A MECHANICAL-STATE OBSERVER FOR HIGH-SPEED VARIABLE-RELUCTANCE MOTOR DRIVES

by

EDWARD CARL FRANCIS LOVELACE

S.B., Mechanical Engineering, Massachusetts Institute of Technology, 1988

Submitted to the Departments of Electrical Engineering and Computer Science and of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degrees of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
and
MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June, 1996

Signature of Author_____
Department of Electrical Engineering & Computer Science

Certified by_____
Jeffrey H. Lang, Professor, Department of Electrical Engineering & Computer Science
Thesis Supervisor

Certified by_____
Kamal Youcef-Toumi, Associate Professor, Department of Mechanical Engineering
Departmental Reader

Accepted by_____
n A. Sonin,
Department of Mechanical Engineering,
uate Students

Accepted by_____
lorgenthaler,
uter Science,
Chairman, Departmental Committee on Graduate Students

A MECHANICAL-STATE OBSERVER FOR HIGH-SPEED VARIABLE-RELUCTANCE MOTOR DRIVES

by

EDWARD CARL FRANCIS LOVELACE

Submitted to the Department of Electrical Engineering and Computer Science and the Department of Mechanical Engineering on April 26th, 1996 in partial fulfillment of the requirements for the degrees of Master Of Science in Electrical Engineering and Computer Science and Master of Science in Mechanical Engineering.

Abstract

A mechanical-state observer is developed for the high-speed operation of a variable-reluctance motor (VRM). The observer is designed to provide rotor position and motor speed state feedback for the purposes of transient speed control and phase commutation. The plant output, providing the innovation terms for the observer, is the rotor position as estimated from phase currents which are measured a short delay after phase commutation.

A Kalman filter design is employed for the observer using a motor model which incorporates the dynamic behavior of interpolation steps between position estimates calculated from phase current measurements. A proportional-plus-integral controller is designed to modulate the commutation angles of the VRM for transient speed control. The observer drives the controller through the feedback of the estimated position and speed which are used as inputs to the controller. To support the observer and controller design, a VRM model is developed using a periodic nonlinear function representing the flux linkage relationship to phase currents. Computer simulations are developed around the VRM model, the speed controller, and the observer. The simulations are verified by laboratory experiments. The experimental drive system designed and constructed for the tests is based on a 2 hp 6-4 VRM with a design speed of 10 krpm.

Simulations predict damped stable response from the observer with a 30 msec typical settling time which is confirmed by the laboratory drive experiments. The typical steady-state rms state errors for the simulation are 4.84 rpm and 0.39 degrees. For the experimental drive the typical errors are 6.03 rpm and 1.03 degrees. Simulations are conducted up to 10 krpm and 2 hp, while experimental tests are limited to 6 krpm and less than 0.15 hp due to switching noise interference with the current measurements at conduction angles near 30 degrees.

Torque modeling errors are discussed in the context of the experimental performance; they have limited effect on the performance of the observer which is

dominated by the innovation terms. For this reason, a sophisticated nonlinear electrical model may not be necessary in applications with limited torque imbalance. Furthermore, decreased accuracy of the position inversion from current measurements results in unstable operation when sensed near the maximum misalignment position. The principal conclusion is that implementation of this observer for propulsion system drives is feasible with the design modifications proposed to maintain similar sensing accuracy for arbitrary commutation angles. Furthermore if the observer and controller are implemented in hardware rather than software, the theoretical speed limit for the observer-based control scheme is dominated by the decay time of the eddy currents, 6.9 μsec, resulting in a maximum VRM speed in excess of 100 krpm.

Thesis Supervisor: Jeffrey H. Lang
Title: Professor of Electrical Engineering

Acknowledgments

This thesis research has certainly been a long process, and its completion would not have been possible without the support of my wife, Susan, and my advisor, Professor Jeffrey Lang. They were always encouraging and supportive regardless of the obstacle. Through the long periods when I was working off-site for General Electric they never abandoned their enthusiasm. In particular, Susan has been both an significant inspiration and an always sympathetic confidant in the pursuit of my interests.

Additionally I would like to thank the companies that provided financial support for me during the years I conducted my research. The research project was principally funded by The General Electrical Company under the MIT P.O.# 201LS-L4Y05647. The General Electric Company, my active employer for six years, also funded my tuition well beyond the boundaries of their normal Advanced Course in Engineering limits. I appreciated the fact that they took into account my competing engineering duties and the time I spent off-site for the company. They have also been kind enough to allow me to continue my graduate research interests on a leave of absence. I would also like to thank SatCon Technology Corporation, and Bill Bonnice in particular, for providing me with an RA for my final year of thesis research.

Contents

List of Figures

List of Tables

List of Symbols

A	state transition matrix
a	matrix of function coefficients
a_i	function coefficient
α_{jk}	Marquardt curvature matrix component
B	drive system viscous damping
B	input distribution matrix
b_k	Marquardt gradient component
C	capacitance
C	drive system Coulomb damping
C	output distribution matrix
X	output distribution matrix
Δ	incremental operator
Δt_k	discrete time interval
Δt_{obs}	current measurement time delay
E.C.	electrical cycle
$\varepsilon\{\}$	covariance function
ε_{jk}	Marquardt scaled curvature matrix component
$f()$	general nonlinear function
F	continuous state transition matrix
Φ	discrete state transition matrix
ϕA	motor phase
G	continuous input distribution matrix
Γ	discrete input distribution matrix
Γ_1	discrete process noise distribution matrix
H	continuous and discrete output distribution matrix
I_P	phase current
I_S	sensed phase current
I	fixed phase current
I	identity matrix
i	phase current
i_{hyst}	current chopping hysteresis
i_{chop}	current chopping limit
J	continuous output distribution matrix from states
J	drive system inertia
K_{cond}	conduction angle control gain
K_I	integral speed error gain
K_{on}	turn on angle control gain
K_p	predictor estimator gain matrix
K_P	proportional speed error gain
k	discrete-time index
L	phase inductance
λ	flux linkage
Λ	Marquardt scaling factor
$\mathbf{L}_p$	predictor estimator gain matrix
M	eigenvector matrix solution from the Riccati equation
m	discrete-time index
M.C.	mechanical cycle
N	number of discrete indices per electrical cycle
n	discrete-time index
Θ	static rotor position/angle
θ	mechanical rotor position/angle
Θ_{cond}	nominal conduction angle
Θ_{EC}	angular period of an electrical cycle
Θ_{on}	nominal turn on angle
θ_{cond}	conduction angle
$\theta_{cond\,max}$	max conduction angle
θ_{on}	turn on angle
$\theta_{on,min}$	min turn angle corresponding to max average torque output
θ_0	conduction angle when $i = 0$

P	total number of machine phases
R	resistance
$\mathbf{R}_v$	covariance of measurement noise
$\mathbf{R}_w$	covariance of process noise
σ_i	standard deviation
$s_{1,2}$	continuous s-plane poles
τ_p, τ	electrical phase torque
T	time span for one electrical cycle
T_e	electrical machine torque
$\langle T_e \rangle$	average machine torque
T_L	machine load torque
t_s	controller update rate
u()	system input
v()	measurement noise
V	phase terminal voltage
V_b	DC supply voltage
V_-	op amp inverting input
V_+	op amp non-inverting input
Ω	motor speed reference setpoint
w()	process noise
ω	rotor speed
$\hat{\omega}$	motor speed error
$\hat{\omega}_{PI,max}$	PI speed error limit
$\hat{\omega}_{rate\,lim}$	PI speed slew rate limit
$\hat{\omega}_{PI}$	PI-compensated speed error
W_p'	phase coenergy
(x_i, y_i)	Marquardt data point
$\mathbf{x}$	plant state vector
$\bar{\mathbf{x}}$	observer state vector
$\hat{\mathbf{x}}$	perturbation state vector
$\tilde{\mathbf{x}}$	plant - observer error state vector
χ^2	Marquardt merit function
$y()$	system output
$y(a, x_i)$	Marquardt output function
Ψ	function of the average torque partials
$z_{1,2}$	discrete z-plane poles

Chapter 1

INTRODUCTION

1.1 Introduction

The purpose of this thesis is to develop a mechanical-state observer for a high-speed variable-reluctance motor, VRM, which uses sparse sampling of the phase currents. In particular, this is accomplished by measuring the phase currents a short time after turning the phase on, and then estimating rotor position from the measured currents and the inductive characteristics of the motor. This estimate is used to provide an innovation term for a dynamic model of the motor rotation. Through stability analysis appropriate gains are chosen to ensure that the estimator errors decay. The performance of this estimator is then explored through both simulation and hardware tests.

This thesis demonstrates that a position sensor can be eliminated from variable-reluctance motor drives even under high speed operation. Previous research [1, 2, 3] has shown that modeling and sensing systems can be developed to sufficiently model a motor and eliminate the position sensor. But high speed applications introduce two limiting conditions: low computation time, and multiply-excited phases. The challenge for this thesis is to design a robust, flexible estimator. The estimator should be applicable to generic VR machines for generator and motor usage. Also the estimator must be stable with respect to disturbances including load torque drifts and instantaneous perturbations, and during loss of a phase.

1.2 Motivation

The VRM is a strong candidate for use in electrical accessory drives for jet engine propulsion systems as both a motor or generator, or as a primary drive system for electric vehicles. In some cases, the VRM is specified for both tasks, such as a turbine-driven tank with packaging constraints or an aircraft engine with weight constraints that preclude using two separate machines. Its benefits are high-reliability and potentially low manufacturing cost. High-reliability exists because the motor can continue to operate with multiple failed windings. By increasing the total number of phases the torque production and generation disturbances resulting from winding failures can be reduced considerably. For a machine with at least five phases this constitutes a substantial reduction of the machine failure rate. Because the geometry is simple and the VRM is normally constructed with fewer windings than other machine topologies, this reduces material and manufacturing costs. Also there are no windings on the rotor to assemble. Since there are no permanent magnets this further reduces material and construction costs for VRMs as well as increasing operating temperature limits.

The major drawback of the VRM for vehicle propulsion and accessories is the requirement for a position sensor. Rotor position with respect to the salient stator poles is required for proper commutation of each phase, and therefore speed or torque control. This thesis provides a simple, robust case for eliminating the position sensor, and estimating position from the current measured in the stator windings. Since current sensors are already required for torque control and current limit protection, elimination of the position sensor greatly increases the operating reliability of the system.

1.3 Background

The background for this thesis comes from previous research in VRM modeling and position estimators. VRM position estimators have been designed and tested before in [1, 2]. However, these designs cannot be easily implemented for high rotor speed applications such as a gas turbine starter-generator system which operates at over 50,000 rpm. VRM modeling and control techniques similar to those applied in [3] are used in this thesis to develop an accurate electromagnetic model of the VRM, and to provide stable transient speed regulation.

Lumsdaine [1] developed a full-state observer for a VRM that utilized continuous sampling of all excited phases. The electrical equations were then integrated in a

dynamic model in order to estimate the motor states. The computation required for this system was prohibitive in a high-speed application. Harris [2] probed unexcited phases using a threshold detection scheme to determine when a specific position relative to rotor-stator alignment had been crossed. The unexcited phase was probed by pulsing on the inverter for a short interval. This produced a sawtooth current profile. The peak current was compared to a threshold current level which detected when the motor position had reached maximum misalignment for the probed phase. This method required no dynamic model integration. Since the currents from the probed phase were low, the winding resistance could be neglected in the calculations. The resulting calculation of rotor position was double-valued where the rotor position either lead or trailed maximum misalignment with the tested phase. The correct position estimate was resolved by making successive measurements until the threshold current near misalignment was reached. The additional commutation points created by probing the unexcited phases increased the controller complexity. Another limitation was that this design could only be operated with one phase being sampled, one being driven, and one coasting for a three phase VRM. The proposed estimator introduced in the following section addresses these issues of speed, complexity, and applicability to situations with multiply excited phases.

The dynamic models developed by Torrey [3] provide the necessary electromagnetic motor description to base a VRM simulation and an observer model upon. The model incorporates saturation effects and current chopping limitations resulting in accurate estimates of instantaneous torque. The mechanical motor dynamics are then determined using the estimated torque. Furthermore Torrey used linearized stability analysis around a constant motor speed to develop a proportional plus integral motor controller. This controller which was designed for optimal operating points is extended for speed-regulated operation at any steady-state operating point in this thesis.

