR);

        /* read the input registers to
           clear their i-have-data flags */
        (void)inp((int)&COM->data);
        (void)inp((int)&COM->interrupt_enable);
        (void)inp((int)&COM->interrupt_id);
        (void)inp((int)&COM->status);

        outp(0x20, 0x20);                   /* clear interrupts */
        _enable();
}
```

E.3.5 Writing to the Serial Port

At this point, the discussion will move to the actual code that is run to perform an experiment. The scope of this discussion is limited to the RS-232 interface operations, and does not presently view the entire software operation.

Once the initialization steps have been completed, it is possible to transmit data to the dynamometer. This is accomplished by way the of `write_port` subroutine using a call similar to the one below. The call below writes the string `"RUNN "` to

the serial port and then waits until the End-Of-Transmission Flag, `EOT`, becomes not zero indicating that a response to the command has been received. At this point, the buffer has data from the dynamometer which can be appropriately processed and to prepare for the next transmission the `EOT` flag can be set back to zero.

```
write_port("RUNN ");
while (EOT==0);
EOT=0;
```

To understand the real workings of the code fragment above, it is necessary to examine the subroutine `write_port`. The subroutine enters with an argument which points to the beginning of the string to be transmitted. The subroutine immediately enters a loop which checks to see if the current location of the pointer in the string points to the string termination character. If it does, then the loop completes and the code continues. If it does not, then the subroutine enters a polling loop which examines the **Line Status Register** and waits until the **Transmitter Buffer Empty** status line is asserted, meaning that a character can be written to the transmitter buffer. After a successful polling, the subroutine writes the contents of the current pointer location in the string to the **Data Register**. It then increments the pointer location in the string and returns back to the beginning of the loop, once again checking to see if the string is at its termination.

Coming out of the loop, the code waits until the **Transmitter Buffer Empty** line is de-asserted and then writes a line-feed character, `0x0A`, to the serial port. This line-feed is the normal termination character of strings written to the dynamometer and is not part of the standard protocol of an RS-232 communication. The writing out of the line-feed completes the operation of the subroutine `write_port`.

```
/*******************************************************
 * Write_port -- write a string to the RS232 *
 *******************************************************/
void write_port(char *s)
{
while(*s!='\0') {
while (!((inp((int)&COM->status)) & S_TBE));
outp((int)&COM->data,*s);
```

```
s++;
}
while(!((inp((int)&COM->status)) & S_TBE));
outp((int)&COM->data,0x0A);
}
```

E.3.6 Reading from the Serial Port

Reading a character from the serial port is a bit more automatic than writing a character. When the PC expects to receive data, it resets the End-of-Transmission flag; it sets `EOT` equal to zero. At some point, the serial port will receive a character and in turn generates an interrupt to the PC. This interrupt is handled by the interrupt handler, whose location had previously been installed in the interrupt vector table during initialization. In this case, the interrupt handler is the routine `serial_interrupt` whose code is listed below. The net effects of the handler will be to read the character from the **Data Register** and to clear the interrupt signal.

```
/*******************************************************
 * serial_interrupt -- interrupt handler for serial  *
 * input *
 * *
 * Called in interrupt mode by the hardware when  *
 * a character is received on the serial input *
 *******************************************************/
static void _interrupt _far new_serial_interrupt()
{
        int     int_status;     /* status during interrupt */

_disable();

        int_status = inp((int)&COM->status);

        /* tell device we have read interrupt */
        (void)inp((int)&COM->interrupt_enable);
        (void)inp((int)&COM->interrupt_id);

        if ((int_status & S_RxRDY) == 0) {
                _enable();
                return;
        }
```

```
        *buffer_in = inp((int)&COM->data) & 0x7F;
if ((*buffer_in)==0x04) {
EOT=1;
((*buffer_in)='\0');
}
if ((*buffer_in)==0x0A) ((*buffer_in)='\0');
if ((*buffer_in)==0x0D) ((*buffer_in)='\0');
buffer_in++;

        if (buffer_in == buffer_end)
                buffer_in = buffer_start;

count++;
        outp(0x20, 0x20);
        _enable();
}
```

The subroutine above begins by disabling further interrupts. Then the routine reads the status of the serial port by reading the **Status Register**. Next, it clears the interrupt by reading the **Interrupt Enable Register** and the **Interrupt Identification Register**. After that, it looks at the data read from the **Status Register** and checks to see if a character is waiting to be read. If not, the subroutine enables interrupts and returns. There does not seem to be any reason why this would happen, but is retained from the original code (see [14]) as a safety precaution.

At this point, the code actually reads the character from the data register, zeros the most significant bit, and stores the character in the location pointed to by `buffer_in`. The code then checks to see if this character is ASCII character `0x04`. This is the character the dynamometer uses to indicate an end of transmission. If the character is `0x04`, then the program sets the `EOT` flag and also converts the character to a null character. This conversion is done so that the contents of the buffer appears as a normal string terminated by the null character. If the character is either a `0x0A`, line-feed, or `0x0D`, carriage return, then a null character is substituted for it. This again keeps the buffer in normal string format. After these conversions are done, the pointer to the current read-in location is incremented and cycled to the beginning of the buffer if necessary. The `count` variable containing the number of unread characters in the buffer is incremented. Finally, interrupts are enabled and the interrupt handler

returns.

Typically, the main program body should be looping waiting for the `EOT` flag to be asserted. This self-defined protocol keeps the software from becoming a coordination nightmare. The communication works such that a command is issued, and the code then waits for the reply. When the reply comes back, it is in string format beginning at the location pointed to by `buffer_out`. The program can then process the dynamometer's response appropriately.

E.3.7 Removing the Interrupt Handlers

Before the program terminates, it is necessary to restore the old interrupt handler locations into the interrupt vector table. The code currently does this as part of the new <**CTRL–BREAK**> handler. The necessary lines of code can be put anywhere else an exit is desired. These lines of code are listed below.

```
_disable();
_dos_setvect(0x1b, old_break_interrupt);
_dos_setvect(0x0b, old_serial_interrupt);
outp(0x21, inp(0x21) | 0x08 );
outp(0x20, 0x20);
_enable();
```

These lines of code first disable the interrupts because, again, the interrupt vector table is being modified. The old interrupt vector locations are then re-inserted into the interrupt vector table and the serial port interrupt is disabled.

In the case of the <**CTRL–BREAK**> key being pressed, the code above is called via the new <**CTRL–BREAK**> interrupt handler, `new_break_interrupt`, which is listed below. This code does the same thing as listed above, except it continues by calling the `old_break_interrupt` interrupt handler.

```
static void _interrupt _far new_break_interrupt()
{
_disable();
_dos_setvect(0x1b, old_break_interrupt);
_dos_setvect(0x0b, old_serial_interrupt);
outp(0x21, inp(0x21) | 0x08 );
```

```
outp(0x20, 0x20);
_enable();

(*old_break_interrupt)();
}
```

E.4 Configuring the Dynamometer

Not only does the PC need initialization but also the dynamometer does. There are two facets to this procedure. First the dynamometer must be calibrated. The calibration only needs to be done once ever (or when it is desirable to re-calibrate the instrument). For more information on this procedure, refer to the Himmelstein Power Instrument's operating instructions [16, Section 3.0]. The second part of the initialization is the programming of the RS-232 protocol into the instrument. This programming is stored in the dynamometer even during power down.

Initialization of the dynamometer RS-232 port can be done either remotely or via the keypad on the front of the instrument. It is much easier, and thus advisable, to initialize the dynamometer via the keypad. This procedure will be described below. Strangely, the dynamometer operator's manual [16, Section 5.11] lumps the details of the RS-232 set-up with the printer set-up. It has been mentioned previously that the present system uses a 9600 baud, 8 bit, no parity protocol. These numbers must also be programmed into the dynamometer settings.

To start, the dynamometer must enter the program mode. The manual refers to it as the "PGRM mode." This can be achieved by hitting the **PGRM** key, then punching in the three digit program code (normally 473), then pressing the **ENTR** key. This is described in Power Instrument's operating instructions [16, Section 5.5.5]. After the program mode is entered, the dynamometer gives the option of changing many settings. In general, the first eleven can be ignored by hitting **ENTR** eleven times.

Once this is done, the prompt "I/O Baud?" will be displayed. The proper baud rate should then be entered using the numeric keypad on the dynamometer. The

possible rates are 75,110,150,300,600,1200,2400,4800,9600, and 19200. In the case of 19200 baud, the number 1920 should be entered. After the digits have been entered the **ENTR** key should be pressed. The dynamometer will then query with "I/O Baud?" again. If the number is to be accepted press **ENTR**, otherwise enter a new baud rate.

After the baud rate has been entered, the system will query for the number of bits, either 7 or 8, with the prompt "# of BITS ?". The appropriate number, **8**, should be pressed on the numeric keypad followed by **ENTR**. The system will again prompt "# of BITS ?" asking the user to accept the current value by pressing **ENTR**.

Once this is done, the system will query for the parity by prompting with the current parity setting. To change the setting, hit the ± key until the correct parity is displayed. Possibilities for this setting include no parity, odd parity, or even parity. Once the desired parity setting is displayed, pressing **ENTR** will accept this value. That completes the necessary protocol set-up.

E.5 Messages Passing

Messages between the PC and dynamometer make it possible to instruct the dynamometer to take certain measurements and report back the results. The general flow of data in the serial link starts with the issue of an ASCII command by the PC to the dynamometer. There are fixed format commands recognized by the Himmelstein Power Instrument which generate fixed format responses. The responses are also in ASCII. Typically the command will be to take some measurement and the response will be a data or an error message.

This section catalogs the command and response formats that arise in typical operation. Also the possible error messages are given. All the possible messages are documented in more detail in the operating manual [16, pp. 79–88].

The first command of interest is the command `RUNN [lf]`. The "`[lf]`" indicates a line-feed character. Note also, that the `RUNN [lf]` command has a space after the word "`RUNN`." This command causes the dynamometer to start a "test". The

dynamometer documentation is sufficiently ambiguous as to the definition of a "test," to warrant using this at the beginning of each measurement session. It appears that it is unnecessary however. When the command executes without error, the message `OK[cr][lf]` is returned. "`[cr]`" is a carriage return. If there is an error, one of the error messages listed below will be returned.

The only other command of interest is the `SCAN ##,##;##,##[lf]` command. Again, the word "`SCAN`" is followed by a space, and the line is terminated with a line-feed. This command instructs the dynamometer to take a measurement. The arguments to the command, given by each "`##`", indicate which channels to measure. A comma indicates a range of channels and a semi-colon delimits ranges. The response to the scan command comes as one line per measurement channel in the format `##,[channel value][cr][lf]`. The "`##`" for each line is the channel number. Following this is the string containing the measurement value. If there is an error, one of the error messages below will be returned instead.

There are six possible error messages. These are listed below:

```
ER01.....INVALID COMMAND[cr][lf]
ER02.....CHANNEL# OUT OF RANGE[cr][lf]
ER03.....VALUE OUT OF RANGE[cr][lf]
ER04.....MEMORY FULL[cr][lf]
ER05.....NONE[cr][lf]
ER06.....PARITY ERROR[cr][lf]
```

The messages are self-explanatory. Further documentation is available in operating instructions [16].

Appendix F

The Spectrum DSP Subsystem

The Digital Signal Processing (DSP) subsystem is the heart and soul of the motor control system. It communicates with the PC, receiving arguments to various programs that are written for the DSP subsystem as well as transmitting measurement data from the rest of the system to the PC. To summarize its capabilities, the DSP controls and reads data from the torque sensing equipment, the position sensor and the Allen-Bradley servo system. An understanding of this appendix may provide greater context for the comments in Section 4.3.2.

The DSP subsystem is composed of SPECTRUM Signal Processing Inc.'s TMS320C30 System Board and two daughter cards, the DSPLINK Prototype Interface Module and the 4 Channel Analog I/O Board. The DSP motherboard contains the DSP chip, two digital-to-analog converters (DAC's) and two analog-to-digital converters (ADC's). It also connects to the DSPLINK, which is SPECTRUM's data bus. It is by way of this bus that the System Board interfaces with the rotary encoder via the DSP Link Prototype Interface Module. The DSPLINK bus also provides a means of interfacing with the 4 Channel I/O Board. This board supplements the ADC's and DAC's on the motherboard by adding two DAC's and a four-input muxed ADC. The drive system uses the two DAC's on the motherboard and two on the 4 Channel I/O Board, as well as the four inputs to the ADC on the 4 Channel I/O Board. It does not use the two ADC channels on the motherboard. The analog inputs and outputs are used to interface with the Allen-Bradley servo system.

To understand the DSP subsystem it will be helpful to describe each of the boards and their functions in more detail.

F.1 The DSP System Board

The TMS320C30 System Board is a board designed by Loughborough Sound Images Ltd. and manufactured in North America by SPECTRUM Signal Processing Inc. The central processing unit on the board is a Texas Instruments TMS320C30 microprocessor. The board comes socketed for up to four banks of random access memory (RAM) storage, of which only banks 0 and 3 are populated with a total of 128K bytes of storage. The board plugs into an 8-bit ISA socket on a PC, and by the default setting of `LK4` occupies I/O ports 0x290 to 0x29F. An alternate address range can be set using link `LK4`.

Since the board resides inside an AT compatible computer, it is necessary to discuss how to move information between the computer and the DSP subsystem. There are primarily two types of information transfers to be done: program transfers and data transfers. The former is necessary to download code onto the DSP board. The latter is employed both to pass arguments (generally from the PC to the DSP) or upload data to the PC. The information transfers discussed are facilitated by using libraries supplied by SPECTRUM. Further documentation on the libraries and their usage can be found in the *TMS320C30 System Board User's Manual* [22, Chapter 3, pp. 41–56; Appendix C, pp. 121–158].

F.1.1 Downloading Code to the DSP Subsystem

To download code to the DSP several things must be done first. Obviously, the code must be written and compiled. Coding can be done in Texas Instrument's C. Next, the DSP System Board must be activated. Once that is done, it is possible to download the code. Upon completion of that task, code execution may begin by means of a reset. The board select, code download, and reset can be accomplished using C library functions supplied by SPECTRUM.

To select the DSP System Board, it is merely necessary to call the `SelectBoard` library function. This function has one unsigned short value as an argument. This value should contain the base address of the System Board. The function returns an unsigned short. If the base address supplied is invalid, the function returns `0`. In the other case, that is if the address is valid, the function returns the base address. The net action of the `SelectBoard` function on the DSP is to cause a reset of the processor, leaving the processor looping at address `0xC0`. It should be noted that this must be the first function called from the SPECTRUM library, as it configures the library to talk to the selected board.

Once the board has been selected, it is possible to download COFF-format object files to the board. This is accomplished via the library function `coffLoad`, taking a pointer to an unsigned character as an argument and returning an integer. The pointer argument points to a string containing the filename of the COFF-format file. The function returns an integer 0 on success and a 1 on failure. The details of what this function accomplishes are not detailed by SPECTRUM.

Calling `Reset` following a `coffLoad` function call begins program execution. There is no argument to the function, and it returns an unsigned short. The return value does not hold much interest in the present discussion.

F.1.2 The DSP Memory Map

The memory map for the DSP subsystem is somewhat configurable by software. The file *MAP.CMD* defines the storage locations for different code segments, and if desired for specific variables. The memory allocation is given in Table F.1. The file is listed in Appendix J.

BANK	Size(words)	Address(hex)
BANK0	64K	000000–00FFFF
BANK1	64K	010000–01FFFF
BANK2	64K	020000–02FFFF
BANK3	64K	030000–03FFFF
Memory Expansion Connector	7936K	040000–7FFFFF
DSPLINK	8K	800000–801FFF
Reserved	8K	802000–803FFF
Analog I/O & PC Interrupts	8K	804000–805FFF
Reserved	8K	806000–807FFF
On-Chip Peripherals	6K	808000–8097FF
RAM0	1K	809800–809BFF
RAM1	1K	809C00–809FFF
Memory Expansion Connector	8152K	80A000–FFFFFF

Table F.1: DSP Memory Map.

The most noteworthy point about the memory map is the ability to put specific variables in specific locations. Combining this with the fact that **BANK3** is a dual-port memory, it becomes possible for the PC-AT to monitor certain memory locations in the DSP memory. This can be used to pass arguments, pass data, or maintain a semaphore signal to keep the PC and DSP in lockstep. All this can be done without the PC interfering with the operation of the DSP. It is through this mechanism that the measurement data is passed.

F.1.3 Passing Data and Arguments

Reading and writing data to the DSP board can be accomplished using the C library functions supplied by SPECTRUM. All data transfers rely on knowing the proper location to read or write. Because of this, it is necessary to dedicate fixed memory

locations. This can be done at compilation by editing the *MAP.CMD* file as discussed below.

Fixing Variable Locations in Memory

When transferring data between the PC and the DSP memory, it is convenient to allocate fixed locations to specific variables. Otherwise, the compiler would not always put the same variable in the same location. This would cause problems for the PC as it has no way of knowing about the actions of the compiler. Although not strictly required, it is advisable that the fixed locations reside in **BANK3** of the DSP memory. This bank is a dual port memory, allowing reads and writes by the PC without halting the DSP processor.

An efficient implementation is to determine which variables need to be passed between the PC and the DSP. These variables can be forced into fixed memory locations by properly instructing the compiler. As mentioned above, these locations should be in **BANK3**, the dual port memory. Putting variables in fixed locations does not affect the operation of the DSP. It does however make it possible for the PC to directly alter the contents of a given variable. All PC needs is the memory location where the variable is stored. Writing or reading that location directly changes the variable value. This can be used by the PC to send parameters to the DSP code, or to retrieve data stored in the DSP.

Fixing memory locations is a relatively simple task. In the file *MAP.CMD*, there is a definition region called `SECTIONS`. In this, there is a definition of the section "`.bss`". In the "`.bss`" definition, lines can be added instructing the linker to put certain variables in fixed locations. These lines have the following syntax.