1.4 Proposed VRM Drive Estimator

The VRM drive estimator proposed here to replace the position sensor in a variable-speed VRM is a mechanical-state observer based on phase current measurements which is applicable to high speed operation with multiply excited phases. The observer requires a motor model, a measurement algorithm for estimating position from phase current, and a structure within which the measurements are used to drive the motor model errors towards zero. The drive controller is designed to use the position and speed feedback from the mechanical-state observer for transient speed regulation. These components are introduced below.

A VRM simulation is developed to model the electromagnetic and mechanical dynamic behavior of the motor. The simulation describes the transient current and flux behavior in each commutating phase, and the instantaneous torque developed. Due to the saliency and magnetic saturation characteristics of the VRM, this model is both non-linear and dependent on the relative alignment of a rotor pole and the commutating phase. Position and speed dynamics are then modeled based on knowledge of the motor inertia, viscous damping, and other load torques. The control angle inputs to the motor model are driven by a transient speed controller. The controller updates the turn-on and conduction angles for each phase based on a speed error to provide accelerating torque. The controller requires estimated position and speed which are supplied by the observer.

The mechanical-state observer model is a duplicate of the motor simulation. The observer model senses current from each phase a short time after the phase is turned on. From the position-dependent inductance characteristics, the rotor position is determined as in [2]. The error between the current-calculated position estimate and the observer position state is used to smooth the model through an innovation term applied to each observer state. This drive the observer states towards the plant states. Thus, with a minimum of one position estimate per phase each electrical cycle, the observer is used to provide the position and speed information necessary for commutation and transient speed control. Stability analysis is conducted to chose appropriate gains for both the speed controller and the observer innovation terms. Simulations are then executed with initial rotor position and speed state errors to evaluate the disturbance rejection capabilities and asymptotic stability of the observer. Transient speed operation is used to evaluate the stability of the controller.

1.5 Experimental Test Drive

To fully evaluate the position estimator, an experimental system was developed. This system includes a VRM, a characterized load, the VRM drive electronics, and a DSP-based observer and controller. The experimental system allows for verification of the motor models and the response of the observer. Also demonstrated are the potential hurdles in implementing the high-speed observer drive. This includes susceptibility to disturbances, bandwidth, and regions of limited operation.

The VRM modeled in the simulation phase of this thesis is the same one used for the experimental system. In this way, the simulation and experiment results may be directly compared, and the effects of hardware implementation and modeling errors

highlighted. In particular, the experiments show the effects of load torque modeling errors, and conditions under which a complex torque model may not be necessary. The tradeoff for an observer torque model is mainly complexity and computation time versus commutation position accuracy. The complexity of the interpolating observer is also evaluated with respect to computation time, stability, and accuracy.

1.6 Thesis Organization

The thesis begins in Chapter 2 with a presentation of VRM applications and the research basis for the VRM model and observer. A physical description of a generalized VRM is given followed by an electromagnetic description of the energy conversion process. Control methods for variable speed VRM applications and the associated drive electronics are then presented. Finally developments in modeling the electromagnetic behavior of VRMs and estimating rotor position are discussed as a point of departure for this thesis.

A VRM model is developed in Chapter 3 which characterizes the dynamic state equations for the drive system. This model is based on the actual motor that is used for the experimental system. The chapter begins with a presentation of the experimental tests performed to obtain a motor map of the flux linkage-current-position relationship. Then parameters for a periodic analytic function describing the relationship are calculated from the experimental data. The parameterized data is then curve fit for smooth interpolation during simulation. The resulting model is then used to produce a modeled flux linkage-current-position map which is compared to the original test data. Using energy methods a model of torque production is developed. The modeled static torque is also compared to experimentally measured static torque data to provide estimates of the model errors. Then the mechanical dynamic equations are developed taking into account both friction-related and arbitrary load functions. Finally torque maps are calculated as a function of turn-on angle, conduction angle, and motor speed for the dynamic equilibrium case.

Chapter 4 outlines the development of the transient speed controller. The linear controller updates the turn-on and conduction angles based on speed error. Stability analysis accounting for the non-linear motor model is performed to determine appropriate controller gains. Limits on the controller operation due to the non-linear nature of the model are also discussed. Finally, issues regarding the discrete implementation of the controller are presented. The controller stability is evaluated in this chapter by incorporating it into the VRM motor simulation.

Chapter 5 completes the VRM simulation discussions with the development of the motion estimator. State equations for a Kalman filter are developed followed by a stability analysis of the observer. The discrete implementation is also considered. The position sensing methodology, which uses current measurements following a phase turn-on and a model transforming those currents to rotor position, is then presented. The trade-offs of computation complexity, accuracy, and disturbance rejection are then examined. A map of phase current following commutation as a function of rotor position is calculated. This is the sensing model that provides the position state information used to drive the observer innovations. The chapter concludes with an evaluation of the observer stability using the VRM simulation.

The experimental system is presented in Chapter 6. The architecture of the PC-based controller, interface electronics, VRM, and mechanical load are first discussed. Specific implementation issues in the electronics hardware are then considered. Next, the software architecture is described with a discussion of software issues concerning the DSP controller. These issues include noise rejection, bandwidth, resolution, and accuracy of the controller system. The experimental drive operation is also described. Then the performance under various operating modes is demonstrated including speed control, off-line observer operation, and observer control of commutation and estimated speed input to the variable speed controller. The comparison of experiments to simulations is then presented. The results of the closed loop simulations demonstrate the stable performance of the linear controller for variable speed operation. The observer model is also shown to both track the motor model and reject initial position and speed error disturbances.

Chapter 7 presents a summary and conclusions, and opportunities for further research. The estimator performance is summarized for both stability and steady-state error. Speed limitations due to computation time are discussed and opportunities to deal with these issue. The accuracy of the observer torque model, and its potential impact on the stability of the system is also discussed. Then the problems of disturbance rejection and stringent performance requirements are considered. Alternative schemes are then considered for further study.

The appendices contain further design and analysis details. Appendix A is the C code listing for the VRM simulation. The experimental drive DSP software code is in Appendix B.

Chapter 2

VARIABLE-RELUCTANCE MOTOR DRIVES

2.1 Introduction

This chapter introduces VRMs and the application of observers to VRM drives. The first half of this chapter introduces the motor, the electronics, and the control algorithms used in VRM drives. This introduction illustrates various design constraints and features leading to the development of VRM observers. The second half of this chapter focuses on the previous work upon which this thesis is based. The investigation of Torrey [3] concerning VRM modeling is presented first. This model forms the basis for the VRM simulations used in this thesis. Then, observer designs from Lumsdaine [1] and Harris [2] are discussed. The tradeoffs of each methodology are evaluated in terms of their advantages and disadvantages for control of high-speed VRM drives. Further review of VRMs can be found from T. J. E. Miller [17, 18, 19].

2.2 Variable-Reluctance Machines

Figure 2.1 shows a VRM designed with six stator and four rotor poles typically referred to as a 6-4 VRM. The significant physical characteristics of the common VRM are:

1. Double Saliency. Both the rotor and stator are constructed with poles which direct the primary flux paths.

2. No Magnets. Since there are no magnets the VRM can withstand wider temperature ranges, both high and low.

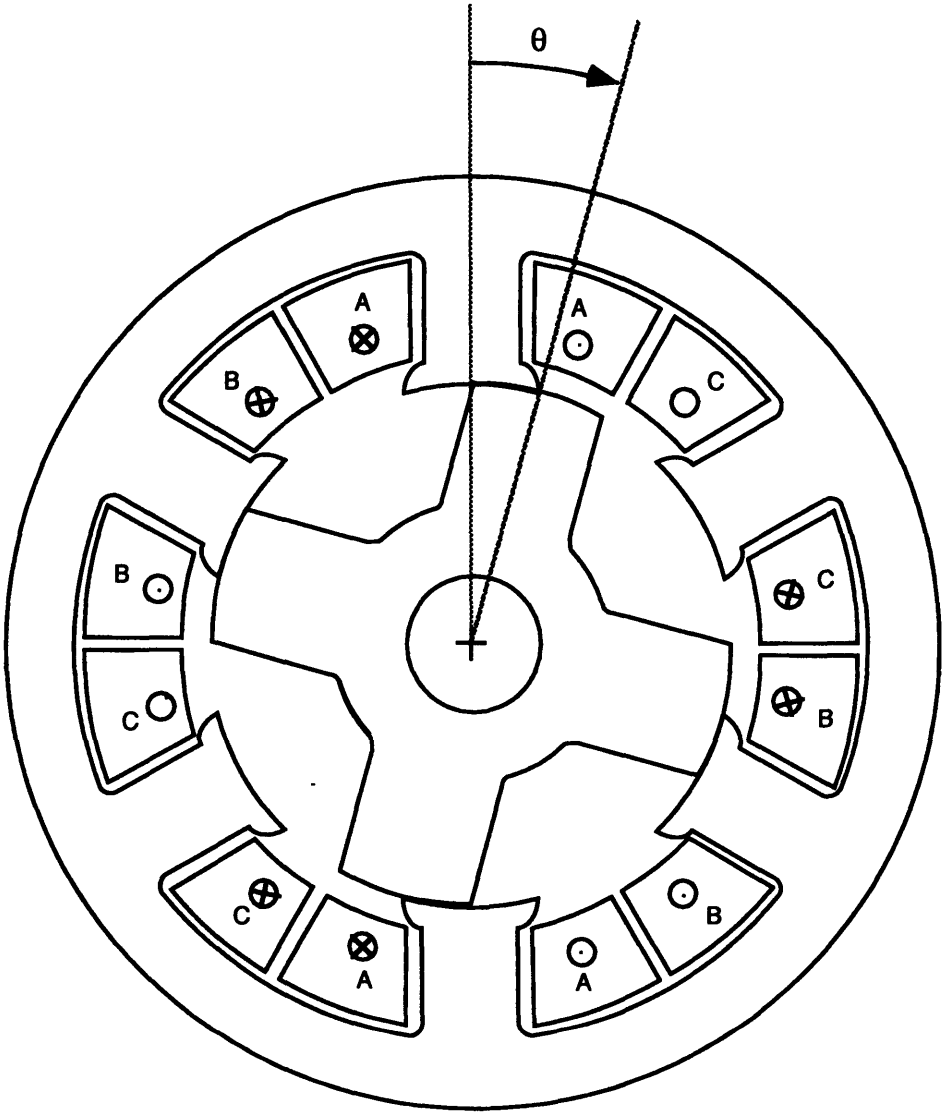

Figure 2.1: The 6-4 experimental variable-reluctance motor. Full scale schematic. Stack length is 40 millimeters. 160 turns per pole pair.

3. Large Slots. For drive system applications there are typically very few poles. This results in correspondingly large slots which eases the installation of the phase windings.

4. Magnetically Independent Phases. Magnetic independence means that the mutual inductance linking each phase is negligible. Therefore the flux linking each phase is due solely to the self-inductance, and then the electrical torque can be calculated from the flux-linkage for each conducting phase separately. This

has implications in development of the VRM model as is be discussed in subsequent sections.

5. High Specific Torque. VRMs with large numbers of poles can be designed with minimal back iron and hollow rotors. This results in a high torque to weight ratio.

6. Position Sensing. For designs with few pole faces a position sensor is normally necessary to detect the control angles when each phase should be electrically excited. This is because the excitation must be synchronous with the rotor poles passing each stator pole face. This issue is central to the purpose of this thesis.

These physical characteristics make it possible to design VRMs that are low in manufacturing cost, highly reliable, and applicable for high speed and hostile operating environments. With few pole faces they are appropriate for consideration in drive systems requiring torque and speed regulation. With higher numbers of pole faces VRMs can be considered in positioning system applications. VRMs are particularly suited to transportation systems which have relatively hostile physical environments. There are of course other motors with some of the same positive characteristics such as the simplicity of induction motors, and the positioning capability of stepper motors which must be considered in any design selection process.

The VRM shown in Figure 2.1 represents the experimental drive that is the focus of the modeling and physical experiments investigated in this thesis; its mechanical specifications are given in [10]. The motor is constructed from silicon steel laminations 0.35 mm thick, with a stack length of 40 mm, and an outer diameter of 123 mm. The windings were increased to 160 turns per pole pair to produce a 2 hp, 10 krpm machine, that satisfied the research requirements of investigating position observers with limited available sampling time under high power, and therefore, magnetically saturated conditions.