```
_Variable0 = .;   . += 1;
_Variable1 = .;   . += 1;
```

Each line of definition does two things. First, the variable `Variable0` (note the underscore is not part of the variable name) is assigned to the memory location pointed to by the ".". If this is the first line of the "`.bss`" definition, it points to the beginning of the "`.bss`" section. After the variable is assigned to a memory location,

the pointer "." is incremented. The next line assigns `Variable1` in a similar manner. As many variables as desired can be assigned locations like this.

Writing to the DSP Memory

Writing to the DSP memory can be accomplished using one of the following library functions: `PutFloat`, `WrBlkFlt`, `Put32Bit`, `WrBlk32`, `PutInt`, or `WrBlkInt`. Of these, the current system only uses `Put32Bit` and `WrBlkFlt`, and so only these two functions will be discussed. Details of the other functions can be found in the *TMS320C30 System Board User's Manual* [22, Chapter 3, pp. 41–56; Appendix C, pp. 121–158].

The function `Put32Bit` writes a 32 bit value to the DSP memory. This is done without any conversion on the 32 bit value. The function takes three arguments. The first arguments is an unsigned long integer storing the address to be written. The second argument is an unsigned short integer which indicates the type of memory to be accessed. With the current system, only the dual port memory is used. The third argument is an unsigned long integer which holds the value to which to write. The function returns an integer 0 if successful and a -1 on a failure.

The second function used in writing to the DSP memory is `WrBlkFlt`, which writes an array of floating-point values to the DSP memory. This write is done with a conversion from IEEE floating-point format (which is used in Microsoft C) to TMS320C30 floating-point format. There are four arguments to this function. The first argument stores an unsigned long integer pointing to the first address to which to write. The second argument is an unsigned short integer indicating the type of memory to be accessed which for this system is dual port memory. The third argument is an unsigned short storing the number of values to be transfered. The fourth argument is a pointer to an array of floating-point numbers. The first element of the array will be written to the first DSP memory location. The second element will be written to the next location, and so forth. The function returns an integer 0 if successful and a -1 if unsuccessful.

A few comments should be made about `WrBlkFlt`. In the current system, `WrBlkFlt` is used for one word writes to the DSP memory. It may seem strange to do a one

word block write. This is done merely to keep parallelism with the read operations where there is a complication in doing a single word transfer. The complication is discussed in the paragraphs below.

Reading from the DSP Memory

Reading from the DSP memory can be accomplished using one of the following library functions: `GetFloat`, `RdBlkFlt`, `Get32Bit`, `RdBlk32`, `GetInt`, or `RdBlkInt`. Of these, the current system uses only `RdBlkFlt`, and so only this function will be discussed. Details of the other functions can be found in the user's manual [22, Chapter 3, pp. 41–56; Appendix C, pp. 121–158].

`RdBlkFlt` reads an array of floating values from the DSP memory to the PC memory. The values are converted from the TMS320C30 floating-point format to the IEEE floating-point which is used by the PC. The function has four arguments. The first arguments is an unsigned long integer specifying the address from which to start reading values. The second is an unsigned short integer indicating the type of memory to be accessed which is generally dual port memory. The third argument, an unsigned short integer, specifies the number of values to read from memory and the fourth argument, a pointer to an array of floating-point values, points to the beginning of the destination array. The function returns 0 if successful and -1 if unsuccessful.

The function `RdBlkFlt` transfers an array of floats. This function can also be used to transfer a single floating-point number by reference instead of by value. Normally when passing by value, Microsoft C converts to double precision a floating-point number passed between functions. When using the function `GetFloat` to retrieve a single floating-point number, to avoid the conversion to double precision it returns its value as an unsigned long integer. This integer representation can then be coerced to a floating-point number. To avoid this circuitous manner of passing values, it is easier to pass pointers which do not undergo any conversion. Thus, passing a single value is more easily accomplished by passing it as an array using `RdBlkFlt`.

F.2 The 4 Channel Analog I/O Board

The 4 Channel Analog I/O Board incorporates two digital-to-analog converters (DAC's) and a single analog-to-digital converter (ADC). However, the ADC can sample four inputs by means of a multiplexer in the signal path. In order to get samples taken from the same instant in time, the inputs to the board are first sampled with a sample-and-hold and then sent to the multiplexer. Details of the conversion process are given in Section H.1. The I/O Board also has a timer but this timer is not used.

To understand the function of the board, in particular as it applied in Section H.1, a few things must be discussed. Section F.2.1 outlines the basic configuration of the board while Section F.2.2 explains how the board fits into the DSP memory map and the functions of each pertinent location. The original documentation can be found in the *4 Channel Analog I/O Board User's Manual* [19], and the *TMS320C30 System Board Technical Reference Manual* [21, Chapter 5, pp. 29–31; Appendix A].

F.2.1 Configuring the 4 Channel I/O Board

The 4 Channel I/O Board operates off the DSPLINK bus. Therefore, this bus must be connected, using the 50-way connector, to the DSP motherboard. As a result of being on the DSPLINK bus, the card utilizes that portion of the memory map allocated to the DSPLINK interface. There are four links that need to be set in order for the board to function properly in the overall system. Link `Lk1` sets the base address of the board relative to the beginning of the DSPLINK address space. The options for the base address are 0x800000, 0x800004, 0x800008, and 0x80000C by setting `Lk1` to the "a", "b", "c", or "d" position respectively. The current system chooses 0x800008 for the base address.

There are three other links necessary to properly configure the board. Links `Lk2` and `Lk3` set the output ranges for the two DAC's on the I/O Board. These are normally set for a ± 2.5 volt output range, by setting the links to position "b". The other option would be for a ± 5.0 volt output range.

Link `Lk4` controls the method by which the DAC input registers are updated. The

options are for update on generation of a "Load DAC" signal, transparent output, or update on the timeout of a counter or some other external trigger event. It is this third option which is chosen and a conversion is started by an external trigger. Link `Lk4` should be set to position "c" for this option.

F.2.2 The I/O Board Memory Map

As mentioned above, the I/O Board fits into the DSPLINK segment of the DSP memory map. For reasons discussed in the previous section, the current system puts the base address of the 4 Channel I/O Board at 0x800008. The base address memory maps to the I/O Board's **Control Register** on a write and its **Status Register** on a read. The next address functions as a **Timer Register** on a write. Address 0x80000A, serves as an input register for the first DAC on a write and an ADC result register on a read. Similarly, address 0x80000B functions as an input register for the second DAC on a write and an ADC result register on a read. Further documentation on the functions of each register can be found in Section H.1 and in the *4 Channel Analog I/O Board User's Manual* [19].

Appendix G

The Position Sensing System

The position sensing system is composed of a rotary shaft encoder and a decoder. The DSPLINK interface is used to read values from the decoder into the DSP subsystem. A value indicating the position is then stored and can be read by the PC-AT subsystem. This appendix gives more information on comments made in Sections 4.2 and 4.3.2. It first treats the workings of the shaft encoder and decoder followed by a treatment of the interactions between the decoder and the DSP subsystem.

G.1 The Rotary Encoder/Decoder

The rotary encoder/decoder system is built around the Canon R-2A shaft encoder. This encoder is mounted inside the stator with an axial shaft protruding into the rotor where it is fastened. The encoder produces two sine waves which are 90° degrees out of phase and both with a frequency of 65,536 cycles per mechanical revolution. This signal was originally intended to be decoded by the Canon C-I-16-1 interpolator board, which would have provided a resolution of 1,048,576 locations per cycle. However, this proved unworkable as the interpolator was not immune to the electromagnetic interference (EMI) generated by the servo system. Thus, a new decoder was constructed. This section intends to explain the workings of the new decoder board.

Before delving into the schematics and logic equations of the new decoder, it is best to examine how the rotary encoder works. The rotary encoder generates the

two phase-shifted sinusoidal waveforms as described above. As the shaft rotates, the phase angle of the sinusoids rotates but with an angular frequency of 65,536 Hz. One can note that when the shaft turns in the clockwise direction (looking from the rotor side) that the A-phase leads the B-phase by 90° and when the shaft rotates in the counter-clockwise manner, the A-phase lags the B-phase by 90°. With this information in hand, it was decided to convert the sine waves into square waves by comparing them with zero, using a comparator with a small amount of hysteresis. A logical inversion was also done during the comparison. Taking these square waves, it was possible to construct an algorithm for determining if the shaft was moving clockwise or counter-clockwise. It was decided that a position change would only happen when the digital version of the B phase input was a logic '0' and when the digital A phase underwent a transition. If the transition was from a '1' to a '0' then the shaft was traveling in a counter-clockwise direction and a counter would be incremented, indicating that the next position had been reached. If the transition was from a '0' to a '1' then the shaft was rotating in a clockwise manner and the counter would be decremented. This scheme has noise immunity for a number of reasons. First, the digital levels are harder to corrupt than the analog levels and even though the analog voltages still remain as inputs, the integrity of their levels doesn't matter much since interference will only result in a slight phase error (always less than one count). Additionally small amounts of noise will have little effect because of the hysteresis in the comparator. Second, the phase based algorithm requires that the proper sequence of phasings be generated before a count will occur. That means that the phases must follow a four-step sequence. Third, In order to implement the algorithm it was necessary to create a finite-state machine that could keep track of the transitions in A and B. This system is synchronous with a 1 MHz clock. This added element of synchrony also helps eliminate errors as only disturbances which are coincident with the clock will ever be noticed by the system. The odds of an EMI disturbance following the proper sequencing in synchrony with the clock is extremely slim. It has however been observed that it is possible to occasionally be off by a few counts after a full revolution of operation. Yet, performance is good enough for an

experimentation system which is zeroed occasionally.

The ripple reduction algorithm discussed in this thesis required the existence of well defined absolute positions. Creating a zero position using the above scheme was not an easy task. Since the position was only generated by a counter, any powering-down of the system would cause the position to be lost. Also, since there are occasional errors, it is necessary to have some way to re-zero the system. Fortunately, the rotary encoder has eight bits of absolute position information in grey code format. The zero position was defined as the point where the decoded bits, now in binary coded decimal format, changed from being all zeros to all ones. At that transition, the counter was zeroed.

Schematics for the decoder circuit are shown in Figures G-1 and G-2. Following the schematics follows the "palasgn" code for programming the 16V8 PAL devices. Table G.1 lists the input signals coming from the rotary encoder and the pin numbers. The pin numbers correspond to the JRC25PG-24 family connectors (manufactured by Hirose) which the encoder uses. The schematics also refer to Jumper J5 which connects between the decoder hardware and the DSPLINK interface. Jumper J5 has a DIP connector on the decoder end. The wire numbers follow the normal ordering convention for DIP connectors. Jumper J6 connects between the JRC25PG-24 connector and the decoder hardware. Again, the pin numbers follow the normal ordering convention for DIP connectors on the decoder end.

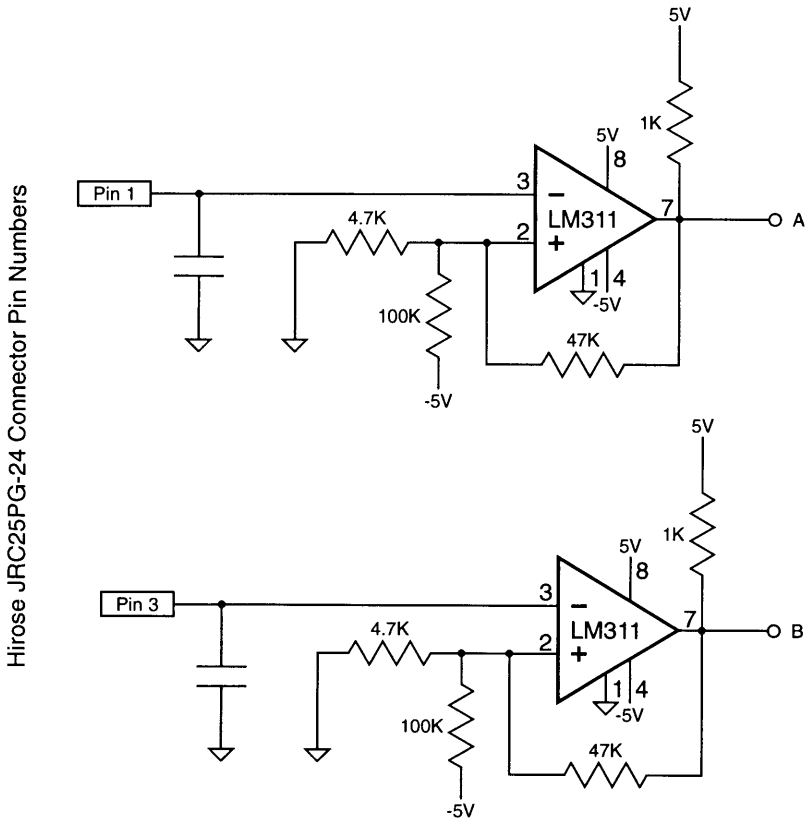

Figure G-1: Rotary Decoder Circuit Schematic (Sheet 1).

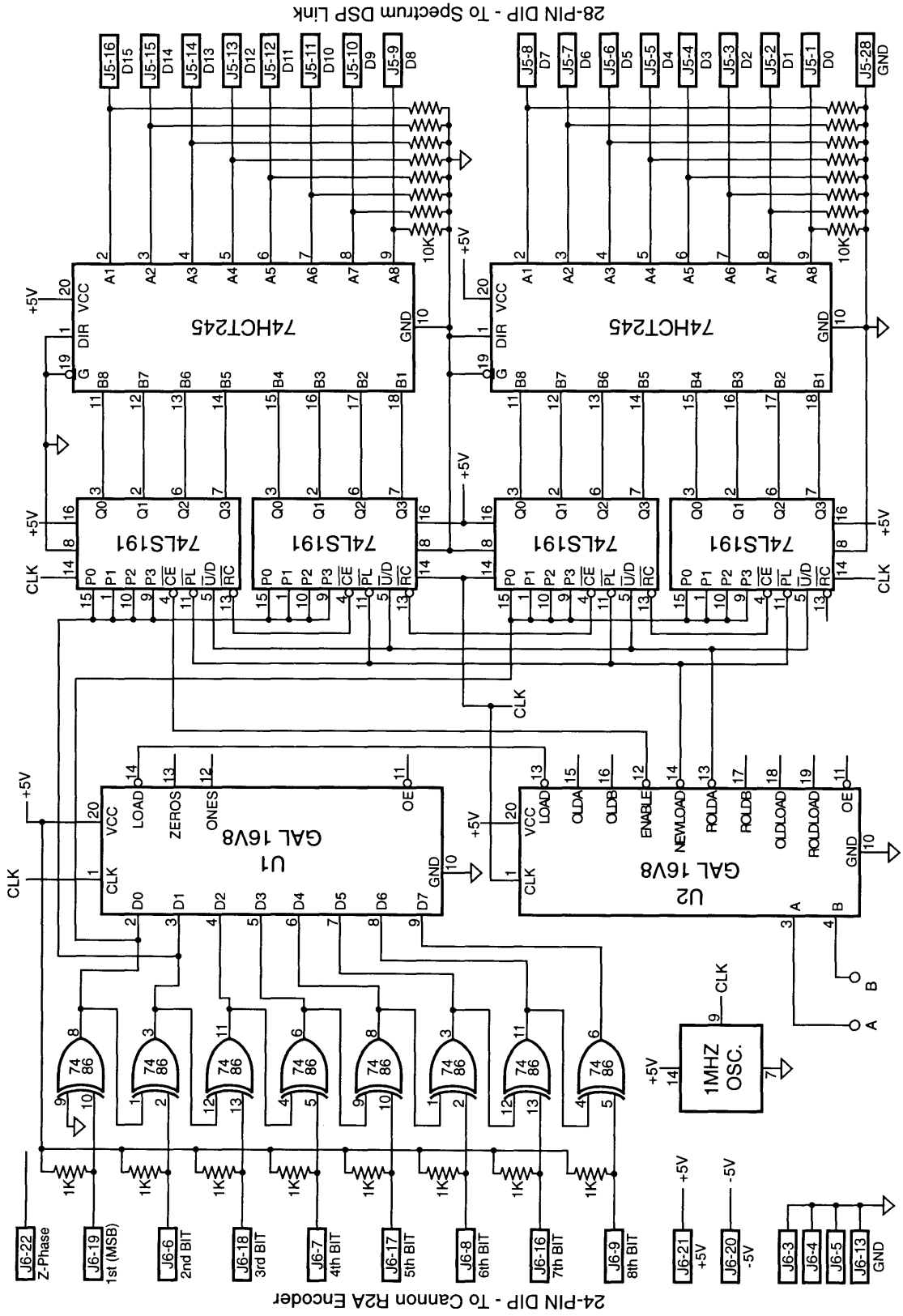

Figure G-2: Rotary Decoder Circuit Schematic (Sheet 2).

```
16v8    D /D /C C C C C C
palasgn output from source file: loadgen.eqn

Massachusetts Institute of Technology
CLK       D0        D1        D2        D3        D4        D5
D6        D7        GND       /OE       ONES      ZEROS     /LOAD
NC        NC        NC        NC        NC        VCC

LOAD =  /D0*/D1*/D2*/D3*/D4*/D5*/D6*/D7*ONES

ONES :=          D0*D1*D2*D3*D4*D5*D6*D7

/ZEROS :=        /D0*/D1*/D2*/D3*/D4*/D5*/D6*/D7
```

Program G.1: Palasgn Code for GAL16V8 U1.

```
16v8    /C /D /C D D D D D
palasgn output from source file: phase4.eqn

Massachusetts Institute of Technology
CLK       /LOAD     A         B         NC        NC        NC
NC        NC        GND       /OE       /ENABLE   /ROLDA    /NEWLOAD
OLDA      OLDB      ROLDB     OLDLOAD   ROLDLOAD  VCC

ENABLE =        OLDA*/OLDB*/ROLDA*/ROLDB+
       /OLDA*/OLDB*ROLDA*/ROLDB

OLDA :=         A

OLDB :=         B

ROLDA :=        OLDA

ROLDB :=        OLDB

OLDLOAD :=      LOAD

ROLDLOAD :=     OLDLOAD

NEWLOAD =       OLDLOAD*/ROLDLOAD
```

Program G.2: Palasgn Code for GAL16V8 U2.

Pin #	Function	Pin #	Function
1	A-phase signal	13	3rd Absolute Position Bit
2	GND	14	4th Absolute Position Bit
3	B-phase signal	15	5th Absolute Position Bit
4	GND	16	6th Absolute Position Bit
5	Z-phase signal	17	7th Absolute Position Bit
6	GND	18	8th Absolute Position Bit (LSB)
7	+5V	19	NC
8	GND	20	+5V Supply Sense
9	-5V	21	GND Sense
10	GND	22	-5V Sense
11	1st Absolute Position Bit (MSB)	23	Shield
12	2nd Absolute Position Bit	24	Case

Table G.1: Pin Out for the JRC25PG-24 Series Connectors.

G.2 Interfacing the Position Measurement System with the DSP Subsystem

The position measurement subsystem has been assigned a location in the memory map of the DSP subsystem. This makes it possible to read a value from the position subsystem by merely reading a memory location in the DSP memory address space. This occurs because the position measurement subsystem interfaces via a DSPLINK interface card. The document entitled *DSPLINK Prototype Interface Module* [20] has more information on the DSPLINK. The *TMS320C30 System Board Technical Reference Manual* [21, Chapter 5; Appendix A] also provides more information. The base address of the card has been chosen to be at `0x800004` and it has an address range up to `0x800007`. This is done by setting the jumper `LINK 2` on the DSPLINK interface to setting "b". The decoder returns its data to the base address of the DSPLINK card. Thus any read from location `0x800004` will return the 16 bit position value generated by the position decoder.

To make the code more readable, DSP address locations have been assigned names by way of global declarations. Below is a code fragment which defines four locations, covering the four locations that the DSPLINK interface can access. The code fragment occurs in the header of the DSP code.

```
#define CANON0  ((unsigned int *) 0x800004)
#define CANON1  ((unsigned int *) 0x800005)
#define CANON2  ((unsigned int *) 0x800006)
#define CANON3  ((unsigned int *) 0x800007)
```

If one analyzes the decoder hardware closely, it will be noticed that the decoder provides its data to all four memory locations. Thus any one of the four locations can be read to access the rotor position. Customarily, only the base location is read though. This is accomplished with the code fragment below.

```
posl=*CANON0;
posh=*CANON0;    /* Dummy variable; Don't leave posh unassigned */
                 /* or the compiler will mess up the placement of */
                 /* variables in specific memory locations.  At least */
```

```
                /* this appears to be happening.  */

posl16=posl>>16;
posr=2*pi*posl16/65536.0;
```

This section of code reads memory mapped address location `0x8000004`, the decoder value, and stores the value in the variable `posl`, which resides in the dual-port memory bank, **BANK3**. After that, the location is read again and stored in the variable `posh`. This would appear to be senseless, however it works around a bug in the compiler. Originally when the interpolator board was used, 24 bits of information were provided. This took two locations of memory. Specific locations in the memory map were allocated to hold the values. When the interpolator board was abandoned, the memory map was left unchanged. However the compiler seems not to respect the memory map allocations if one of the variables is unused. The simplest work-around for the problem was just to access the memory location. To further complicate the situation, a simple assignment will not suffice. If nothing else is done with the value, the optimizer recognizes that the assignment is never used and it removes the assignment. Thus, the assignment was placed in a loop which sufficiently confuses the optimizer.

Once the position has been read into a variable with the above code, the information is stored in the sixteen most significant bits. Thus it is necessary to shift right sixteen times. This value can then be converted to a radian measure of angle. The radian measure of angle can then be introduced into the control algorithms.

G.3 Interfacing the Position Measurement System with the PC-AT Subsystem

The position measurement system can be indirectly interfaced with the PC-AT subsystem by way of the DSP subsystem. Using the code discussed in the previous section, the position can be read from the decoder into a specific memory location in the DSP memory map. Using the techniques discussed in Section F.1.3, the value in

the DSP memory can be read by the PC-AT subsystem. An appropriate conversion on the value can then be done by the PC. A short discussion of the necessary code will now be done. Note that no additional code is necessary for the DSP.

The PC first defines global variables which hold the locations in the DSP memory map where the variable of interest, `posl` and `posh`, are located.

```
#define  COMM          0x30000    /* Start of .bss memory area.    */
#define  POSL          COMM0 + 0  /* Absolute memory locations     */
                                  /* reserved in LSICMAP.CMD.      */
#define  POSH          COMM0 + 1
```

The position is read from the DSP memory as a 32 bit value, which should then be shifted right sixteen times. The value can then be converted to degrees or radians. The line of code below accomplishes the transfer of data.

```
positl=Get32Bit(POSL,DUAL)>>16;
```

The following line of code gives an example of the type of conversion on the data that can be done. The line prints the current angle in degrees.

```
printf("Degrees: %f\n",360.0*((int) positl)/65536.0);
```

Appendix H

The Drive Electronics

This appendix supports statements made in Section 4.3.1.

The responsibility of the drive electronics subsystem is to deliver a precise current to each motor winding. The current desired is calculated by the control algorithm on the DSP. The DSP then writes its data to the Digital-to-Analog Converters (D-to-A, or DAC) of the drive electronics subsystem. These in turn produce analog voltages proportional to their digital inputs. These analog signals pass through a stage of isolation amplifiers which provide galvanic isolation and these outputs drive a modified Allen-Bradley servo system. The servo system provides current outputs which are fed into the windings of the motor. The servo system also provides feedback voltages which are proportional to the output currents. These feedback voltage are then galvanically isolated by another stage of isolation amplifiers whose outputs then serve as inputs to a muxed-input Analog-to-Digital Converter (A-to-D, or ADC). The outputs of the ADC are digital representations of the winding currents. These digital feedback values are then used in a feedback loop to obtain greater regulation on the winding currents (while reducing the system dynamic bandwidth). The digital values can also be stored for retrieval by the PC.

The following sections detail the operation of each step in the above signal flow. The above description provides a more easily understood description of the signal flow. However, because of details of the implementation, it is more accurate to start the description with the reading in of the feedback voltages by the A-to-D converter.

This is the event that triggers the beginning of a new cycle.

H.1 Controlling the ADC's

The drive electronics system's cycle begins with the A-to-D conversion. To understand the conversion cycle, it is necessary have an understanding of the conversion hardware and how to interface to it. Sections H.1.1 and H.1.2 detail the interfacing to the motherboard converter system and the 4 Channel I/O Board converter system, respectively. Following those sections, Section H.1.3 and Section H.1.4 detail the initialization procedures for the motherboard and 4 Channel Analog I/O Board.

H.1.1 An Introduction to the Motherboard Converter System

The ADC's and DAC's are controlled via memory mapped ports in the DSP memory space. This is true both of the converters on the motherboard and on the daughter-card. The motherboard's analog I/O interface occupies addresses `0x804000`, `0x804001`, and `0x804008`. These locations have been given global definitions with names `IASREF`, `IBSREF`, and `SOFTCON` respectively. The code where this is done follows below.

```
#define IASREF  ((unsigned int *) 0x804000)
#define IBSREF  ((unsigned int *) 0x804001)

⋮

#define SOFTCON ((unsigned int *) 0x804008)
```

Writing to `IASREF` writes data into the motherboard's Channel A DAC. Reading from the same location reads the output of the Channel A ADC. The operation for location `IBSREF` is similar except the actions occur to the motherboard's Channel B. It should be noted that since the ADC's and DAC's generate sixteen bit data words, all the bits of the memory location are utilized by the ADC's and DAC's. The global

definitions are very suggestive of where each channel interfaces in this particular application. Channel A which uses the memory mapped location named `IASREF` is connected to the signal lines for the stator A winding. Similarly, Channel B, operating from memory mapped location `IBSREF`, is connected to the stator B winding. The memory location `SOFTCON` causes a software-initiated A-to-D conversion of both the A and B channels upon either a read or a write. The current system does not use this method of conversion cycle control. One note should be made regarding reading and writing the ADC's and DAC's. The ADC's must be read prior to writing the DAC registers, otherwise the value just written to the DAC's will be read back as if it were the ADC output.

H.1.2 An Introduction to the 4 Channel I/O Board

The 4 Channel I/O Board occupies a configurable address in the DSP memory. The exact address location of the I/O Board is selected via jumper `Lk1` on the I/O Board. The link is set so that the I/O Board has a base address of `0x800008`. It should be noted that the 4 Channel I/O Board interfaces with the motherboard via the DSPLINK interface and thus resides in the memory space allocated to the DSPLINK system, addresses `0x800000` through `0x801FFF`. On a write operation, the base address serves as a **Control Register**, `IOCREG`, and on a read operation it functions as a **Status Register**, `IOSTAT`. The next memory location, `0x800009`, serves as a **Timer Register**, `IOTCTL`, on a write and is unused on a read. For this system's purpose, this timer goes unused since the motherboard timer suffices. However, a conversion cycle is initiated by writing to the **Timer Register**. The next location, `0x80000A`, provides an input register for the first DAC on a write, and an ADC result register on a read. The DSP source code uses the global reference of `IARREF` to refer to this location on a write to the DAC and `IMEAS` to refer to this location on an ADC read. These names are appropriate because the `IARREF` memory location controls the DAC connected to the rotor A channel. The `IMEAS` location is connected to the one output of the ADC which measures all the currents. Location `0x80000B` provides an input register for the second DAC on a write. The source code refers to the DAC

register by the reference IBRREF. The ADC's and DAC's on the I/O Board have 12 bits of resolution. Therefore not all the bits to and from the conversion hardware are meaningful. In the case of read operations from the ADC registers, bit 15 is used for a sign bit followed by eleven data bits followed by four replicas of the sign bit. For writes to the DAC registers, bit 15 is the sign bit which is followed by eleven data bits. The last four bits of the DAC registers are unused. The code which defines the global references for the 4 Channel I/O Board is given below.

```
#define IOCREG ((unsigned int *) 0x800008)
#define IOSTAT  ((unsigned int *) 0x800008)
#define IOTCTL ((unsigned int *) 0x800009)
#define IARREF ((unsigned int *) 0x80000A)
#define IBRREF ((unsigned int *) 0x80000B)
#define IMEAS ((unsigned int *) 0x80000A)
```

Before operating the DAC's and ADC it is necessary to configure the motherboard and 4 Channel Analog I/O Board to work in the present application. The following sections detail the initialization procedures.

H.1.3 Initializing the Motherboard for Analog I/O

All Analog I/O is synchronized by the motherboard timer, **TIMER1**. To control the timer, first the timer must be reset by writing RSTCTRL, 0x000601, to the **Timer Global Control Register**, TIMECTL, at address 0x808030. Writing 0x000601 has the effect of placing the timer in hold mode with no reset being performed. It also configures the timer such that the timer counter is clocked by a clock internal to the DSP, and such that a timer output pulse is generated by an equality comparison between the period register and the timer counter. The following line of code accomplishes this register write.