A VRM operates through the variation of the air gap geometry as a function of rotor position, θ, with respect to a fixed stator position. The angular position is taken with respect to a stator pole face thus becoming the mechanical angle of alignment. Opposite stator poles form a pair, and an electrical phase winding wrapped in series around each pair produces a phase inductance that varies periodically with position. The angular period of phase inductance for a 6-4 VRM is 90 mechanical degrees. The two additional symmetrically wound phases result in similar inductance variations that are displaced by 30 mechanical degrees each. By properly designing the motor geometry with sufficient back iron, the flux produced by any phase is independent of the excitation from either of the other phases. Thus mutual inductance between phases can be neglected in the

motor analysis. The self-inductance of each phase is maximized when the alignment angle is zero. Qualitatively this occurs when a rotor and stator pole face are lined up and the effective air gap is at its minimum reluctance. Figure 2.2 is a schematic representation of the phase inductances as a function of the mechanical rotor angle, θ.

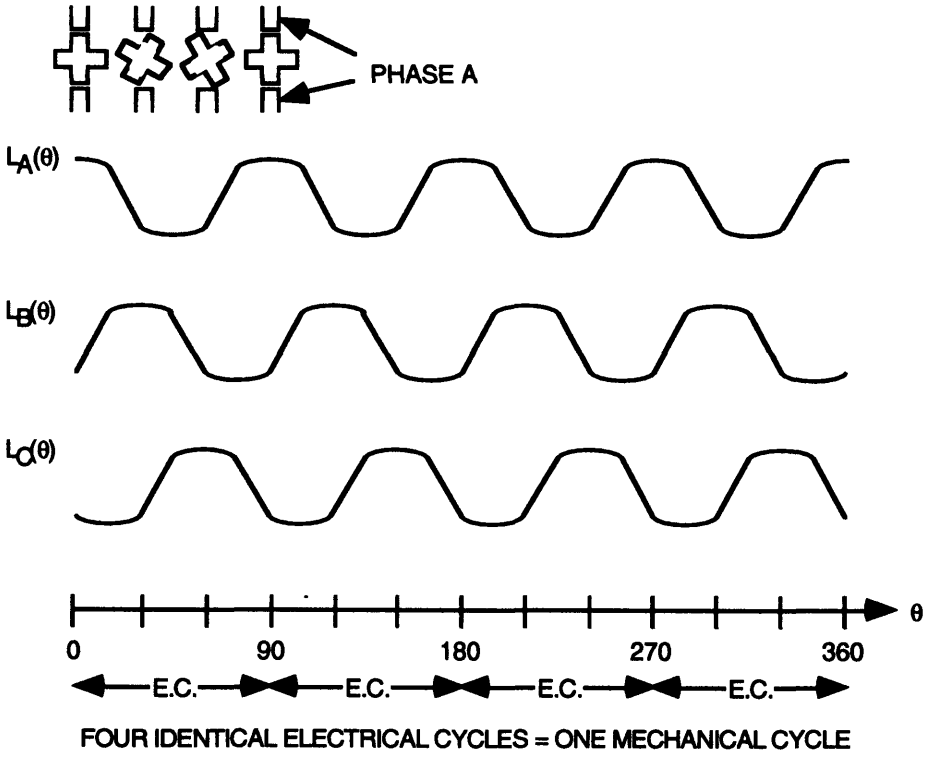

Figure 2.2: Phase inductance versus mechanical rotor position, θ, for a 6-4 VRM.

Driving current through a winding results in a torque which acts to align the nearest rotor pole faces to the excited stator phase. By controlling the current in each phase winding separately, each phase can be switched on when the nearest rotor pole is trailing the corresponding stator pole. Since the gradient of the phase inductance is positive when the rotor pole is trailing, the torque produced is positive. Thus the rotor is 'dragged' from stator pole to stator pole in one direction. Conversely negative torque is produced when current is driven through a winding after the rotor pole face passes the stator pole face. By changing the relative alignment angles at which each phase is turned on or off, the average motor torque is controlled, and depending on the mechanical loading, a relatively steady rotor motion can be established. The action of changing the path of current flow is called commutation, but commutation is also often used to describe the entire period of current flow from the turn on angle to when current

returns to zero after the turn off angle. The turn on and turn off angles are commonly referred to as the commutation angles or control angles.

2.3 Drive Electronics and Controller

The development of VRMs in drive applications has been coincident with improvements in the power electronics necessary to drive them. Primarily this is due to the high power, low loss electronic switching devices required for VRM drives. Though the topology requirements are similar to other AC and DC drives, the specific control sequences required differ thus off the shelf power electronics and controllers are not typically used. On the other hand the number of transistors required for a VRM drive is either equivalent or less, depending on the topology chosen, when compared to other AC drives with the same number of phases. Furthermore since there is no permanent excitation protection against generated open-circuit voltage is not required. Further discussion of the characteristics of VRM power electronics and controllers is found in [17, 18, 19].

As described in Section 2.2 above, the magnitude and direction of torque production depends on the phase commutation angles with respect to alignment. The task of the power electronics is to 'turn on' each phase by connecting the winding to a DC power source, and then 'turn off' or disconnect the phase at specific rotor positions corresponding to commands from the drive controller. During the turn on portion of the commutation, currents in the phase increase until they reach a steady-state level limited by the phase resistance, or the turn off angle is reached. During the turn off period currents are forced to decrease by reversing the voltage applied to the phase winding until the currents reach zero. This can be accomplished through various DC switching supply topologies known as inverters. Additionally current limiting can be incorporated to prevent over-temperature in the windings or provide another degree of torque control. Typically when the current limit is reached the phase is commutated and then turned back on again after a specified delay time or current hysteresis level is reached.

The two main inverters used with VRMs are shown in Figures 2.3 and 2.4 below. The first inverter is based on a motor with a primary and secondary (bifilar) winding for each phase. The advantage of this topology is potential low cost due to the use of only one controllable switch. The alternative topology is a monofilar phase inverter using two switches. The advantages for this electronics configuration include lower resistive losses with half the phase wiring of the bifilar system, lower cost switches due to low voltage blocking requirements, and the potential elimination of the snubbing

network required to keep the controllable switch in the bifilar system within safe operating limits. References [3,14] discuss the tradeoffs in power electronics selection in greater depth. For this thesis, the monofilar arrangement was chosen based on the winding space constraints of the experimental VRM, and to piggyback the development of the inverter on design effort already completed by Cameron [14].

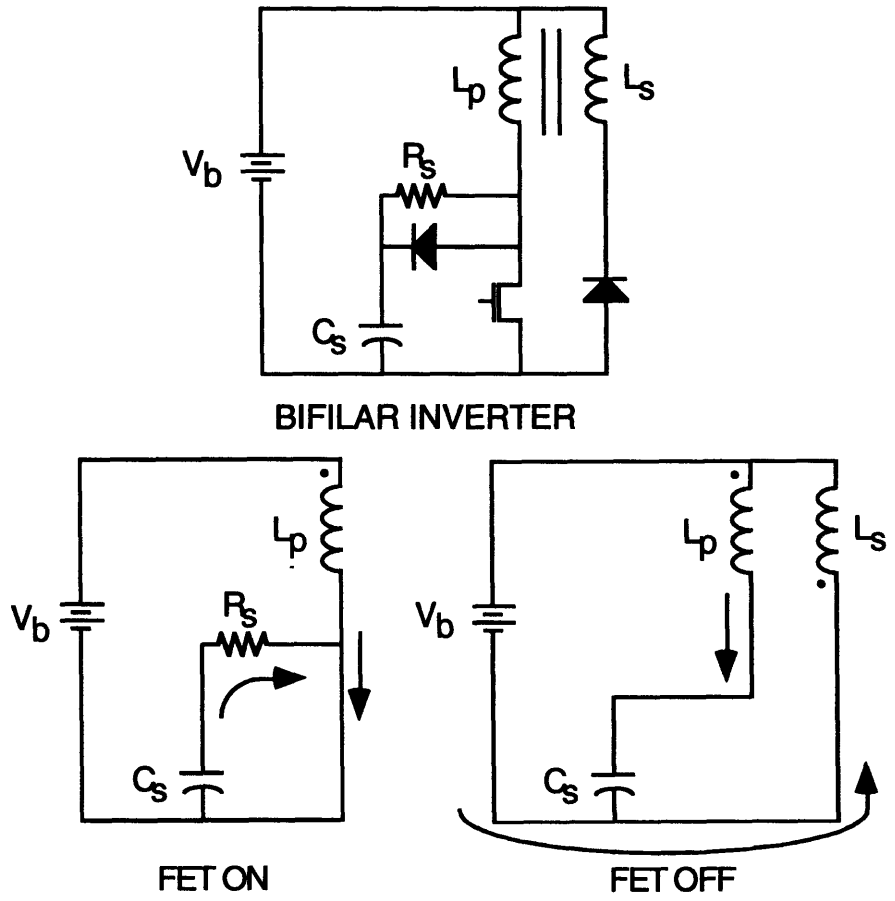

Figure 2.3: One phase circuit of a bifilar VRM inverter using one FET switch. The arrows indicate the current flow directions.

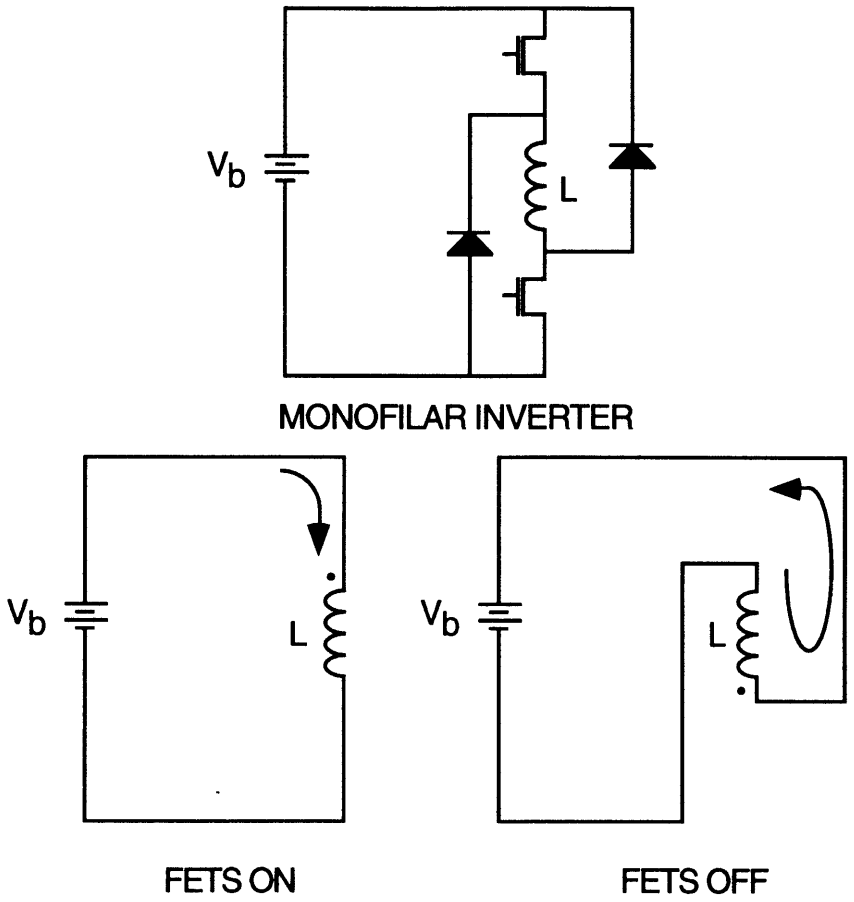

Figure 2.4: One phase circuit of a monofilar VRM inverter using two FET switches. The arrows indicate the current flow directions.

The function of the VRM drive controller is to specify the commutation angles through commands to the power electronics switches. The turn on and turn off angles are controlled with respect to a reference speed or torque for motors, or average voltage for generators. At slower speeds, the controller may also function to limit current as the steady-state resistive current drop is usually far in excess of the efficient and safe operating regime. A typical drive controller for a variable speed VRM is shown in Figure 2.5. The nominal turn on and or turn off angles are augmented in response to the error between the reference and actual motor speed. The motor speed is differentiated from a rotary position resolver. The resolver also serves to provide the position feedback so the controller can trigger when to turn on and off each phase given the desired commutation angles. This is the general control scheme that is used for the experimental VRM drive studied in this thesis.

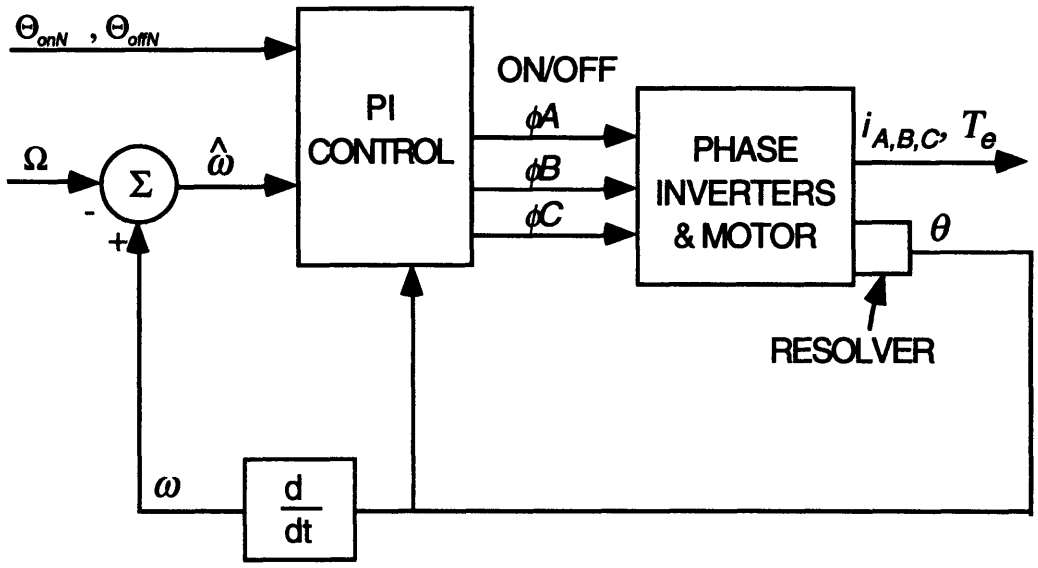

Figure 2.5: Generic variable speed control scheme for a VRM.

2.4 Electromagnetic Models

Much literature in motor design and analysis has been devoted to modeling the air gap interactions and material losses. The modeling process is complicated due to the highly nonlinear nature of electric motors. This is particularly true for salient pole machines including VRMs where in addition to the material and excitation nonlinearities, there are geometric nonlinearities. As described in Section 2.2, the geometric nonlinearities can be estimated and lumped into a value representing the self-inductance. The mutual inductance is often insignificant and ignored. Though the inductance is defined as the slope of the instantaneous flux linkage - current relationship, qualitative reasoning and experimental measurements show that the inductance is a nonlinear function of phase current as well as rotor position and other parameters such as temperature and air gap variations. For the purposes of this discussion, the temperature, air gap, and any other effects will be assumed constant.

Inductance models of varying complexity are commonly used in VRM analysis. The simplest model is one where the inductance is constant with respect to current at a given rotor position. This is usually coupled with a periodic, piece-wise linear model of the self-inductance versus rotor position. Saturation of the pole faces and back iron can be taken into account in a piece-wise linear model as employed by Vallese [5]. Such a model could be tuned to provide accurate predictions of cycle averages such as

converted energy and torque. The piece-wise linear model, though, results in significant RMS and peak current prediction errors, and would therefore be inadequate for simulating instantaneous rotor position for evaluation of the VRM observer.

Torrey [3] used the following analytic function to represent the nonlinear flux linkage relationship:

$$\lambda(i,\theta) = a_1(\theta)(1 - e^{a_2(\theta)i}) + a_3(\theta)i \tag{2.1}$$

where the coefficients a_1, a_2, and a_3 are periodic functions of θ. λ and i are the instantaneous flux linkage and current for one phase. a_3 represents the incremental inductance when the current is high while a_1 models the saturation flux linkage and a_2 provides smoothing during the onset of saturation as current is increased. Ilic-Spong et al. [8.9] originally presented the form of this equation with only a_1 being a function of θ. Implicit in this function are three periodic functions describing the coefficients. Experimentally measured flux linkage - current data at instantaneous values for θ, i, and λ were fit to Equation 2.1 resulting in a table of values for the coefficients as a function of θ. Torrey then successfully fit the coefficient table to a partial Fourier series representation thus allowing for smooth interpolation of the periodic functions in a drive simulation.

The nonlinear model is employed for the simulations and observer developed in this thesis. The experimental measurements and process of fitting the data for the candidate VRM of Figure 2.1 are described in Chapter 3.

2.5 Observers

Observers are a common control technique used to provide full-state feedback when the entire state vector is not available for measurement. Figure 2.6 shows a generic schematic of a full-state observer as applied to linear dynamic system. In this schematic the input vector, $\mathbf{u}$, is input to both the actual plant represented by $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, and a plant model shown as $\overline{\mathbf{A}}$, $\overline{\mathbf{B}}$, and $\overline{\mathbf{C}}$. The plant model is designed to approximate the dynamic behavior of the actual plant and is referred to in controls literature as an observer or estimator. The error between the plant and observer outputs, $\mathbf{y} - \overline{\mathbf{y}}$, is fed back into the observer input vector to provide an innovation to the model. By appropriately selecting the gains in the matrix, $\mathbf{E}$, the observer state vector, $\overline{\mathbf{x}}$, can be designed to track the plant state vector by driving the observed output vector errors towards zero. The observer state vector, $\overline{\mathbf{x}}$, can then be used in a control system to

provide feedback for the plant states, **x**, that are not present in the plant output vector, **y**.

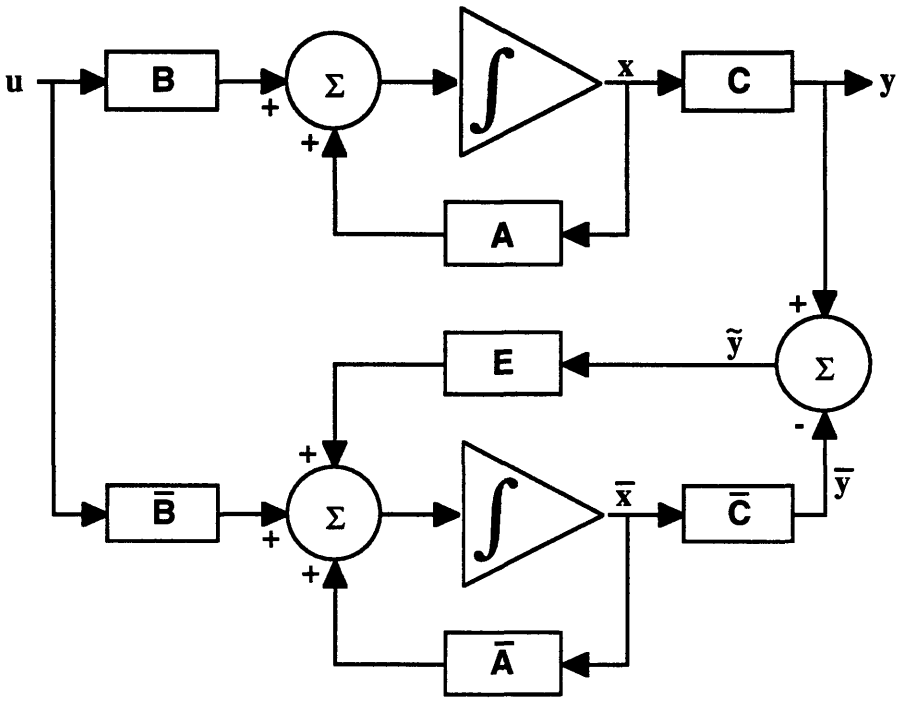

Figure 2.6: Generic full-state observer representation for a linear dynamic system.

There are many issues to consider when deciding to use an observer for control input purposes. Often times there is a tradeoff concern between the increased control computation complexity of an observer and providing the necessary sensors to reconstruct the full state vector. This assumes of course that the desired state information is controllable, but not observable, given the current output measurements; for further discussion of this issue see Luenberger [11]. Usually, some of the desired plant states can be reconstructed from the output vector. In this case, a reduced-order observer can be designed to estimate only the missing states. The combined vector of estimated states from the observer and observable states from the output vector are then fed back to the control system. One final note is that the dynamic behavior of both the full-state and reduced-order observers can be designed without affecting the pole locations of the feedback control system. This result is given by the Eigenvalue Separation Theorem [11] for linear systems. Exploiting this general result, observers are often designed with poles that are significantly faster (five to ten times) than those of the feedback control system. In this way, errors in the observer are driven to steady-state well before the effects of this state information can effect the performance of the control system.

Observers are a desirable feature in VRM control systems because their application can eliminate the need for rotor position sensing. As described in the previous sections, operation of the VRM is achieved by electrically exciting each phase in synchronism with a rotor pole passing the excited stator pole. This requires position sensing to determine the appropriate instants to turn on and turn off each phase. Previous studies of VRM observers have explored different design options of varying complexity, performance, and limitations. It is precisely these characteristics that must be compared to the expense, packaging constraints, and reliability associated with incorporating a position sensor into the VRM drive system. The research of Lumsdaine [1] and Harris [2] provide examples of two such approaches with different design characteristics that can be examined based on the tradeoffs just noted.

Lumsdaine developed a full-state, nonlinear observer based on continuous monitoring of the VRM phase currents. By examining all phases continuously, this observer predicts rotor position within one of the four identical electrical cycles (see Figure 2.2). The phase currents, defined as the output vector, drive the observer corrections at each time step of the simulation. Recalling that the flux-linkage and current are dependent quantities, only one can be selected as an independent state for the system. Thus choosing the phase flux linkages as the independent states necessitated designing a full-state observer. The observer error matrix gains were chosen based on the extended Kalman-Bucy design criteria for nonlinear systems, and the observer loop was analyzed for stability using Liapunov's methods. Note that the extended Kalman filter seeks to minimize the state errors with the assumption of the presence of Gaussian noise. Performance of the full-state, nonlinear observer is equivalent to a 14-bit resolver in off-line simulations. In order use the observer on-line for constant speed control of a VRM, code and model simplifications are required. This reduced the accuracy to the equivalent of a 12-bit resolver, and reduced the stability robustness of the observer with respect to noise, modeling errors, and speed perturbations. The chief advantage of Lumdaine's observer is its high accuracy at instantaneous rotor positions. This feature becomes significant in precision servo applications, or in drive applications requiring high efficiency and high specific torque.

Harris' observer design functions by probing the unexcited VRM phases with short voltage pulses. Given a commutation angle at which a phase is scheduled to be turned on, the unexcited phase is probed with brief pulses to estimate the instantaneous inductance. The inductance function is then inverted to calculate the position. This method has two position solutions within one electrical cycle. The problem can be solved by taking two position measurements thus establishing the gradient direction and eliminating one of the solutions. Harris' solution is to detect the minimum inductance point which occurs at only one position per electrical cycle. Any other rotor angles required for commutation are estimated with a simple observer without an innovation

term. The effects of eddy currents were also considered, and so the pulse duration and strength was designed to minimize this error source. This design has significantly less computational complexity than the full observer method of Lumsdaine. Additionally, with reasonable modeling accuracy and slow mechanical dynamics relative to the time traversing one electrical cycle, interpolation between minimum inductance measurements will follow rotor position quite well. One restriction of this method is that the phases cannot be excited simultaneously because the algorithm assumes that one phase is excited, another is coasting, and the third is being pulsed. This method also requires hardware control of the distinct excitation voltages used for pulse measurements and for commutating a phase. Harris experimentally verified the position detection technique by using it to drive a VRM speed controller with a fixed commutation angle. The observer was proposed in theory but never experimentally tested.

The features common to both designs is the use of phase current measurements to drive the observer to asymptotic stability. Both assume phase independence and knowledge of the phase terminal voltage. Lumsdaine directly uses the current feedback error to correct the observer while Harris calculates rotor position directly. The position state is reconstructed by sensing phase currents and inverting this information back to rotor position based on models of phase inductance as a function of position. The computational complexity of Lumsdaine's observer is prohibitive for high speed operation, and is potentially more costly to implement than simply installing a position sensor if a more sophisticated controller device is required. Harris' design, though simple to implement, restricts operation to non-overlapping conduction in the multiple phases.