```
*TIMECTL=RSTCTRL;
```

TIMECTL and RSTCTRL are defined as global variables in the following lines which occur in the header of the DSP source code.

```
#define TIMECTL ((unsigned int *) 0x808030)
```

```
⋮

#define RSTCTRL 0x000601
```

After the timer is reset, the **Timer Period Register**, which sets the number of clock counts between each timer output pulse, can be set. Each count adds 120 ns of delay between timer output pulses. A typical number of counts is around 2500, corresponding to about 300 μs. This amount of time has been empirically found to be slow enough to allow the DSP software for this application to keep up. The **Timer Period Register** resides at memory location 0x808038. Below are code lines which define the location of the **Timer Period Register** and the final count value. Following that, is the code line that writes the count value to the **Timer Period Register**.

```
#define PERIOD  ((unsigned int *) 0x808038)

⋮

#define COUNT 2500

⋮

*PERIOD=COUNT;
```

Once the **Timer Period Register** has been set, the timer can be enabled. This is accomplished by writing the value SETCTRL, 0x0006c1, to the **Timer Global Control Register**. This control word configures the timer exactly as before but instead of putting the timer in hold mode, it causes the timer to reset to zero and start counting up. This write is implemented with the following lines of code.

```
*TIMECTL=SETCTRL;
```

SETCTRL is defined in the following code line from the header.

```
#define SETCTRL 0x0006c1
```

H.1.4 Initializing the 4 Channel I/O Board

Initializing the motherboard timer sets the entire conversion loop in motion. From this point on, the completion of the motherboard timer will cause a conversion to occur. Thus before starting the conversion loop it is safer to initialize the 4 Channel Analog I/O Board as detailed below. Before beginning the conversion loop by setting the motherboard timer, a standard initialization routine first calibrates the sample-and-hold circuits of the ADC circuit [19, p. 25,p. 19]. To do this, it is necessary to pulse bit 5 of the **Control Register** high. After this, bit 5 of the **Status Register** can be polled to wait for completion of the calibration routine. This action is shown in the code below.

```
*IOCREG=IOCINIT;
*IOCREG=IOCAL;
*IOCREG=IOCINIT;
while(!(*IOSTAT & CALMSK));
```

`IOCINIT`, `IOCAL` and `CALMSK` are defined globally in the code by the following lines. `CALMSK` is merely a bit-mask to look for the status of bit 5.

```
#define IOCINIT 0x000000
#define IOCAL 0x200000

⋮

#define CALMSK 0x200000
```

After the calibration is completed, the initialization procedure writes a control word, `IOSTRT` to the **Control Register**. This actions puts the ADC sample-and-hold circuits in sample mode. This must be done prior to the beginning of an ADC conversion cycle to allow for the acquisition of the new input values. Also, this write inhibits the timer on the I/O Board, which is necessary in order to use software initiated conversions [19, p. 17,p. 19]. The code which writes the value is shown below.

```
*IOCREG=IOSTRT;
```

`IOSTRT` is defined by the following global variable.

```
#define IOSTRT  0x400000
```

H.1.5 Controlling the Motherboard ADC's

The control of the ADC's begins when the motherboard timer, **TIMER1**, counts to the `COUNT` value (as defined in Section H.1.3), at which point the timer resets and starts again. Also at this time, the motherboard ADC's perform a conversion. The decision to operate the motherboard ADC's from the timer is hard-wired into the motherboard by installing jumper `LK2a` [21, Chapter 4, pp. 21–23]. Since the inputs of the motherboard ADC's are floating this conversion is meaningless. However, the DSP motherboard interrupt `INT1` is enabled so that when the a motherboard A-to-D conversion is finished, an interrupt is signaled [22, Chapter 2, pp. 35–39] [23, Section 8.1, pp. 8-2–8-9]. When this happens, an interrupt handler can cause the 4 Channel I/O Board's ADC to enter a conversion cycle.

H.1.6 Controlling the 4 Channel I/O Board's ADC

The first action of the interrupt handler is to trigger a conversion cycle on the 4 Channel Analog I/O Board. This software initiated conversion is accomplished by writing to the I/O Board's timer register, `IOTCTL`. This is shown in the code line below.

```
*IOTCTL=0;     /* write used to generate software trig */
```

A few comments need to be made about this as it is not as straight forward as would first appear. In order for a software initiated conversion cycle to triggered by the above write operation two things must be done. First, the I/O Board timer must be inhibited. This generally needs to be done only once somewhere in the initialization portion of the DSP code, as described in Section H.1.4. The second action necessary for a conversion cycle to begin is to put the sample-and-holds in hold mode. This is accomplished by writing the control word `IASMEAS` to the **Control Register**. This control word instructs the I/O Board to continue to inhibit the timer, put the sample-and-holds in hold mode and indicates that the first conversion to take place should be that of sample-and-hold channel 0, which corresponds to the stator A current feedback signal.

After the conversion cycle has been started, it will be normal for the interrupt handler to poll the I/O daughter-card until the status register indicates that a conversion is complete and another byte is available. This is done in the following line of code. It simply checks that bit 7 of the status register is high indicating an end of conversion.

```
while(!(*IOSTAT & ADMSK));
```

ADMSK is a bit-mask defined by a global declaration.

```
#define ADMSK 0x800000
```

After each successful polling, a value is read from the ADC result register IMEAS. Following that, the next command word is written instructing the system to convert the next channel. On the last channel to be converted, the sample-and-hold is commanded to return to sample mode. The return to sample mode happens at the end of the conversion. The code below executes what has been described above.

```
while(!(*IOSTAT & ADMSK));
iasnew=((int) (*IMEAS))>>20;
*IOCREG=IBSMEAS;
while(!(*IOSTAT & ADMSK));
ibsnew=((int) (*IMEAS))>>20;
*IOCREG=IARMEAS;
while(!(*IOSTAT & ADMSK));
iarnew=((int) (*IMEAS))>>20;
*IOCREG=IBRMEAS;
while(!(*IOSTAT & ADMSK));
ibrnew=((int) (*IMEAS))>>20;
```

H.2 Controlling the DAC's

After reading the feedback current values via the ADC, the interrupt handler writes the new output values for the rotor and stator phase currents to the appropriate DAC registers. It should be noted that before writing to the DAC registers, the internal representations of the desired current need to be scaled appropriately. The output bits must occupy the sixteen most significant bits of the 32 bit output word. The

code below accomplishes the write operation. The variables `iasd`, `ibsd`, `iard` and `ibrd` are the command values for the stator A phase current, stator B phase current, rotor A phase current and rotor B phase current respectively.

```
*IASREF=((unsigned int)(-32767.0*iasd/25.5))<<16;
*IBSREF=((unsigned int)(-32767.0*ibsd/25.5))<<16;
*IARREF=((unsigned int)(-32767.0*iard/25.5))<<16;
*IBRREF=((unsigned int)(-32767.0*ibrd/25.5))<<16;
```

Once the new values have been written to the DAC's, they are converted to analog voltages. This conversion does not take place immediately however. It is necessary to get a conversion signal. For the stator currents, the DAC's reside on the motherboard. The motherboard is configured such that the conversion commences at the timeout of the timer, **TIMER1**, as documented in Section H.1.3 and Section H.1.5. The rotor currents use the DAC's residing on the Analog I/O card. Because of the setting of Link `Lk4` the conversion begins when the **Timer Register** is written to, as is done at the beginning of the ADC conversion cycle.

H.3 Isolation Between the DAC's and the Servo System

The analog voltages created by the DAC's are galvanically isolated from the Allen-Bradley servo system. This is done to protect the user from injury and the equipment from damage in the case of a fault in the servo system. The isolation system provides 3500V of isolation. The isolation stage also performs offset and scaling of signals. An illustration of the signal flow and power connections for the isolation system is shown in Figure H-1.

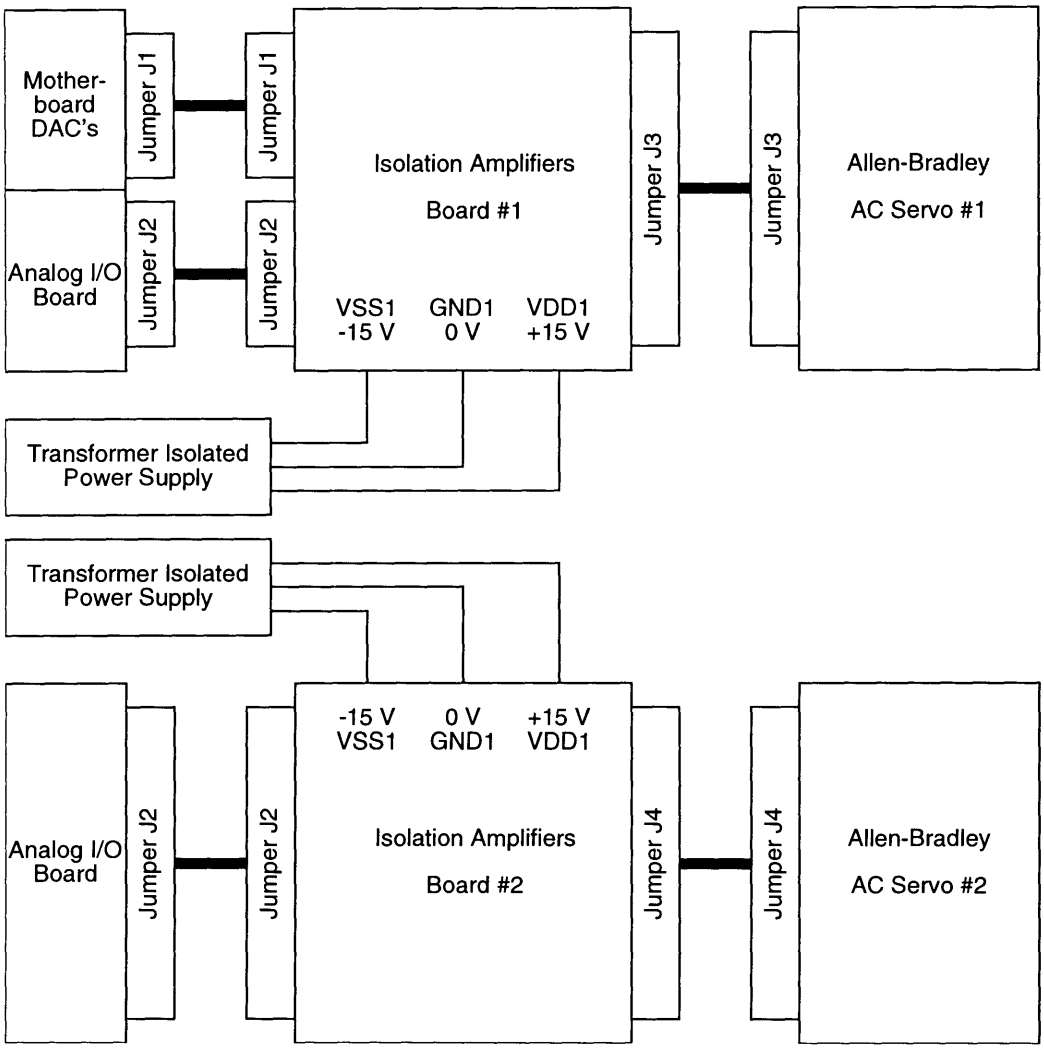

Figure H-1: Isolation Amplifier System.

Figure H-1 depicts four jumpers. Jumper J1 is a 15-pin 'D' type connector. Jumper J2 is a 25-pin 'D' type connector. Jumpers J3 and J4 are 9-pin 'D' type connectors. The following diagrams illustrate the pin assignments for each connector.

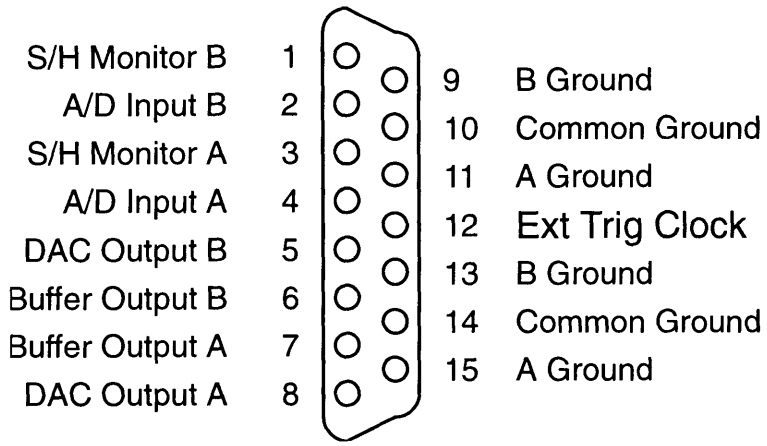

Figure H-2: Jumper J1 Pin Assignments.

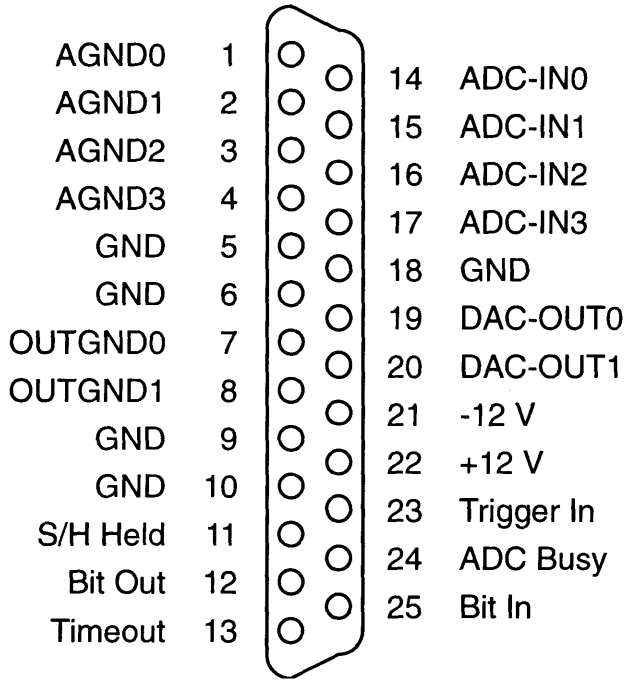

Figure H-3: Jumper J2 Pin Assignments.

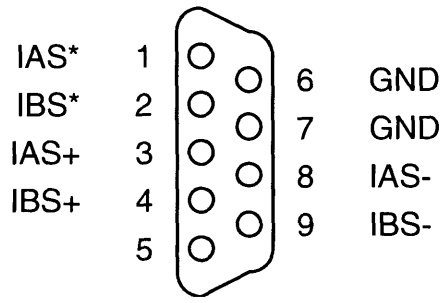

Figure H-4: Jumper J3 Pin Assignments.

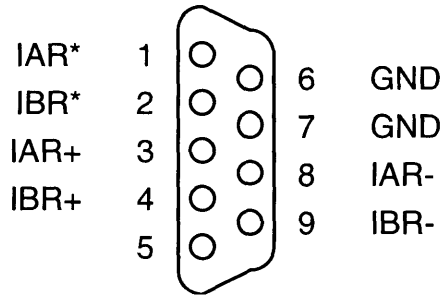

Figure H-5: Jumper J4 Pin Assignments.

For Jumper J1, the important signals are `DAC Output A`, `DAC Output B`, `A Ground`, and `B Ground`. These are the motherboard DAC outputs discussed in previous sections. For Jumper J2, the important signals are `ADC-IN0`, `ADC-IN1`, `ADC-IN2` and `ADC-IN3`, and their corresponding grounds `AGND0`, `AGND1`, `AGND2` and `AGND3`. These are the four inputs to the ADC muxing circuitry on the 4 Channel Analog I/O Board. Also of importance are the two DAC outputs, `DAC-OUT0` and `DAC-OUT1` and their corresponding grounds, `OUTGND0` and `OUTGND1`. These are the Analog I/O Board's DAC outputs. On Jumper J3, all the signals shown in Figure H-4 are used. The signal `IAS*` is the stator A phase command signal to the servo system. Similarly, `IBS*` is the command signal for the stator B phase. The grounds, `GND`, are used as return paths for the circuit. Signals `IAS+` and `IAS−` comprise a differential voltage signal representing the measured stator A phase current. Signals `IBS+` and `IBS−` perform identically for the stator B phase current. Jumper J4 has the same basic function except the quantities of interest refer to the rotor instead of the stator.

Following the signal path in Figure H-1, the DAC output voltages are brought to the isolation system via jumpers J1 and J2. The isolation system then scales and translates the each voltage independently. This is achieved by an offset adjustment potentiometer and a gain adjustment potentiometer. The use of the offset and gain adjustments is explained in more detail in Section H.7. After the scaling and translation, each signal is fed through the AD210 Isolation Amplifier [3], which provides the galvanic isolation. Following the isolation amplifier, there is an LF356 amplifier which inverts the signal, undoing the inversion done by the isolation amplifier. The isolated signals then go out on jumpers J3 and J4 and enter the Allen-Bradley Servo System.

Schematics for the isolation system are shown in Figure H-6. It should be noted that the motherboard DAC isolation is done on a board separate from the Analog I/O board isolation. There is no particular reason for this, but it something of which to be aware.

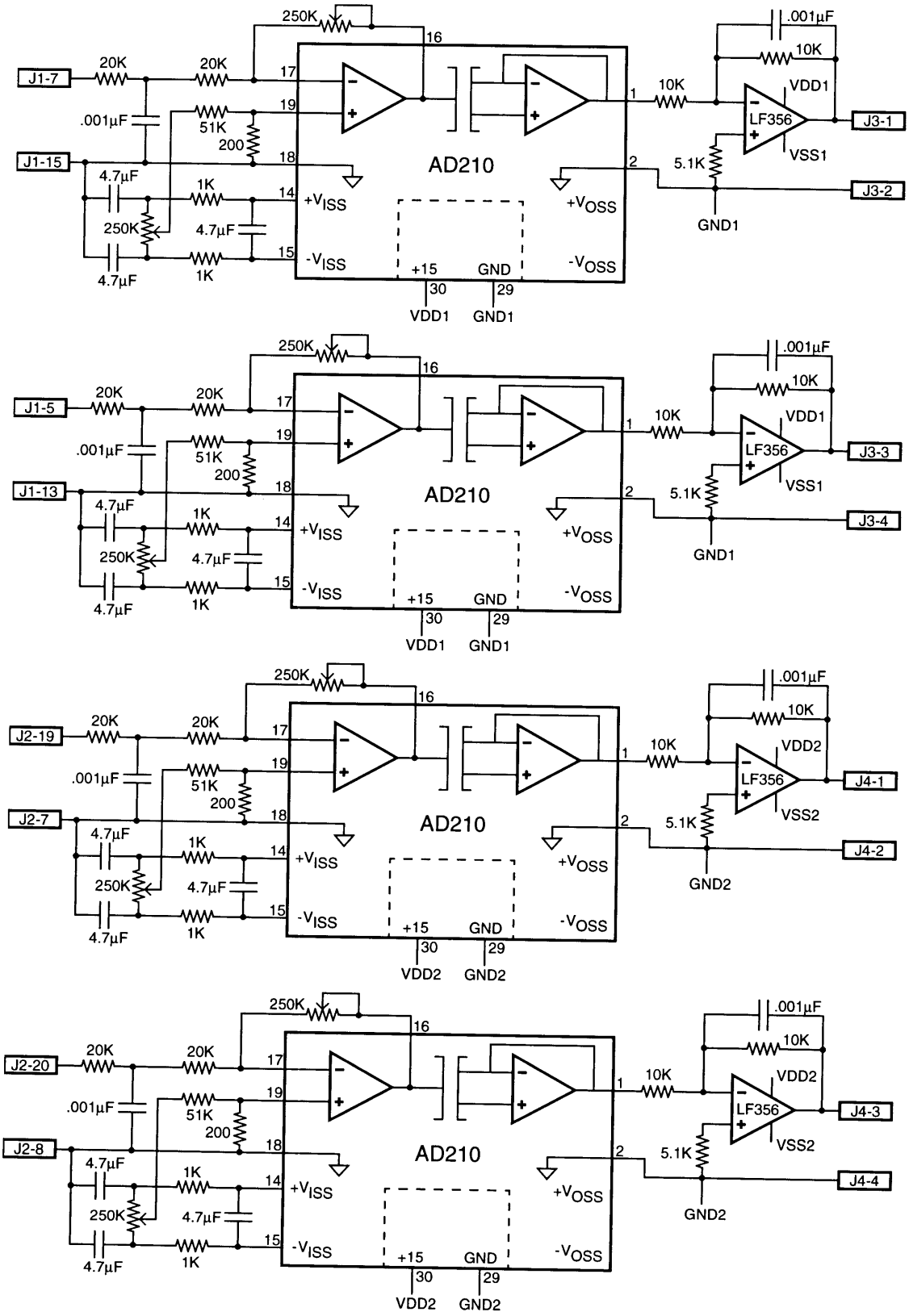

Figure H-6: Schematic for the DAC to Servo System Isolation.

H.4 The Allen-Bradley Servo Amplifier System

Once the command signals leave the isolation subsystem, they enter a modified Allen-Bradley Bulletin 1839 AC Servo Amplifier System. This system is capable of delivering 17 amps of continuous RMS current per winding. Modifications have been made to the servo amp system so that the isolated versions of the DAC outputs can directly command the winding currents, in a proportional manner. Furthermore, the servo system can measure the instantaneous output currents and feed these measurement signals back to the A-to-D via another isolation subsystem. This section will cover the configuration and operation of the servo amplifier system.

The hardware for the servo system is composed of two 1389-AA17 servo amplifier modules, one for the rotor and one for the stator, a 1389-PAT10 power supply and associated chassis module, and a 1389-T100DA isolation transformer. The interconnect of the system is shown in Figure H-7. Detailed information on the associated subsystems can be found in the *Bulletin 1389 User Manual* [1].

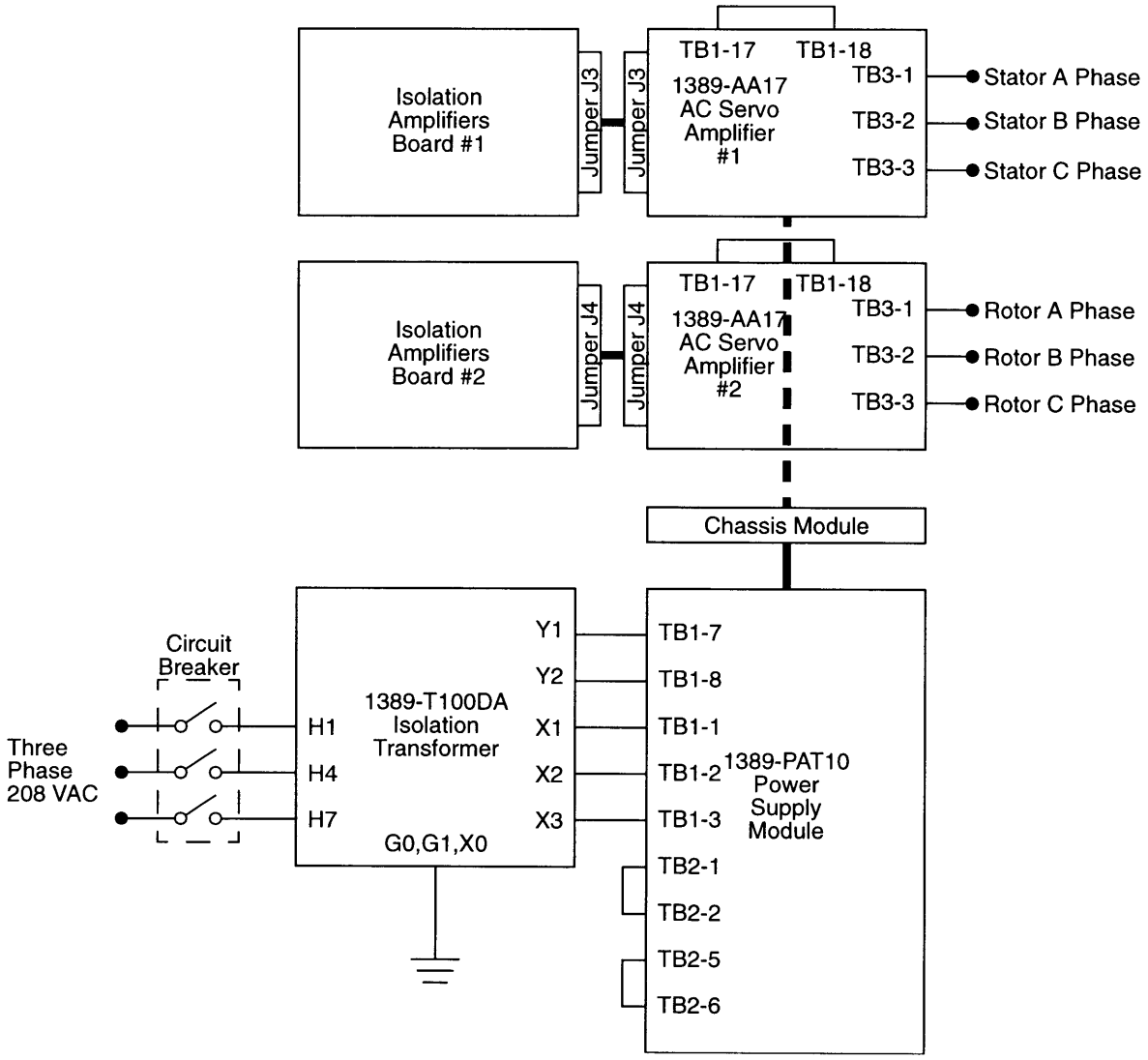

Figure H-7: Servo System Block Diagram.

The isolation transformer is where the power flow begins. The transformer operates with three-phase 208 VAC on the primary in a delta configuration. The transformer has a three-phase 230 VAC output from the Wye-connected secondary. The three-phase output will eventually provide the power to the motor windings. There is also a single-phase 230 VAC output. This phase is used by the power supply module to generate a +12V supply. A schematic depiction of the transformer as it is shipped by Allen-Bradley is given in Figure H-8. A number of jumper connections need to be

made for proper operation. Nodes H2 and H3, H5 and H6, and H8 and H9 need to be jumpered together to form the delta on the primary. The shields need to be tied to the transformer chassis, which should be tied to earth ground. The node X0, the center node of the Wye, should be grounded to the transformer chassis as well.

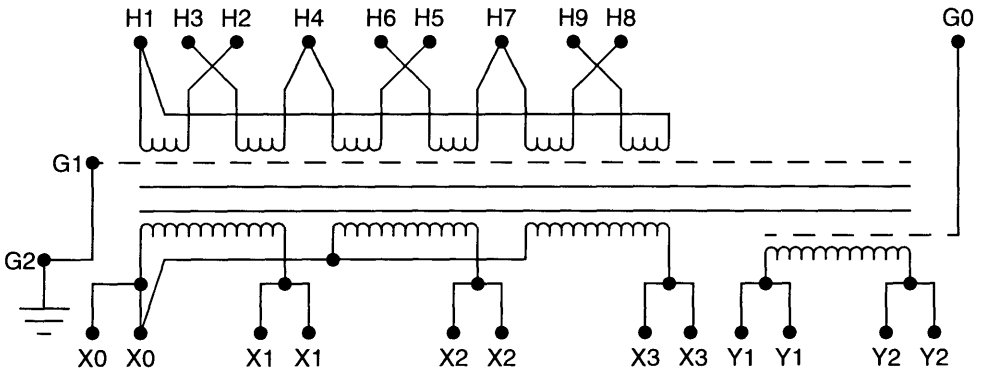

Figure H-8: Schematic for the Isolation Transformer.

The power leaving the isolation transformer enters the power supply module. The exact workings of the power supply module are not known, and it is not necessary to know. The power supply module creates internal supplies that are fed to the servo amplifiers via the chassis module. To configure the power supply correctly, it should be wired as shown in Figure H-7. Of particular note, are the shorting jumpers between TB2-1 and TB2-2, and TB2-5 and TB2-6, which control the reset and enable signals. Inserting both jumpers enables the power supply module. Furthermore, the PAT10 has two internal jumpers which need to be set appropriately. Jumper JU1 needs to be set to Position A and jumper JU2 needs to be set to Position A as well.

Once the power supply module is feeding voltage to the chassis module, it remains for the servo amplifiers to do their work. The servo amplifiers have been modified into mere transconductance amplifiers, whereas normally they have much greater functionality. The input signals come from the isolation system as discussed in Section H.3 and below. The outputs are located on a terminal block on the front face of the servo amp module. These can be wired to the motor windings.

The following modifications have been made to the servo amplifier modules. An

aid in understanding what is described below will be a reference to the *Bulletin 1389 User Manual* [1] and to a fax entitled *1389 Block Diagrams* [2]. On the jumper CNC1, internal to each servo amp, pins 4 and 5 have been removed. A connector has been attached to the inside of each of the servo amplifiers where jumpers J3 and J4 can be attached. From this connector, connections are made to various test-points in the servo amp circuitry. These connections are detailed in Table H.1 for servo amp #1 and Table H.2 for servo amp #2. These modifications change the servo amp so that the command voltage from the isolation amplifier subsystem, either IAS*, IBS*, IAR*, or IBR*, controls proportionally the output current of the servo amp. They also provide differential voltages proportional to the measured output current. These were given the names IAS+, IAS−, IBS+, IBS−, IAR+, IAR−, IBR+, and IBR−. These signals exit the servo amplifier subsystem via the jumpers where they return to the isolation subsystem.

Pin #	Function	Test Point
J3-1	IAS*	TP7
J3-2	GND	TP9
J3-3	IBS*	TP8
J3-4	GND	TP9
J3-5	IAS+	TP3
J3-6	IAS−	TP9
J3-7	IBS+	TP6
J3-8	IBS−	TP9
J3-9	NC	NC

Table H.1: Connections from Jumper J3 to Servo Amp #1.

Pin #	Function	Test Point
J4-1	IAR*	TP7
J4-2	GND	TP9
J4-3	IBR*	TP8
J4-4	GND	TP9
J4-5	IAR+	TP3
J4-6	IAR−	TP9
J4-7	IBR+	TP6
J4-8	IBR−	TP9
J4-9	NC	NC

Table H.2: Connections from Jumper J4 to Servo Amp #2.

In addition to the modifications required to be done to the servo amplifiers, it is necessary to set some jumpers internal to the servo amplifiers. Table H.3 details the settings of the jumpers.

Jumper	Position	Jumper	Position
JP1	A	JP10	A
JP2	A	JP13	B
JP3	A	JP14	B
JP4	A	JP15	A
JP5	B	JP16	A
JP6	A	JP17	A
JP7	B	JP18	A
JP8	B	SW1	F

Table H.3: Servo Amplifier Jumper Settings.

H.5 Isolation Between the Servo System and the ADC

As mentioned in the previous section, the measurements of the output currents are returned to the ADC via the isolation subsystem. Pin assignments can be found in Figures H-2—H-5. The schematics are shown in Figure H-9. They are nearly identical to the schematics shown in Section H.3 except the LF356 components have been removed and the fact that the signal names are different.

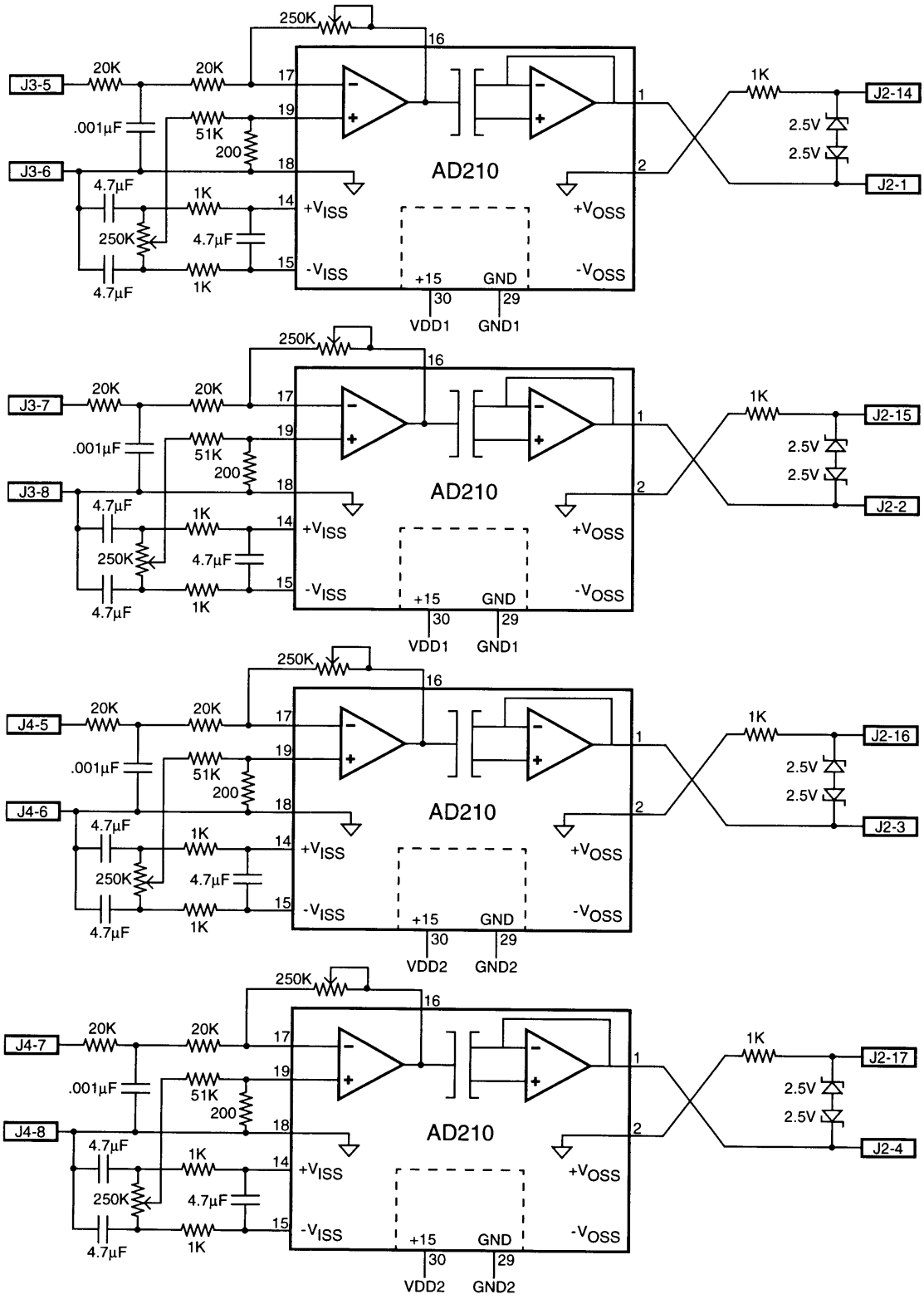

Figure H-9: Schematic for the Servo System to ADC Isolation.

Once the signals return to the ADC the signal loop is completed by a conversion. The fed-back current measurement signals are used to establish a feedback loop on the DSP. This helps to establish a greater control on the DC value of the currents, at the cost of reduced dynamic bandwidth.

H.6 The Drive System Feedback Loop

The drive system has a feedback loop that is used to establish greater DC accuracy. It was noticed early on that the DC regulation on the servo system was poor. The loop works off the instantaneous current as a feedback quantity. This is the current as measured at the servo system which undergoes an A-to-D conversion. Thus the feedback loop is in the digital domain. The block diagram of the loop is shown in Figure H-10. One can see that the inner loop is a low pass filter and the outer loop forces the average value of the output to the command value.

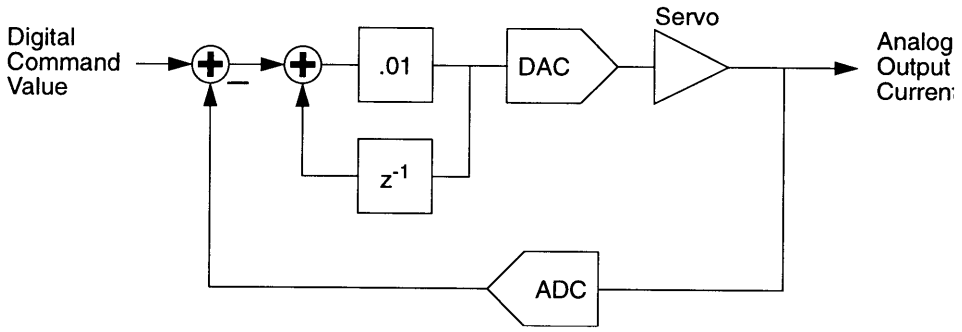

Figure H-10: Block Diagram of Feedback Loop.

Code on the DSP for the feedback loop is listed below. The variables `iasnew`, `ibsnew`, `iarnew`, and `ibrnew` are temporary storage locations for the raw A-to-D converted values. These are then scaled appropriately and stored in values `iasm`, `ibsm`, `iarm`, and `ibrm`. These are the measured currents. The drive currents, `iasd`, `ibsd`, `iard`, and `ibrd`, are then calculated as the low pass filter of the difference between the commanded currents and the measured currents. The drive current

values are clipped if they exceed a reasonable value. This is to prevent the drive values from increasing without limit if the feedback were to be broken or if the servo system were not turned on. The latter problem would otherwise result in a very large initial current at power-on which would decay slowly. Thus the clipping is a way to protect the motor from excessive current.