The proposed observer shown in Figure 2.7 is similar to the Harris observer except that the measurements are made on an excited phase a fixed time after commutation begins. This increases the flexibility of the drive design allowing multiple phases to commutate simultaneously. The current measurements are inverted to estimate rotor position, and then used to drive the innovation terms for the observer model. Between commutation estimates the observer model interpolates the state transitions similar to Lumsdaine's design.

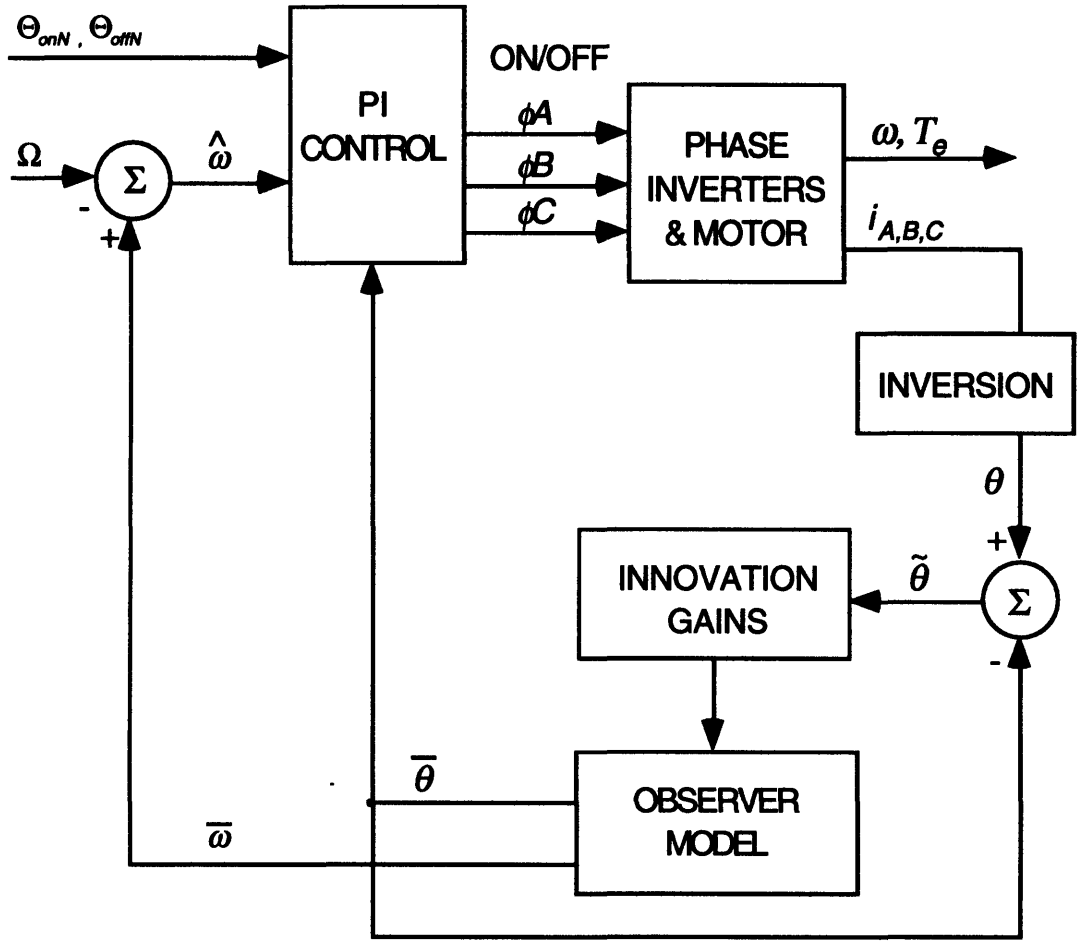

Figure 2.7: VRM control system schematic with proposed mechanical-state observer.

2.6 Summary

This chapter identifies the major physical and operational characteristics of VRMs. The machine is simple to construct from punched steel laminations for both the rotor and stator. Typically it has few windings (three or five) on the stator, and no electrical excitation on the rotor. Furthermore, the machine is capable operating as a motor or generator even with winding failures. These characteristics make VRMs reasonable choices for selection in combined drive and generator applications with cost and space constraints. Since operation requires knowledge of the rotor position, this constitutes the only major drawback of the VRM for such applications. Application of a position

observer will eliminate the need for a position sensor thus reducing the cost of the system and in many cases improving the reliability.

The work of Torrey in VRM modeling provides the basis for the software simulation structure and the experimental observer model. The nonlinear flux linkage model improved the accuracy of previous models with respect to instantaneous phase current, position, and torque calculations. As the subsequent chapters demonstrate, the nonlinear model allowed for accurate representation of observer errors even within an electrical cycle during simulation experiments.

The observers of Lumsdaine and Harris presented in this chapter provide the departure points for the observer presented in this thesis. The proposed observer combines and balances the improved accuracy and tracking of Lumsdaine's solution with the simplicity of Harris'. While Lumdaine's observer involved continuous sampling and innovation based on current measurements, this design uses a single estimate per phase electrical cycle. However unlike Harris' solution, this design does not operate by probing unexcited phases and therefore is more flexible for drive situations requiring multiply excited phases. These observer characteristics are combined with the accurate flux linkage model based on Torrey's investigations. The following chapters present the various components of the model, and their integration into the drive system based, in part, on this previous work.

Chapter 3

VRM MODEL

3.1 Introduction

This chapter presents the dynamic VRM model development used for the drive simulations and for the experimental system observer model. The model includes electrical and mechanical state dynamics to describe the evolution of phase current, torque, rotor position, and motor speed. The VRM model parameter values are based on the experimental VRM that is used to verify the simulation predictions.

Using the generalized description of VRM structure and operation outlined in Chapter 2 an appropriate dynamic model is constructed. The basis of the model is the relationship between the flux linkage, current, and rotor position ($\lambda - i - \theta$) for each phase. Coupling this characteristic to the electrical terminal relation fully describes the electromagnetic dynamic behavior. The torque model is then determined based on knowledge of the phase current and flux linkage. Finally, given the electrically produced torque, the mechanical dynamics is presented. To simplify the simulations while maintaining sufficient accuracy certain assumptions are made throughout the development. The phase windings are assumed to be independent. Also core losses are assumed to be dominated by the resistive losses.

As stated above, the torque production model requires knowledge of the $\lambda - i - \theta$ relationship for the phases. Using the assumption of magnetic independence introduced in the previous chapter, this becomes a single variable problem of calculating the position-dependent self-inductance of a phase. Static measurements of the experimental VRM provide the necessary data to specify this relationship. Exploiting the periodic, continuous nature of the winding inductance in a VRM, a nonlinear function is used to model this relationship as shown in Chapter 2. Energy methods are then employed to

determine the instantaneous torque produced. Using appropriate parameterization and curve-fitting techniques an accurate model is developed.

The mechanical dynamics are coupled to the electromagnetic behavior through the electrical torque production. Acceleration is calculated from the difference between the electrical and load torque, referred to as the torque imbalance, and the system inertia. The loads opposing the electrical torque are estimated from measurements on the experimental drive system. Speed and rotor position are then integrated from the acceleration.

3.2 Flux Linkage-Current Measurements

As discussed in Chapter 2, VRM models are commonly based on the periodic relationship of flux linkage and current. This relationship is indirectly determined by performing experimental tests on the motor at various static rotor positions. The method used here measures the phase current generated by a given sinusoidal voltage. The flux linkage, $\lambda(t_k)$, for a given current, $i(t_k)$, is then given by

$$\lambda(t_k) = \int_0^{t_k} \left(V(t_k) - Ri(t_k)\right)dt + constant \tag{3.1}$$

where $V(t_k)$ is the terminal voltage and R is the winding resistance. R is assumed to be negligible which simplifies the modeling process, and does not introduce significant errors for flux linkage calculations at low current values necessary for the position estimation algorithm. This assumption, though, does affect the accuracy of the electrical torque calculations. Both the position and torque estimation accuracy are discussed in later chapters.

The integration limits and constant are set so that the resulting $\lambda - i - \theta$ relationship is symmetrical around zero current and flux-linkage. By assuming that $R = 0$ and that the excitation is sinusoidal, then $d\lambda / dt = V$. Furthermore the assumption of symmetry leads to the conclusion that $\lambda = 0$ when $V = max(V)$. If the lower integration limit, $t = 0$, is set to the time when the terminal voltage is maximized, $V(0) = max(V)$, then the integration constant is zero. Using an AC voltage excitation signal produces a $\lambda - i$ curve which exhibits symmetric hysteresis around zero current and flux linkage. This magnetic hysteresis phenomenon is the result of the grain-oriented, crystal structure of the motor's magnetic material [7]. The resulting flux linkage waveform is a sinusoid that is 90 degrees out of phase with respect to the AC voltage excitation. The current

waveform is a distorted sinusoid less than 90 degrees out of phase from the excitation due to the nonlinear $\lambda - i$ relationship and the magnetic hysteresis of the motor material. The DC relationship, or normal magnetization curve, through the origin is estimated by averaging the magnetizing and demagnetizing flux linkage for a given current.

The measurements for the VRM were taken in the manner described above. An AC voltage signal was applied to each phase at static rotor positions in regular intervals over the electrical period of the VRM, 90 degrees. The terminal voltage and current were recorded using a digital storage oscilloscope, and then the voltage data was integrated using the procedure described above to estimate the flux linkage. An example test experiment taken at 30 mechanical degrees on phase A of the experimental VRM is shown in Figure 3.1. Note that all rotor position measurements are expressed in mechanical degrees from the magnetic alignment position of the nearest rotor pole to the excited winding unless otherwise specified. The flux linkage calculated from the terminal voltage is shown in Figure 3.2, and Figure 3.3 shows the resulting $\lambda - i - \theta$ model for this rotor position including the normal magnetization curve estimated from the magnetizing and demagnetizing flux linkage. The normal curve was estimated by linearly interpolating the magnetizing and demagnetizing flux linkage measurments at fixed currents and then averaging the two interpolated flux linkages.

The test results from successive rotor alignment positions are combined to develop the experimental VRM $\lambda - i - \theta$ map shown in Figure 3.4. The measurements were taken on all three phases and found to be sufficiently equivalent and symmetrical, so the entire model could be represented by one family of $\lambda - i - \theta$ curves phase shifted by 60 degrees for each phase. In Section 3.3, the flux map is fit to position-dependent parameters of a function representing the $\lambda - i - \theta$ relationship analytically. To facilitate parameterization to the analytic function, a limited number of data points are collected and are identified by the markers shown in Figure 3.4. The data is plotted for rotor positions from 0 to 45 degrees. Since the self-inductance is assumed to be periodic and symmetric over 90 degrees, it follows that the flux linkage is related by $\lambda(\theta, i) = \lambda(90 - \theta, i)$ for $45 \leq \theta \leq 90$ degrees.

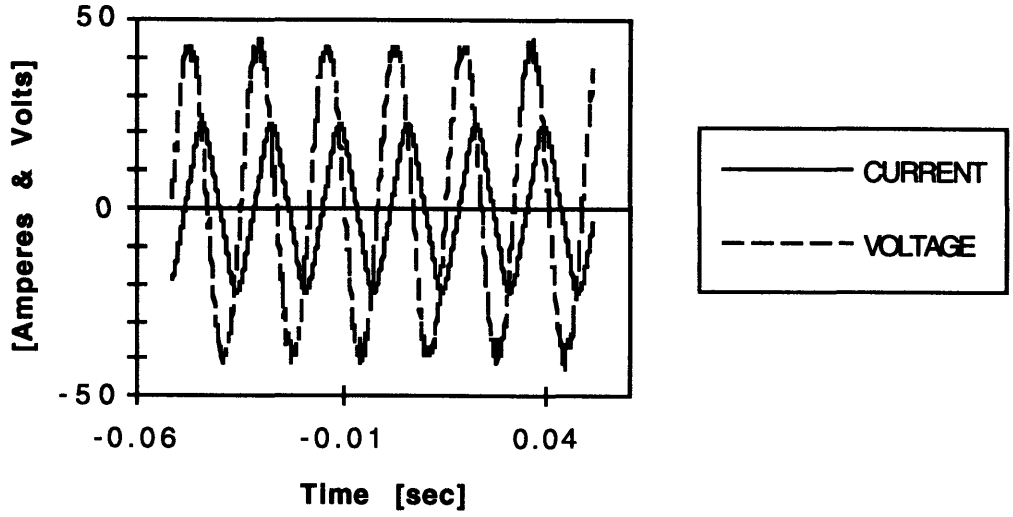

Figure 3.1: Phase current generated by applying a sinusoidal voltage at the fixed rotor position of 30 mechanical degrees from alignment for phase A.