```
while(!(*IOSTAT & ADMSK));
iasnew=((int) (*IMEAS))>>20;
*IOCREG=IBSMEAS;
while(!(*IOSTAT & ADMSK));
ibsnew=((int) (*IMEAS))>>20;
*IOCREG=IARMEAS;
while(!(*IOSTAT & ADMSK));
iarnew=((int) (*IMEAS))>>20;
*IOCREG=IBRMEAS;
while(!(*IOSTAT & ADMSK));
ibrnew=((int) (*IMEAS))>>20;

iasm=17.5*iasnew/2047.0;
ibsm=17.5*ibsnew/2047.0;
iarm=17.5*iarnew/2047.0;
ibrm=17.5*ibrnew/2047.0;

iasd+=0.01*(iasc-iasm);
ibsd+=0.01*(ibsc-ibsm);
iard+=0.01*(iarc-iarm);
ibrd+=0.01*(ibrc-ibrm);

if (iasd>12.5) iasd=12.5;
if (iasd<-12.5) iasd=-12.5;
if (ibsd>12.5) ibsd=12.5;
if (ibsd<-12.5) ibsd=-12.5;
if (iard>12.5) iard=12.5;
if (iard<-12.5) iard=-12.5;
if (ibrd>12.5) ibrd=12.5;
if (ibrd<-12.5) ibrd=-12.5;

*IASREF=((unsigned int)(-32767.0*iasd/25.5))<<16;
*IBSREF=((unsigned int)(-32767.0*ibsd/25.5))<<16;
*IARREF=((unsigned int)(-32767.0*iard/25.5))<<16;
*IBRREF=((unsigned int)(-32767.0*ibrd/25.5))<<16;
```

H.7 Calibration

Calibrating the servo system is necessary to ensure an accurate relationship between commanded current and output current. The simplest way to calibrate the system is to put a current probe on each winding. Then turn the gains up to the maximum on the isolation amplifiers. Command a current of zero amperes to each winding. Zero the output currents by adjusting the potentiometer on each isolation amplifier until the measured winding current, on the probe meter, is zero amperes. Then null the feedback isolation amplifiers so that the ADC is reading 0 amperes for each winding. Once this is completed, command some maximum output current to each winding. Adjust the gain potentiometers until the proper current is delivered to each winding, as read on the meter. Then adjust the feedback amplifier's gains so that the ADC is reading back the maximum current properly. This should complete the calibration. It is wise, however, to check the calibration at several points to ensure accurate calibration.

Appendix I

The Temperature Sensing System

The temperature sensing system is quite crude, but sufficient to carry out the measurements of Chapter 5. There are four T-type thermocouples situated around the motor, with two on the rotor and two on the stator. Thermocouple T1 is located in one of the slots of the rotor. Thermocouple T2, also on the rotor, is located closer to the water-cooled back-metal. T3 and T4 are in the same positions as T1 and T2 respectively, but on the stator side. Coming away from the thermocouple, the wires pass through a K-type switch box and then as a single wire to a K-type thermocouple meter. The switch box is used to select the channel which is to be measured. Readings are taken manually and recorded in a notebook as read off the meter. Since the actual thermocouple in the motor is a T-type thermocouple and the meter is a K-type thermocouple meter, a conversion needs to be done to obtain the correct temperature reading.

In order to understand the conversion, the system set up must be analyzed. The system is set up so that the switch box and thermocouple meter are located quite far from the motor and thus see the ambient temperature. With this arrangement, the meter will measure a voltage corresponding to the T-type thermocouple voltage plus the Seebeck voltage of the K-type thermocouple at room temperature. The meter itself is cold-junction compensated for K-type thermocouple wire. Figure I-1 illustrates the connections. To a rough approximation, the sum of the voltage across the K-type thermocouple and the K-type thermocouple cold-junction compensation

provides a voltage which nearly equals that of the cold-junction compensation voltage for a T-type thermocouple. Thus, the temperature reading taken using the K-type meter can be converted to an equivalent cold-junction-compensated T-type thermocouple voltage. This is done by looking up the K-type cold-junction-compensated voltage corresponding to the readout temperature. This voltage can then used as the cold-junction-compensated T-type thermocouple voltage. Using this voltage, the actual temperature can be looked up in tables.

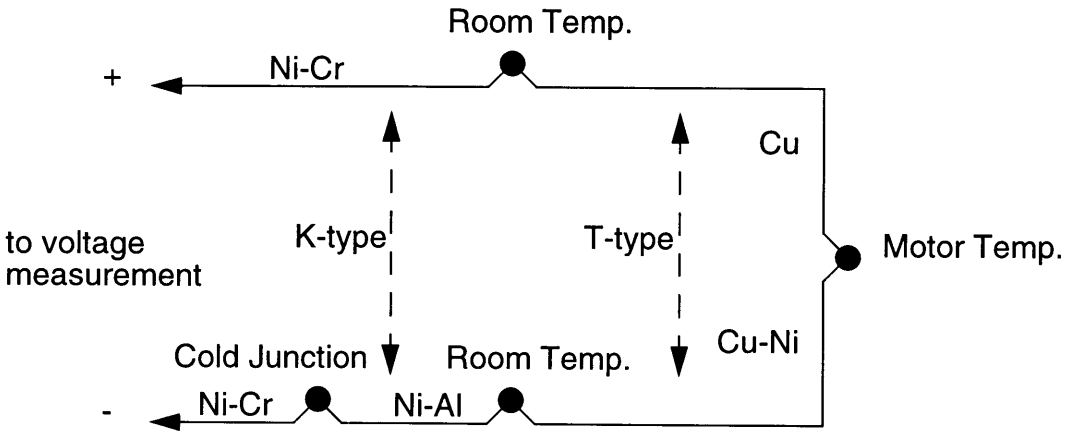

Figure I-1: Thermocouple Setup.

Appendix J

Sample Code

Appendix J gives a complete listing of some sample codings. This appendix may be useful for understanding Section 4.3.2 and the other appendices. The intent in giving these listings is to give a larger view of the coding that may not be evident in the code fragments discussed in the previous Appendices. The first listing is a listing of the code for the PC-AT compatible machine. Following this is a listing of the header *serial.h* which is used by the PC code. The third listing is the code for the DSP, followed by the memory map file *MAP.CMD* which is used during compilation.

To compile the PC code, execute the following batch file at the DOS prompt:

```
cl /c /AL /FPi87 /F f000 %1.c
LINK %1.obj lmcload.obj,%1.exe,,lm30dev.lib graphics.lib llibc7.lib,,
```

Typing the batch file name followed by the name of the C program (without the extension) will invoke the compiler and linker. The output will be a file whose name is formed from the C program name followed by ".exe".

To compile the DSP code, execute the following batch file:

```
cl30 -s -al %1.c -z -cr -m %1.map map.cmd c:\c30tools\boot.obj
c:\c30tools\rts.lib -o %1.out
```

Typing the batch file name followed by the name of the C program (without the extension) will invoke the compiler/linker. The output will be a file whose name is formed from the C program name followed by ".out".

J.1 PC Code

```
/*
PCBAL.C -     This is a PC C program that downloads and runs
                bal.out on the DSP32C board.
*/

#include "tms30.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include <math.h>
#include <errno.h>
#include <sys\timeb.h>
#include <sys\types.h>
#include <time.h>
#include "serial.h"
#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif TRUE

/* size of the buffer to use for string incoming characters */
#define COM_BUF_SIZE (1 * 1024)

#define  BOARDADR       0x290      /* Factory default I/O address.  */

#define  COMM0          0x30000    /* Start of .bss memory area.    */
#define  POSL           COMM0 + 0  /* Absolute memory locations     */
                                   /* reserved in LSICMAP.CMD.      */
#define  POSH           COMM0 + 1

#define  IASMLOC        COMM0 + 2
#define  IBSMLOC        COMM0 + 3
#define  IARMLOC        COMM0 + 4
#define  IBRMLOC        COMM0 + 5

#define  IASDLOC        COMM0 + 6
#define  IBSDLOC        COMM0 + 7
#define  IARDLOC        COMM0 + 8
#define  IBRDLOC        COMM0 + 9
#define  IASCLOC        0x3000a
#define  IBSCLOC        0x3000b
#define  IARCLOC        0x3000c
```

```
#define  IBRCLOC        0x3000d

#define  IMAGLOC        0x3000e
#define  GONOGOLOC      0x3000f

#define SAMPLES 1024.0

static char *buffer_start;      /* beginning of the buffer */
static char *buffer_end;        /* end of the buffer */

/* pointer to next place to put character in */
static char *buffer_in;
static char *buffer_out;        /* place to get next character from */

static int count = 0;           /* number of characters in buffer */
volatile int EOT = 0;

static void (_interrupt _far *old_serial_interrupt)();
static void (_interrupt _far *old_break_interrupt)();

void init_buf(void);
void init(void);
void empty_buffer(void);
void write_port(char *s);

/*******************************************************
 * serial_interrupt -- interrupt handler for serial     *
 *      input                                           *
 *                                                      *
 * Called in interrupt mode by the hardware when        *
 * a character is received on the serial input          *
 *******************************************************/
static void _interrupt _far new_serial_interrupt()
{
        int     int_status;     /* status during interrupt */

        _disable();

        int_status = inp((int)&COM->status);

        /* tell device we have read interrupt */
        (void)inp((int)&COM->interrupt_enable);
        (void)inp((int)&COM->interrupt_id);

        if ((int_status & S_RxRDY) == 0) {
```

```
                _enable();
                return;
        }
        *buffer_in = inp((int)&COM->data) & 0x7F;
        if ((*buffer_in)==0x04) {
                EOT=1;
                ((*buffer_in)='\0');
        }
        if ((*buffer_in)==0x0A) ((*buffer_in)='\0');
        if ((*buffer_in)==0x0D) ((*buffer_in)='\0');
        buffer_in++;

        if (buffer_in == buffer_end)
                buffer_in = buffer_start;

        count++;
        outp(0x20, 0x20);
        _enable();
}

static void _interrupt _far new_break_interrupt()
{
        _disable();
        _dos_setvect(0x1b, old_break_interrupt);
        _dos_setvect(0x0b, old_serial_interrupt);
        outp(0x21, inp(0x21) | 0x08 );
        outp(0x20, 0x20);
        _enable();

        (*old_break_interrupt)();
}

void main(int argc, char *argv[])
{
unsigned short loadstat;
unsigned long posith,positl;
float curs[12];
int status;
void empty_buffer(void);        /* dump the data buffer */
float torque=0.0;
char keyin;
FILE *data;
float imag;
char *filename;
unsigned long lastpos,lookf;
```

```
imag=0;
printf("Magnitude of current? ");
scanf("%f",&imag);

if (argc==1) filename="DUMP.DAT";
else filename=*++argv;
if ((data = fopen(filename,"w"))==NULL) {
        printf("Can't open data file %s\n",filename);
        return;
}


EOT=0;

init_buf();

old_serial_interrupt=_dos_getvect(0x0b);
old_break_interrupt=_dos_getvect(0x1b);

_disable();
_dos_setvect(0x0b, new_serial_interrupt);
_dos_setvect(0x1b, new_break_interrupt);
_enable();

/* enable interrupts */
outp(0x21, inp(0x21) & 0xF7);
outp(0x20, 0x20);

init();

_clearscreen(_GCLEARSCREEN);

/* Initialize board:   */

SelectBoard(BOARDADR);
loadstat = coffLoad("bal.out");  /* Special load function; required */
                                 /* with -cr (RAM) linker option.   */

if (loadstat != 0)
  {
  printf("\n\nError During Program Load!!!!\n");
  printf("coffLoad() returned %x\n\n", loadstat);
  exit (0);
  }
```

```
Reset();        /* Start the DSP program running.    */

WrBlkFlt(IMAGLOC,DUAL,1,&imag);

Put32Bit(GONOGOLOC,DUAL,1);

write_port("RUNN ");
while (EOT==0);
EOT=0;
buffer_out=buffer_start;
buffer_in=buffer_start;

lookf=0;
positl=Get32Bit(POSL,DUAL)>>16;

while(1) {
    _settextposition(1,1);
    lastpos=positl;
    positl=Get32Bit(POSL, DUAL)>>16;
    printf("Position: %lu     \nNext Bin: %lu     \n",positl,lookf);
    printf("Degrees: %f\n",360.0*((int) positl)/65536.0);

/*  if (((long) positl)>((long) lookf)+16)) printf("%c",7);*/

    if ((((long) lastpos)<((long) positl)) &&
      (((long) positl)>((long) lookf)-16)) &&
      (((long) positl)<=((long) lookf)+16))) {
            lookf+=32;
            write_port("SCAN 1,1");

            while (EOT==0);
            EOT=0;
/*          if (strncmp(buffer_out,"ER",2)==0) printf(buffer_out);
            else { */
                    torque=(float) strtod(&buffer_out[3],NULL);
                    RdBlkFlt(IASMLOC,DUAL,12,curs);

                    printf("     %10s   %10s   %10s\n",
                           "Command","Drive","Measured");

                    printf("IAS: %+10.6f   %+10.6f
                                %%+10.6f\n",
                          curs[8],curs[4],curs[0]);
                    printf("IBS: %+10.6f   %+10.6f   %+10.6f\n",
```

```
                        curs[9],curs[5],curs[1]);
                printf("IAR: %+10.6f   %+10.6f   %+10.6f\n",
                        curs[10],curs[6],curs[2]);
                printf("IBR: %+10.6f   %+10.6f   %+10.6f\n",
                        curs[11],curs[7],curs[3]);

                printf("Torque: %+7f       \n",torque);
                fprintf(data,"%u ",positl);
                fprintf(data,"%+10.6f %+10.6f %+10.6f %+10.6f ",
                curs[8],curs[9],curs[10],curs[11]);
                fprintf(data,"%+10.6f %+10.6f %+10.6f %+10.6f ",
                     curs[4],curs[5],curs[6],curs[7]);
                fprintf(data,"%+10.6f %+10.6f %+10.6f %+10.6f ",
                     curs[0],curs[1],curs[2],curs[3]);
                fprintf(data, "%+7f\n",torque);
/*          }*/
            buffer_in=buffer_start;
            buffer_out=buffer_start;
    }

    if (kbhit()!=0) {
           if ((keyin=getch()) == 'q') {
                   fclose(data);
                   _disable();
                   _dos_setvect(0x1b, old_break_interrupt);
                   _dos_setvect(0x0b, old_serial_interrupt);
                   outp(0x21, inp(0x21) | 0x08 );
                   outp(0x20, 0x20);
                   _enable();
                   return;
            }
    }

}  /*  End of while     */

}  /*  End of main()    */


/*********************************************************
 * init_buf -- initialize the buffer pointers           *
 *********************************************************/
void init_buf(void)
{
        buffer_start = malloc(COM_BUF_SIZE);
        buffer_in = buffer_start;
```

```
        buffer_out = buffer_start;
        buffer_end = buffer_start + COM_BUF_SIZE - 10;
}

/*******************************************************
 * init -- initialize the port                         *
 *******************************************************/
void init(void)
{
        /* don't allow interrupts while we do this */
        _disable();
        /* receive interrupts */
        outp((int)&COM->interrupt_enable,   I_CHAR_IN);

        outp((int)&COM->format,
         F_BAUD_LATCH|F_NO_BREAK|F_PARITY_NONE|F_STOP1|F_DATA8);

        /* now that we have the baud latch set, send baud */
        outp((int)&COM->baud_l, SPEED & 0xFF);
        outp((int)&COM->baud_h, SPEED >> 8);

        outp((int)&COM->format,
         F_NORMAL|F_NO_BREAK|F_PARITY_NONE|F_STOP1|F_DATA8);

        outp((int)&COM->out_control, O_OUT1|O_OUT2|O_RTS|O_DTR);

        /* read the input registers to clear */
        /* their i-have-data flags           */
        (void)inp((int)&COM->data);
        (void)inp((int)&COM->interrupt_enable);
        (void)inp((int)&COM->interrupt_id);
        (void)inp((int)&COM->status);

        outp(0x20, 0x20);                   /* clear interrupts */
        _enable();
}

/*******************************************************
 * Empty_Buffer -- dump all the data buffered by       *
 *              the interrupt routine                  *
 *******************************************************/
void empty_buffer(void)
{
        while (count > 0) {
                fputc((*buffer_out)&0x7F, stdout);
```

```
                buffer_out++;
                if (buffer_out == buffer_end)
                        buffer_out = buffer_start;
                _disable();
                count--;
                _enable();
        }
}

/*********************************************************
 * Write_port -- write a string to the RS232             *
 *********************************************************/
void write_port(char *s)
{
        while(*s!='\0') {
                while (!((inp((int)&COM->status)) & S_TBE));
                outp((int)&COM->data,*s);
                s++;
        }
        while(!((inp((int)&COM->status)) & S_TBE));
        outp((int)&COM->data,0x0A);
}
```

J.2 *serial.h*

```
/*******************************************************
 * serial.h -- define the structures and bits for the   *
 *      serial i/o hardware                             *
 *******************************************************/
/*
 * define the register structure for the serial i/o
 */
struct sio {
    char    data;           /* data register */
    char    interrupt_enable;/* interrupt enable register */
    char    interrupt_id;   /* what kind of interrupt is going on */
    char    format;         /* communications format */
    char    out_control;    /* modem control lines */
    char    status;         /* status byte */
    char    i_status;       /* input status */
    char    scratch;        /* extra pad */
};
#define baud_l data             /* alias for sending baud rate */
#define baud_h interrupt_enable /* alias part 2 */

/*
 * Defines for Interrupt Enable Register (interrupt_enable)
 */
#define I_STATUS        (1 << 3) /* interrupt on modem status changed */
#define I_REC_STATUS    (1 << 2) /* interrupt on rec. status changed */
#define I_TRANS_EMPTY   (1 << 1) /* interrupt on trans. empty */
#define I_CHAR_IN       (1 << 0) /* interrupt on character input */

/*
 * Defines for Line control register  (format)
 */
#define F_BAUD_LATCH    (1 << 7) /* enable baud rate registers */
#define F_NORMAL        (0 << 7) /* normal registers enabled */

#define F_BREAK         (1 << 6) /* set a break condition */
#define F_NO_BREAK      (0 << 6) /* no break condition */

#define F_PARITY_NONE   (0 << 3) /* no parity on output */
#define F_PARITY_ODD    (1 << 3) /* odd parity on output */
#define F_PARITY_EVEN   (3 << 3) /* even parity on output*/
#define F_PARITY_MARK   (5 << 3) /* parity bit is always 1 */
#define F_PARITY_SPACE  (7 << 3) /* parity bit is always 0 */
```

```
#define F_STOP1         (0 << 2) /* Use one stop bit */
#define F_STOP2         (1 << 2) /* Use two stop bits */

#define F_DATA5         (0)      /* 5 data bits on output */
#define F_DATA6         (1)      /* 6 data bits on output */
#define F_DATA7         (2)      /* 7 data bits on output */
#define F_DATA8         (3)      /* 8 data bits on output */

/*
 * Defines for the MODEM control register (out_control)
 */
#define O_LOOP          (1<<4)   /* loopback test */
#define O_OUT1          (1<<3)   /* Extra signal #1 */
#define O_OUT2          (1<<2)   /* Extra signal #2 */
#define O_RTS           (1<<1)   /* Request to send */
#define O_DTR           (1<<0)   /* Data terminal ready */

/*
 * Line Status register (Status)
 */
#define S_TXE           (1 << 6)
#define S_TBE           (1 << 5) /* Transmitter buffer empty */
#define S_BREAK         (1 << 4) /* Break detected on input */
#define S_FR_ERROR      (1 << 3) /* Framing error on input */
#define S_PARITY_ERROR  (1 << 2) /* Input parity error */
#define S_OVERRUN       (1 << 1) /* Input overrun */
#define S_RxRDY         (1 << 0) /* Receiver has character ready */
/*
 * Modem Status Register (i_status)
 */
#define I_DCD           (1 << 7) /* DCD control line is on */
#define I_RI            (1 << 6) /* RI control line is on */
#define I_DSR           (1 << 5) /* DSR control line is on */
#define I_CTS           (1 << 4) /* CTS control line is on */
#define I_DEL_DCD       (1 << 3) /* DCD line changed */
#define I_DEL_RI        (1 << 2) /* RI line changed */
#define I_DEL_DSR       (1 << 1) /* DSR line changed */
#define I_DEL_CTS       (1 << 0) /* CTS line changed */


/*
 * constants are used to define the
 * baud rate for the serial i/o chip
 * (Selected entries from Table-III of the National 8250
 *  data sheet)
```

```
 */
#define B1200   96
#define B2400   48
#define B9600   12

/*
 * The location of the i/o registers on the IBM PC
 */
#define COM1    ((struct sio near *)0x3f8)
#define COM2    ((struct sio near *)0x2f8)

/*
 * Use COM1 for this program
 */
#define COM      COM1
#define SPEED    B9600
```

J.3 DSP Code

```
#include <stdlib.h>
#include <math.h>


/* Program bal.c
generate balanced three-phase currents in both the stator and
in the rotor.  This program allows the stator current to have a phase
angle relative to its A phase, and it allows the rotor to have a
phase angle between its phase A and the stator's phase A.
Furthermore, this program rotates the currents in the phases while
maintaining a constant angle between stator phase A and rotor phase
A.  */

#define TRUE 1
#define IASREF  ((unsigned int *) 0x804000)
#define IBSREF  ((unsigned int *) 0x804001)
#define TIMECTL ((unsigned int *) 0x808030)
#define PERIOD  ((unsigned int *) 0x808038)
#define SOFTCON ((unsigned int *) 0x804008)

#define CANON0  ((unsigned int *) 0x800004)
#define CANON1  ((unsigned int *) 0x800005)
#define CANON2  ((unsigned int *) 0x800006)
#define CANON3  ((unsigned int *) 0x800007)
#define IOCREG ((unsigned int *) 0x800008)
#define IOSTAT  ((unsigned int *) 0x800008)
#define IOTCTL ((unsigned int *) 0x800009)
#define IARREF ((unsigned int *) 0x80000A)
#define IBRREF ((unsigned int *) 0x80000B)
#define IMEAS ((unsigned int *) 0x80000A)

#define pi 3.14159265

#define RSTCTRL 0x000601
#define SETCTRL 0x0006c1
#define COUNT 2500

#define IOCINIT 0x000000
#define IOCAL 0x200000
#define IOSTRT  0x400000
#define CALMSK 0x200000
#define IASMEAS 0x000000
#define IBSMEAS 0x010000
```

```
#define IARMEAS 0x020000
#define IBRMEAS 0x430000
#define ADMSK 0x800000

main()
{

double sqrt(),cos(),sin();
double w,phi,phir,n;
double posr;
extern int gonogo;
extern float iasm,ibsm,iarm,ibrm;
extern unsigned int posl,posh;
extern float iasc,ibsc,iarc,ibrc,imag;
float iarcref,ibrcref;
extern float iasd,ibsd,iard,ibrd;
int i;
unsigned int posl16;

gonogo=0;
while(gonogo==0);

iasm=0.0;
ibsm=0.0;
iarm=0.0;
ibrm=0.0;

iarcref=fabs(imag)*cos(-pi/2);
ibrcref=fabs(imag)*cos(-pi/2+2.0943951);

posl=*CANON0;
posl16=posl>>16;
posr=2*pi*posl16/65536.0;
phir=-9*posr+3.1042;
iasc=imag*cos(phir);
ibsc=imag*cos(phir+2.0943951);
iarc=iarcref;
ibrc=ibrcref;

iasd=iasc;
ibsd=ibsc;
iard=iarc;
ibrd=ibrc;

*IOCREG=IOCINIT;
```

```
*IOCREG=IOCAL;
*IOCREG=IOCINIT;
while(!(*IOSTAT & CALMSK));
*IOCREG=IOSTRT;

*TIMECTL=RSTCTRL;
*PERIOD=COUNT;
*TIMECTL=SETCTRL;

*IOCREG=IASMEAS;

asm(" OR 2h,IE");
asm(" OR 2000h,ST");

while(TRUE) {
posl=*CANON0;
posh=*CANON0; /* Dummy variable; Don't leave posh unassigned */
/* or the compiler will mess up the placement of */
/* variables in specific memory locations. At least */
/* this appears to be happening.  */

posl16=posl>>16;
posr=2*pi*posl16/65536.0;
phir=-9*posr+3.1042;
iasc=imag*cos(phir);
ibsc=imag*cos(phir+2.0943951);
iarc=iarcref;
ibrc=ibrcref;
}

}

void c_int02(void) {

extern float iasm,ibsm,iarm,ibrm;
int iasnew,ibsnew,iarnew,ibrnew;
extern float iasc,ibsc,iarc,ibrc;
extern float iasd,ibsd,iard,ibrd;