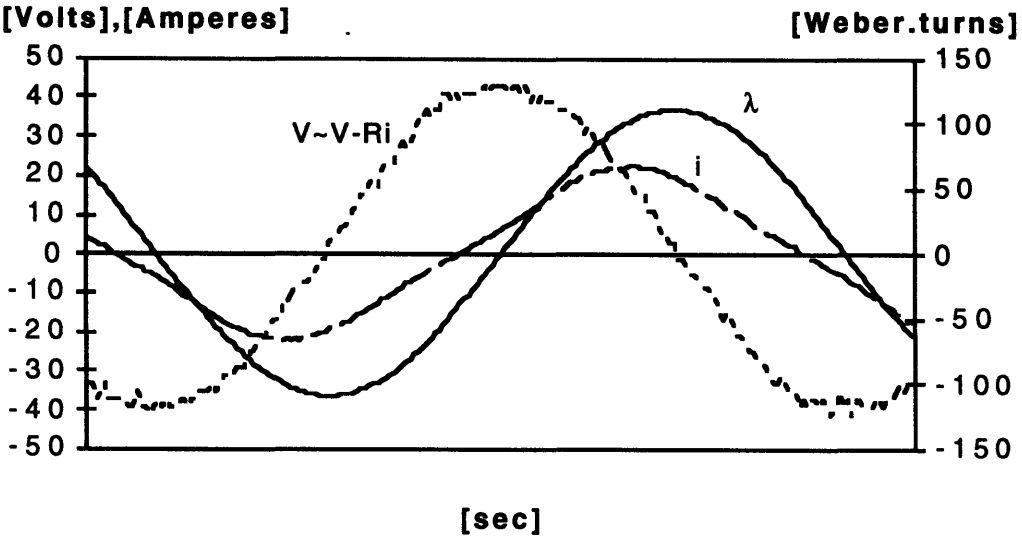

Figure 3.2: Flux-linkage integrated from the measured sinusoidal voltage at the fixed rotor position of 30 mechanical degrees from alignment for phase A.

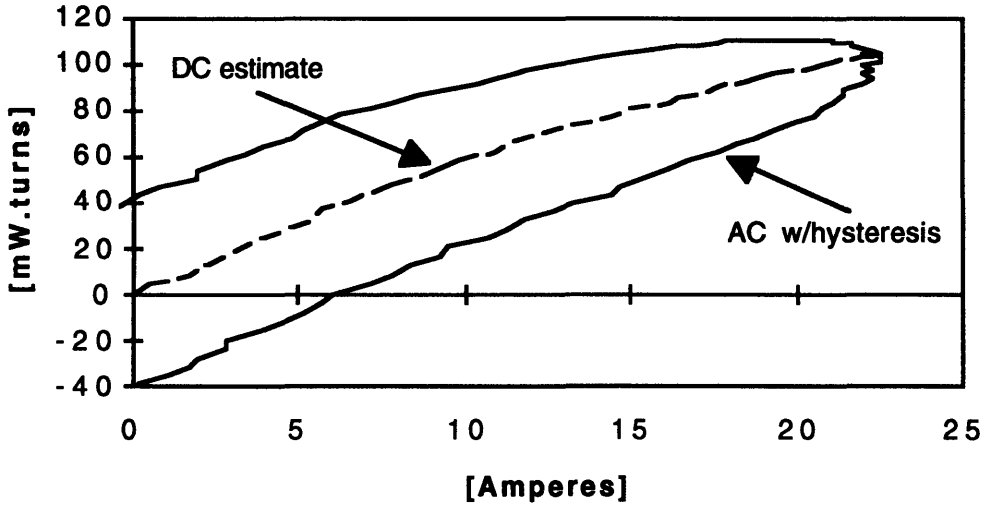

Figure 3.3: Flux-linkage versus measured current calculated at the fixed rotor position of 30 mechanical degrees from alignment for phase A.

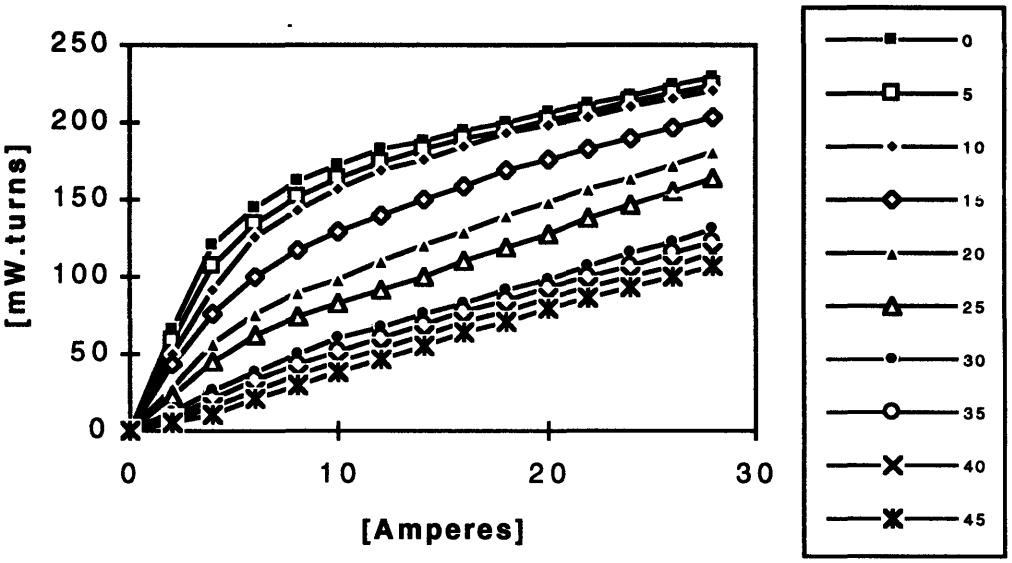

Figure 3.4: Flux linkage - current map determined from experimental measurements of phase current and terminal voltage at fixed rotor positions.

In this development a number of assumptions have been employed. By assuming phase independence, the total flux linkage is confined to the phase self-inductance. Thus measurements can be taken separately on each phase with the others unexcited. Using only measurements from one phase employed the assumptions that the phases are identically wound with a symmetrical separation of 60 degrees. Furthermore, the model developed assumes that when a phase is turned on there is neither residual flux nor realignment required for flux to develop (i.e. no magnetic hysteresis for a step change in the terminal voltage). The model does not represent possible eddy currents which are assumed to decay away within time intervals of interest. The errors introduced by these assumptions as they contribute to measurement and dynamic modeling accuracy are revisited in the discussion of the experimental results in Chapter 6.

3.3 Parameterization and Curve Fitting

The function introduced in Chapter 2 in Equation 2.1 is employed to transform the $\lambda - i - \theta$ map into a sufficiently accurate model for any condition of current and rotor position. The three function coefficients a_1, a_2, and a_3 are fit to the data points in Figure 3.4 using the Marquardt gradient-expansion algorithm [4] as functions of rotor position. The data points generated by the expansion algorithm become a lookup table of coefficients versus rotor position to be used in the model. The model uses a spline fit technique to provide a smooth interpolation for rotor positions between the data points.

The Marquardt Method is based on minimizing a merit function, χ^2, given a nonlinear model and a set of data points. The method improves on more general nonlinear techniques by closing rapidly on a solution even when the initial guesses for the coefficients are poor.

Evaluation of the merit function gives the least squares estimate of the error as

$$\chi(\boldsymbol{a})^2 = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}\left[y_i - y(\boldsymbol{a}, x_i)\right]^2, \quad n = \#\text{ of data points} \tag{3.2}$$

where (x_i, y_i) is the ith data point with an assumed standard deviation of σ_i, and $y(\boldsymbol{a}, x_i)$ is the corresponding nonlinear function value for the current estimate of the coefficients $\boldsymbol{a} = [a_1 \quad a_2 \quad a_3]$. The gradient components

$$b_k = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}\left[y_i - y(\boldsymbol{a}, x_i)\right]\frac{\partial y(a, x_i)}{\partial a_k}, \quad k = 0, 1, \& 2 \tag{3.3}$$

and a curvature matrix

$$\alpha_{jk} = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}\frac{\partial y(x_i)}{\partial a_j}\frac{\partial y(x_i)}{\partial a_k}, \quad j = 0, 1, \& 2 \tag{3.4}$$

are also calculated for the current $\boldsymbol{a}$. The Marquardt Method uses a scaling factor, Λ, applied to the curvature matrix components,

$$\varepsilon_{jk} = \begin{cases} \dfrac{\alpha_{jk}(1+\Lambda)}{\sqrt{\alpha_{jj}\alpha_{kk}}} & j = k \\ \dfrac{\alpha_{jk}(1+\Lambda)}{\sqrt{\alpha_{jj}\alpha_{kk}}} & j \neq k \end{cases} \tag{3.5}$$

and the gradient components to calculate the next guess for the coefficients

$$a_j' = a_j + \sum_{k=0}^{m} b_k \varepsilon_{jk}^{-1}, \quad m = 2; j = 0, 1, \& 2 \tag{3.6}$$

The trending of the merit function determines whether the scaling factor is increased or decreased according to

$$\begin{cases} \text{If } \chi^2(\boldsymbol{a}') > \chi^2(\boldsymbol{a}) & \text{then } \Lambda' = 10\Lambda \\ \text{If } \chi^2(\boldsymbol{a}') > \chi^2(\boldsymbol{a}) & \text{then } \Lambda' = \Lambda / 10 \end{cases} \tag{3.7}$$

where $\boldsymbol{a}'$ and Λ' are the next guesses for the coefficient matrix and scaling factor respectively.

The effect of Λ is to weight the iterative solution process so that it converges by the steepest descent method initially and then by the inverse-Hessian method as the minimum is approached. This produces a method that converges even when the initial guess is poor and reduces the number of iterations by varying the weight of terms. Coefficients for the experimental VRM are calculated using the Marquardt Method described above to fit Equation 2.1 to the data represented by the markers in Figure 3.4. Table 3.1 is the Marquardt solution for the coefficients given this experimental VRM test data. Due to the periodicity and symmetry of each electrical cycle, the curve fit is fully described over half a cycle (45 degrees). The solution points are used in a

lookup table as a function of rotor position. The cubic spline fit shown in Figures 3.5, 3.6, and 3.7 then provides a smooth interpolation between these points to be used the VRM simulation and observer algorithm. C code implementations of the Marquardt Method and the cubic spline are found in *Numerical Recipes in C* [4].

Table 3.1: Flux linkage model coefficients generated as a function of rotor position by the Marquardt algorithm for one half of an electrical cycle. The second half of the cycle is symmetric about 45 degrees.

Rotor Position	a1	a2	a3
0	0.151906	-0.305662	0.002493
5	0.1505	-0.29	0.00252
10	0.142	-0.28	0.00254
15	0.125	-0.277	0.0027
20	0.098	-0.274	0.00290667
25	0.07	-0.265	0.00311333
30	0.045	-0.24	0.00332
35	0.022	-0.167	0.00346
40	0.009	-0.07	0.00349
45	0.007056	-0.0052784	0.0035

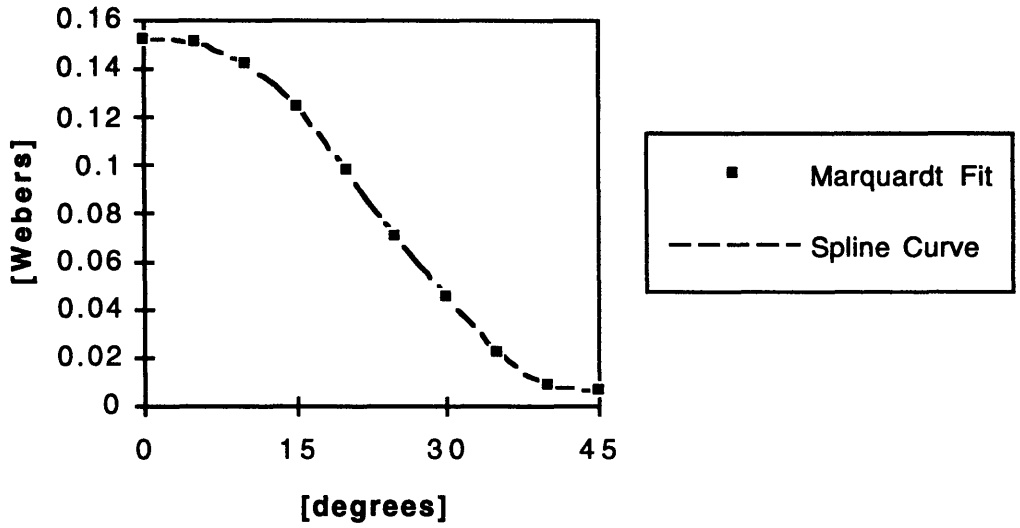

Figure 3.5: The analytic function parameter a_1 as a function of rotor position.

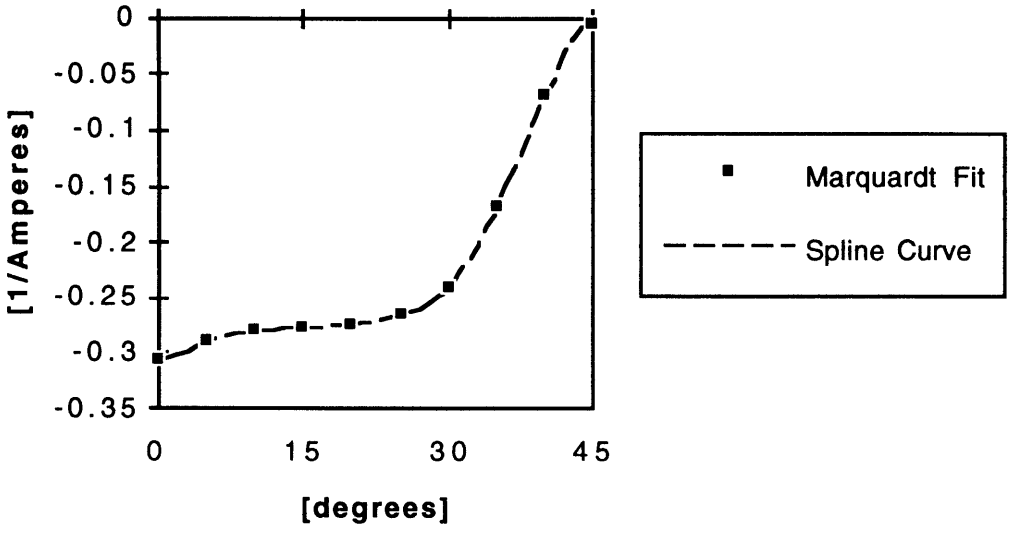

Figure 3.6: The analytic function parameter a_2 as a function of rotor position.

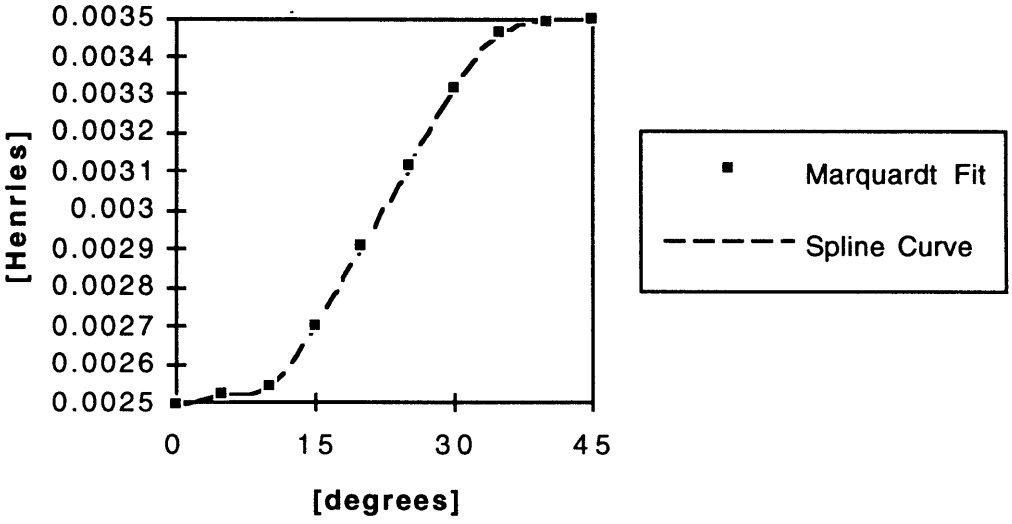

Figure 3.7: The analytic function parameter a_3 as a function of rotor position.

3.4 Current-Flux Mapping

The coefficients given in Table 3.1 are applied to Equation 2.1 to reproduce a flux linkage map for the motor as shown in Figure 3.8. This model technique represents the relationship well as demonstrated in Figure 3.9 which compares the original data to the model representation at the rotor position of 10 degrees. This level of accuracy is typical for all rotor positions. More importantly as shown in the following section, the resulting modeled torque is also matches experimental measurements well using this technique. The modeled phase inductance at 2 Amperes excitation is shown in Figure 3.10. This represents the linear inductance region before the onset of saturation. This inductance profile becomes significant for the development of the observer measurement algorithm which is discussed in Chapter 6.

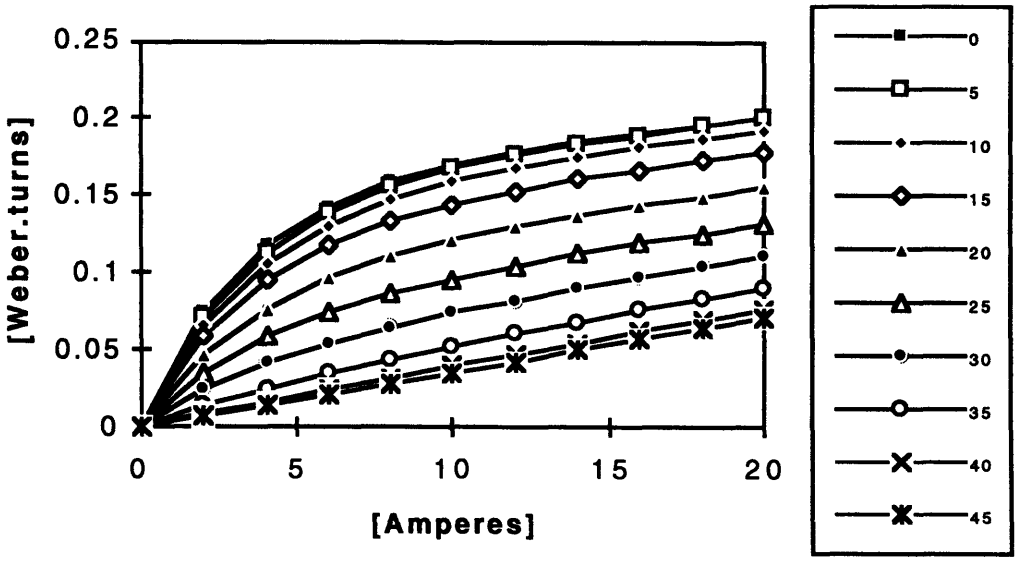

Figure 3.8: The modeled flux-linkage as a function of winding current and rotor position. The data points are calculated from Equation 2.1 using the parameter values given in Table 3.1.

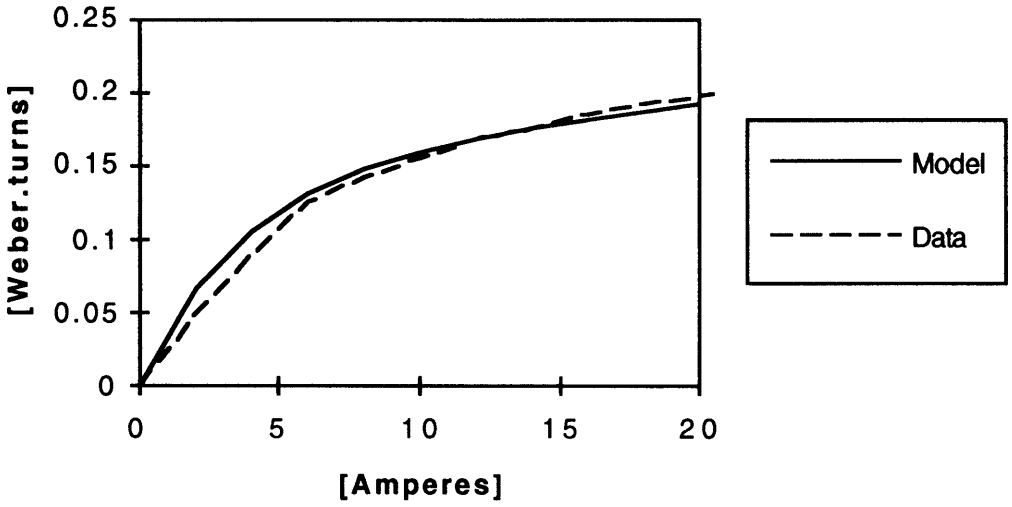

Figure 3.9: Comparison of flux linkage from nonlinear model and calculated data from experimental measurements at 10 degrees rotor position.

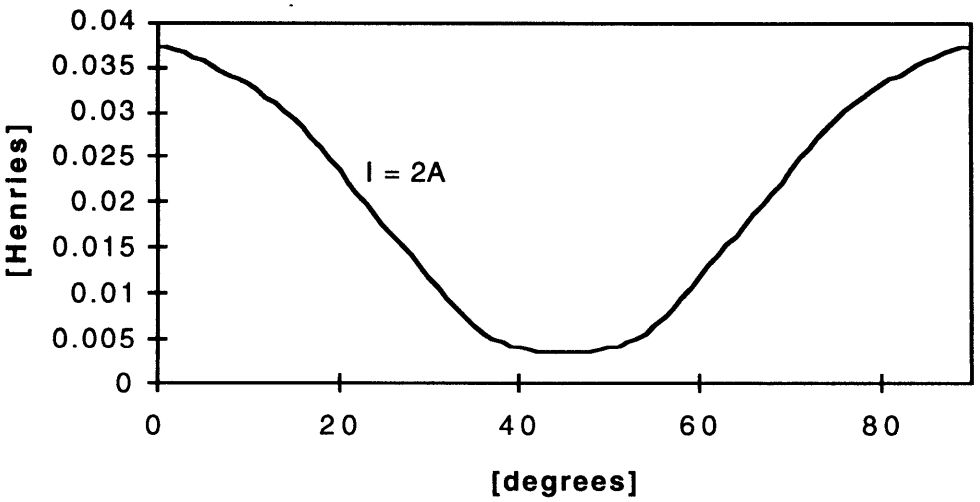

Figure 3.10: Phase inductance for low current (2 Amperes). This is the linear region of the model. Rotor positions from 0 to 90 degrees are shown to demonstrate the symmetry of the model.

3.5 Instantaneous Torque

The electromagnetic torque produced by the VRM is found from the flux linkage model using energy methods. The instantaneous torque can be expressed in terms of the stored coenergy, W_p', in each phase [6] by

$$T_e = \sum_p^{a,b,c} \tau_p(i,\theta) = \sum_p^{a,b,c} \left. \frac{\partial W_p'}{\partial \theta} \right|_{i=const} \tag{3.8}$$

where τ_p is the torque produced by one phase, and T_e is the total electromagnetic machine torque. The simple summation over the three phases of the experimental VRM assumes that the three phases are electromagnetically independent. Furthermore, the phase coenergy is equal to the path integral of flux linkage and current in the winding circuit as shown by

$$W_p' = \left. \int_0^i \lambda(\tilde{i},\theta) d\tilde{i} \right|_{\theta=const} . \tag{3.9}$$

Substituting the flux linkage model, Equation 2.1, into Equation 3.9 gives the torque per phase in terms of the current and nonlinear model coefficients evaluated at the instantaneous rotor position. This yields

$$\begin{aligned} \tau = & \left[i + \frac{1}{a_2}(1 - e^{a_2 i}) \right] \frac{da_1}{d\theta} \\ & - \left[\frac{a_1}{a_2^2}(1 - e^{a_2 i}) + \frac{a_1 i}{a_2} e^{a_2 i} \right] \frac{da_2}{d\theta} \\ & + \frac{1}{2} i^2 \frac{da_3}{d\theta} \end{aligned} \tag{3.10}$$

where the electrical phase torque subscript has been dropped for clarity.