*IOTCTL=0;    /* write used to generate software trig */

while(!(*IOSTAT & ADMSK));
iasnew=((int) (*IMEAS))>>20;
*IOCREG=IBSMEAS;
while(!(*IOSTAT & ADMSK));
```

```
ibsnew=((int) (*IMEAS))>>20;
*IOCREG=IARMEAS;
while(!(*IOSTAT & ADMSK));
iarnew=((int) (*IMEAS))>>20;
*IOCREG=IBRMEAS;
while(!(*IOSTAT & ADMSK));
ibrnew=((int) (*IMEAS))>>20;

iasm=17.5*iasnew/2047.0;
ibsm=17.5*ibsnew/2047.0;
iarm=17.5*iarnew/2047.0;
ibrm=17.5*ibrnew/2047.0;

iasd+=0.01*(iasc-iasm);
ibsd+=0.01*(ibsc-ibsm);
iard+=0.01*(iarc-iarm);
ibrd+=0.01*(ibrc-ibrm);

if (iasd>15.5) iasd=15.5;
if (iasd<-15.5) iasd=-15.5;
if (ibsd>15.5) ibsd=15.5;
if (ibsd<-15.5) ibsd=-15.5;
if (iard>15.5) iard=15.5;
if (iard<-15.5) iard=-15.5;
if (ibrd>15.5) ibrd=15.5;
if (ibrd<-15.5) ibrd=-15.5;

*IASREF=((unsigned int)(-32767.0*iasd/25.5))<<16;
*IBSREF=((unsigned int)(-32767.0*ibsd/25.5))<<16;
*IARREF=((unsigned int)(-32767.0*iard/25.5))<<16;
*IBRREF=((unsigned int)(-32767.0*ibrd/25.5))<<16;

*IOCREG=IASMEAS;

}
```

J.4 *MAP.CMD*

```
/*************************************************************/
/* MEMORY MAP: MAP.CMD                                       */
/*                                                           */
/* This is a memory map for LSI TMS320C30 System Board, for  */
/* use with teh C Compiler (When not using SPOX).            */
/*************************************************************/

MEMORY    /* This maps memory SECTIONS to the board hardware.   */
{

    /* EXTERNAL SRAM ON THE MAIN BOARD:                       */
    /* Location 0 to C0h are reserved for interrupt vectors   */
    /* and Debug Monitor usage.  Although you COULD start using */
    /* memory at C1h, this map starts at 100h -- to allow for */
    /* future Monitor expansion, and for ease of adding hex   */
    /* address offsets.                                       */

    VECTS: origin=000000h  length=00000ch /* Interrupt vectors.  */
    BANK0: origin=000100h  length=00ff00h /* Std SRAM (0-wait).  */
    BANK1: origin=010000h  length=010000h /* SRAM upgrade option.*/
    BANK2: origin=020000h  length=010000h /* SRAM upgrade option.*/
    BANK3: origin=030000h  length=00f400h /* Std dual-access     */
                                          /* (1-wait).           */

    /* Bank 3 is dual-access between the 'c30 and the PC.
    The length shown is for the default 64Kx4 devices, but
    16Kx4 can be used.  In both cases, the top c00h locations
    are reserved for Debug Monitor use.  If you will never use
    the debug monitor, your programs can use this area.
    */

    /* CACHED DRAM MEMORY EXPANSION ON THE DAUGHTER BOARD: */

    EXPAND: origin=400000h  length=400000h

    /*  ON-CHIP MEMORY: */

    BLOCK0: origin=809800h  length=0000400h
    BLOCK1: origin=809c00h  length=0000400h
}

SECTIONS
   /* Assings program sections to the MEMORY statement, above. */
```

```
   /* The .data section, below, is not used by the linker to   */
   /* link C compiler output files.  It is used by the linker  */
   /* when it is linking Assembler output files.  The section  */
   /* is included in this "map" file so that the same map can  */
   /* be used to link files produced by EITHER the Assembler   */
   /* or C Compiler (useful if you write some functions in     */
   /* assembly language and happen to use the .data section).  */
{
   .text:          {}      >BANK0
   .bss
   {
   _posl = .;   . +=1; /* Define global address lables that */
   _posh = .;   . +=1; /* can be used for communication     */
   _iasm = .;   . +=1; /* between the PC and DSP programs.  */
   _ibsm = .;   . +=1; /* These will each occupy one 32-bit */
   _iarm = .;   . +=1; /* word starting at zero offset from */
   _ibrm = .;   . +=1; /* the beginning of the .bss section.*/
   _iasd = .;   . +=1;
   _ibsd = .;   . +=1; /* If you need more locations, you   */
   _iard = .;   . +=1; /* could add more "Comm" locations   */
   _ibrd = .;   . +=1; /* or you could create a "hole" in   */
   _iasc = .;   . +=1; /* memory here that you address using*/
   _ibsc = .;   . +=1; /* absolute pointers (instead of     */
   _iarc = .;   . +=1; /* these labels).                    */
   _ibrc = .;   . +=1;
   _imag = .;   . +=1;
   _gonogo = .;   . +=1;
}  >BANK3

.data:          {}      >BANK3
.cinit:         {}      >BANK3
.stack:         {}      >BLOCK0

/* Forces Reset and Interrupt Vectors to absolute locations: */
/* Your C source code should initialize these locations      */
/* using "ASM" in-line assembly macros (except for location  */
/* 00, which is initialized in the LSIBOOT.SRC startup file. */

 .int00   00h: {}    /*  Reset (Power-on or otherwise).   */
 .int01   01h: {}    /* INT0    */
 .int02   02h: {}    /* INT1 (A/D & D/A end of convert).  */
 .int03   03h: {}    /* INT2    */
 .int04   04h: {}    /* INT3    */
 .int05   05h: {}    /* XINT0   */
 .int06   06h: {}    /* RINT0   */
```

```
  .int07   07h: {}    /* XINT1   */
  .int08   08h: {}    /* RINT1   */
  .int09   09h: {}    /* TINT0   */
  .int10   0ah: {}    /* TINT1   */
  .int11   0bh: {}    /* DINT    */
}
```

Appendix K

Practical Issues

There are a number of intricate practical issues involved in making this system work. Some of the problems have been solved, some have only been worked around, and some have been annoying but harmless enough to be ignored. The impact of these problems on this thesis has been discussed to some degree in Section 4.3.1 and 4.3.2. Future system design should take into account all the learnings related in this appendix, in order to yield a more reliable and better performing system. Commercial systems especially should always keep in mind the environment in which the system operates and some of the problems discussed below which crop up in these severe environments. The most straightforward way to discuss these issues is to break them down by subsystem: control, drive and measurement. Another key area to analyze is the motor itself.

K.1 Problems with the Control Subsystem

The control subsystem works well, but the issue of having enough disk space does come up. The nearly antique system in use for this thesis has only 30 megabytes of hard disk storage. This is insufficient for an experimentation environment where two compilers are needed, one for the PC control software and one for the DSP software, and where large amounts of data are stored. This causes two machines to be used, one for each compiler. This still leaves a meager amount of space for data and code

storage. However, most systems today are equipped with 213 megabyte hard disks or bigger, which should amply cover an experimental environment's needs.

K.2 Problems with the Drive Electronics

The drive electronics has one extremely severe defect and one mild problem. The fast switching times of the servo system, cause large voltage transients. The transients couple into the measurement subsystem and have varied effects. In the position encoder they cause random counting behavior, thereby totally corrupting the position measurement. This is unacceptable in a system whose commutation algorithm relies on the position measurement. In the dynamometer circuits, they coupled into the input signals creating inaccurate torque readings.

The switching transients become a problem because of a juxtaposition of causes. First, to obtain reasonable conversion efficiencies the servo system needs to be a Class D, switching, type amplifier with fast switching edge rates. Second, the servo system is not designed to minimize radiated emissions and the measurement system is not designed to be immune to radiated emissions. In this situation it is almost inevitable that switching noise be picked up by the measurement apparatus.

There are a number of possible ways around this problem. First, the switching edge rates can be slowed down. This has the effect of reducing emissions but decreasing efficiency as well. Because of the large powers involved in this servo system, this is not a feasible option. The second solution is to minimize the conducted radiation. This has been attempted but does not help. If this approach to solving the problem is taken, it must be done up front with the problem in mind all the way through development. The third solution is to minimize the received radiation. This also has been attempted, but the system again has not been designed from the beginning with this in mind.

Another problem with the drive electronics is the accuracy of the currents produced. As the equipment used for this thesis was inherited, there was not any control over its selection. A more appropriate selection of electronics would yield a tighter

tolerance on the output currents. This would have the favorable effects of making experiments more repeatable and being able to control torque with greater precision.

K.3 Problems with the Measurement Subsystem

The measurement subsystem is negatively impacted by the electro-magnetic interference caused by the switching current source. Specifically, the original rotary encoder sporadically counted position and the dynamometer picked up erroneous signals. The best solution to these problems would be to fix the servo system. As workaround to this however a few things can be done. Analog rotary encoders should be used with trepidation in a switching environment and a more reliable choice is to use a digital encoder, as they a more immune to noise. To work around the erroneous dynamometer readings, the bandwidth can be reduced so as to average the fluctuation out. One should be aware though, that this has the undesirable effect of forcing the measurements to be taken more slowly.

Another major lesson learned during experimentation is to be careful about overheating electronics. In the current setup, the rotary encoder is physically mounted inside the stator. The stator can operate to temperatures of 150°C, whereas the electronics of the rotary encoder are only guaranteed to operate up to 50°C. This has caused the encoder to begin to operate erratically, probably as a result of thermal stressing and thermally accelerated failure. Thus, it is important to design the motor keeping in mind the possibility of electrical component failure.

An aspect of the present system configuration that could potentially cause problems is the asynchronous measuring procedure. Currently, the system requests torque data, then requests measurements for the rotor position and phase currents. There is no means for coordinating the measurements to happen at a given clock edge. This opens the possibility for phase errors in the measurements. Furthermore, there is no ability to take torque measurements at specific positions, but only within ranges of positions. This limitation arises as a result of the inability to trigger measurements in all measurement subsystems at specific points in time. Although there is no verified

problem, this is one aspect of system operation that merits future consideration.

K.4 Problems with the Motor

Two issues regarding motor design also deserve further consideration. First, with the present configuration of the motor, disassembly of the motor invalidates all previous torque versus position data. Because there is no means to preserve the relative position between rotor and stator and between the rotor and rotary encoder, any time the motor is disassembled, a completely new set of characterization data must be taken. Ideally, it would be possible to have an alignment scheme so that the motor can be assembled the same way every time. This would also make it easier to correlate torque fluctuations from motor to motor, possibly allowing for entire production runs to be characterized by the data collection on one unit.

The second problem that needs to be solved is that as the motor rotates, it spindles the power cables and cooling lines around the rotor. This causes position dependent torque variation. More specifically, the torque changes from rotation to rotation. Combined with the fact that the spindling is not neat, this phenomenon leads to variations in torque for which it is hard to compensate. It also creates the possibility that the motor will tear its own power or cooling lines.

Hopefully, these appendices present enough information for others to duplicate the current system. Furthermore, problems associated with the current setup have been delineated. Enhancements have been discussed which should advance the current setup, providing for a more feasible and reliable system architecture.

Bibliography

[1] Allen-Bradley, A Rockwell International Company, Milwaukee, WI. *Bulletin 1389 AC Servo Amplifier System*, publication 1389-5.1 edition, August 1987.

[2] Allen-Bradley, A Rockwell International Company. System Block Diagram 1389 4.5 and 9 Amp Axis Module. Non-Published Schematics, August 1987.

[3] Analog Devices, Inc., Norwood, MA. *Linear Products Databook*, 1990/1991 edition, 1990.

[4] Asada, Haruhiko and Youcef-Toumi, Kamal. *Direct-Drive Robots* Theory and Practice. The MIT Press, Cambridge, Massachusetts, 1987.

[5] Fitzgerald, A.E. and Kingsley, Jr., Charles and Umans, Stephen D. *Electric Machinery.* McGraw-Hill Series in Electrical Engineering. McGraw-Hill Publishing Company, New York, New York, fifth edition edition, 1990.

[6] Ilić-Spong, Marija, Miller, Timothy J.E., MacMinn, Stephen R., and Thorp, James S. Instantaneous Torque Control of Electric Motor Drives. *IEEE Transactions on Power Electronics*, PE-2(1):55–61, January 1987.

[7] Jackson, Deron K. Torque-Ripple Compensation for an Axial-Airgap Synchronous Motor. Master's thesis, Massachusetts Institute of Technology, August 1994.

[8] Kamiya, Shingo, Shigyo, Masakane, Makino, Tatsuo, and Matsui, Nobuyuki. DSP-Based High-Precision Torque Control of Permanent Magnet DD (Direct Drive) Motor. *Electrical Engineering in Japan*, 110(4):51–58, 1990.

[9] Kim, Kwang-Heon, Sim, Dong-Joon, and Won, Jong-Soo. Analysis of Skew Effects on Cogging Torque and BEMF for BLDCM. In *Conference Record of the 1991 IEEE Industry Applications Society Annual Meeting*, pages 191–197. IEEE, September 1991.

[10] Low, T.S., Tseng, K.J., Lee, T.H., Lim, K.W., and Lock, K.S. Strategy for the instantaneous torque control of permanent-magnet brushless DC drives. In *IEE Proceedings B (Electric Power Applications)*, volume 137, pages 355–363, November 1990.

[11] Matsui, Nobuyuki, Akao, Norihiko, and Wakino, Tomoo. High-Precision Torque Control of Reluctance Motors. *IEEE Transactions on Industry Applications*, 27(5):902–907, September/October 1991.

[12] Newman, Wyatt S. and Patel, Jay J. Experiments in Torque Control of the AdeptOne Robot. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 1867–1872, April 1991.

[13] Nogarede, B. and Lajoie-Mazenc, M. Torque Ripple Minimisation Methods in Sinusoidal Fed Synchronous Permanent Magnet Machines. In *Fifth International Conference on Electrical Machines and Drives*, number 341 in Conference Publication, pages 41–45, London, UK, September 1991. IEE, IEE.

[14] Oualline, Steve and Peter Norton Computing. *Advanced C Programming.* The Peter Norton Programming Series. Simon & Schuster, Inc., New York, New York, 1992.

[15] Putman, Byron W. *RS-232 Simplified, Everything You Need to Know About Connecting, Interfacing & Troubleshooting Peripheral Devices.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[16] S. Himmelstein and Company, Hoffman Estates, IL. *Operating Instructions, Model 66032 Power Instrument.*

[17] R. Sepe. *McGill Motor Torque Controller.* Technical report, Massachusetts Institute of Technology, Cambridge, MA, April 1991.

[18] R. Sepe. *McGill Motor Torque Controller Progress Report 2.* Technical report, Massachusetts Institute of Technology, Cambridge, MA, September 1991.

[19] SPECTRUM Signal Processing, Inc., Westborough, MA. *4 Channel Analog I/O Board User's Manual*, version 2.1 edition, November 1988.

[20] SPECTRUM Signal Processing, Inc., Westborough, MA. *DSP LINK Prototype Interface Module*, version 1.3 edition, November 1988.

[21] SPECTRUM Signal Processing, Inc., Westborough, MA. *TMS320C30 System Board Technical Reference Manual*, issue 1.0 edition, May 1990.

[22] SPECTRUM Signal Processing, Inc., Westborough, MA. *TMS320C30 System Board User's Manual*, issue 1.0 edition, May 1990.

[23] Texas Instruments. *TMS320C3x User's Guide*, a edition, April 1990.