Evaluation of the coefficient derivatives gives an indication of the ability of the curve-fitted model to predict the torque accurately. Figures 3.11, 3.12, and 3.13 show the derivative functions as determined from the spline-interpolated data table. The total VRM torque is calculated using Equation 3.10 and the evaluated derivatives, and is shown in Figure 3.14.

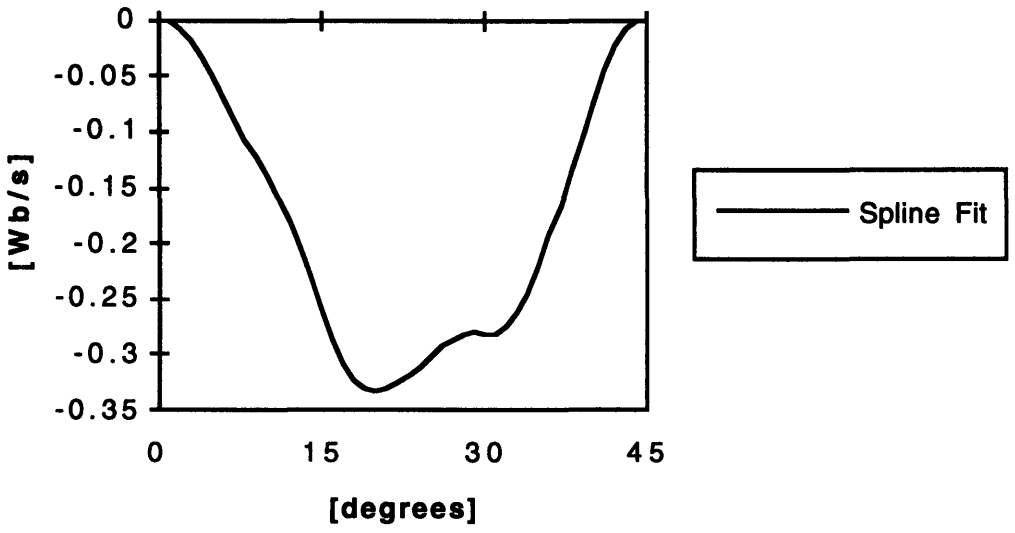

Figure 3.11: The analytic function derivative parameter da_1/dt as a function of rotor position.

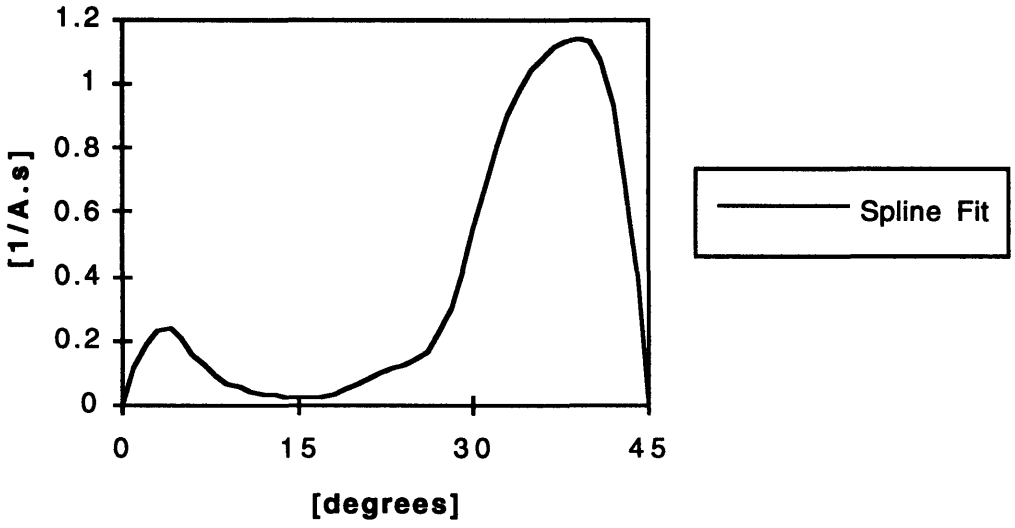

Figure 3.12: The analytic function derivative parameter da_2/dt as a function of rotor position.

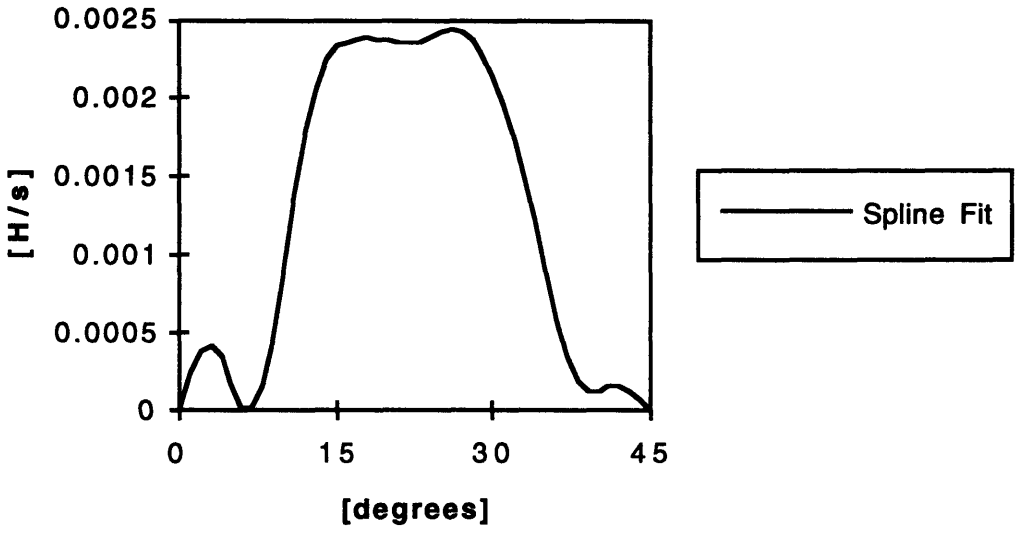

Figure 3.13: The analytic function derivative parameter da_3/dt as a function of rotor position.

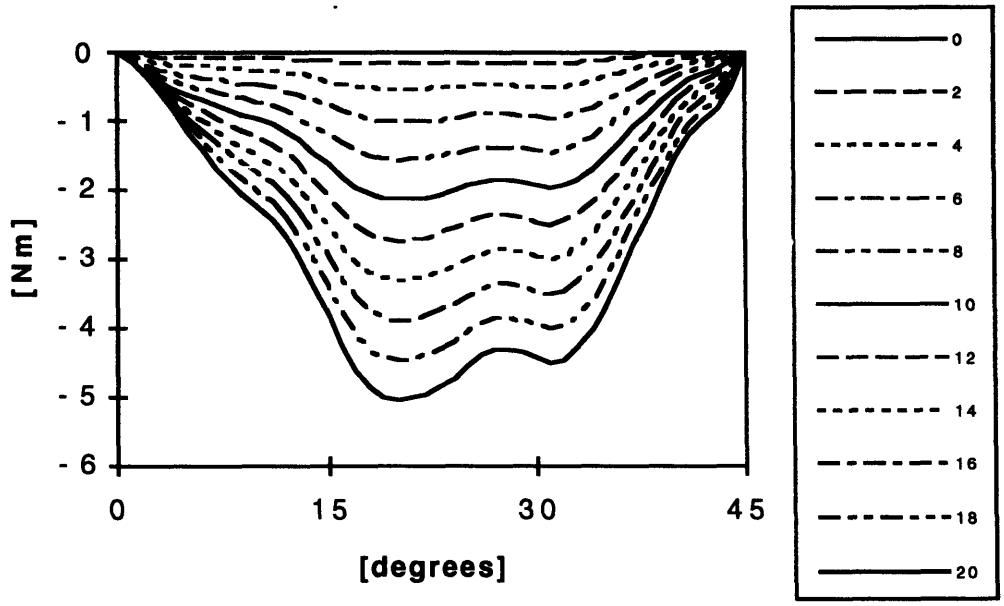

Figure 3.14: The modeled instantaneous torque produced as a function of rotor position and phase current.

Coefficient estimates were also calculated using a partial Fourier series representation prior to implementation using the cubic spline. The series expansion method was based on previous work by Torrey [3]. The Fourier series produces torque ripple which does not exist in the laboratory motor. The torque profile resulting from the cubic spline interpolation of the coefficients, though, provides a smoother model which more accurately represents reality.

In addition to the static AC terminal voltage - current measurements, static torque measurements were taken to provide independent verification of the model profile. The same phase winding used to create the flux linkage map was excited with a DC current source to measure the static torque at different rotor positions. Using a strain gage sensing block and a moment arm clamped between the rotor and the sensing block, rough measurements of the force exerted by the rotor on the clamping rig were recorded. Based on the geometry of the rig and motor, the VRM torque was calculated from the force and moment arm length as shown in Figure 3.15. Figure 3.16 demonstrates the typical errors observed when comparing this calculated torque to the torque model developed from the flux linkage. It was not possible to accurately set the alignment angle 0.0 degrees accurately with the force measurement rig, so the data may be shifted left or right. Additionally the measurements have a built in bias towards misalignment at 45 degrees due to the shaft compliance towards alignment. Due to this uncertainty the main feature of interest in these plots is the shape of the profile. As stated earlier, a smooth torque model requires not only accurate modeling of the coefficients, but of their derivatives as well. Sharp gradients in the coefficients due to sparse modeling data and minimal data measurement filtering could result in drastically altering the resulting torque model shape which are not easily observed in the flux linkage model shape. Figure 3.16 indicates that the flux linkage model predicts the instantaneous torque production reasonably well in shape, but the magnitude of the peak torque uncertainty with respect to the independent force measurements reaches 15% to 20%. The level of model uncertainty should be kept in mind when considering the performance of the observer design in later chapters. The uncertainty affects the accuracy of the calculated average torque for the observer plant model, and directly affects the position sensing accuracy for the observer innovations.

If there were equal confidence in both data measurement methods, steps could be taken to weight the two estimates for a more accurate machine model. Since the electromagnetic torque is related to the coenergy by Equation 3.8, the coenergy can be calculated from the force measurement data. The total coenergy can then be determined as a weighted average of calculations based on electrical terminal measurements and mechanical force measurements. From this averaged coenergy either the torque or flux linkage can be determined by taking the appropriate partial derivatives as defined by Equations 3.8 and 3.9. Since the objectives of this research are well served with some

model uncertainty, and since there is less confidence in the force measurement experiments this methodology was not employed for the final model.

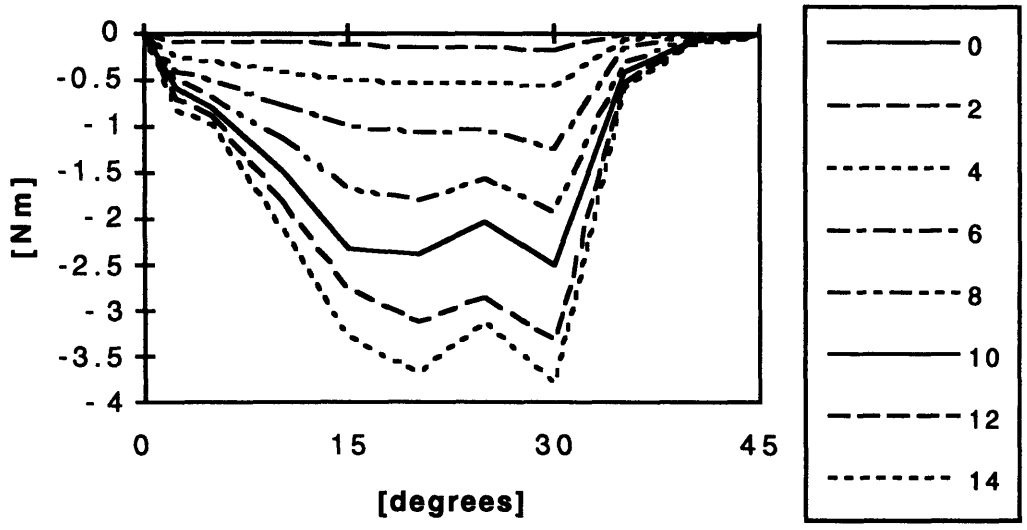

Figure 3.15: Static torque measured from the force sensor and DC current excitation.

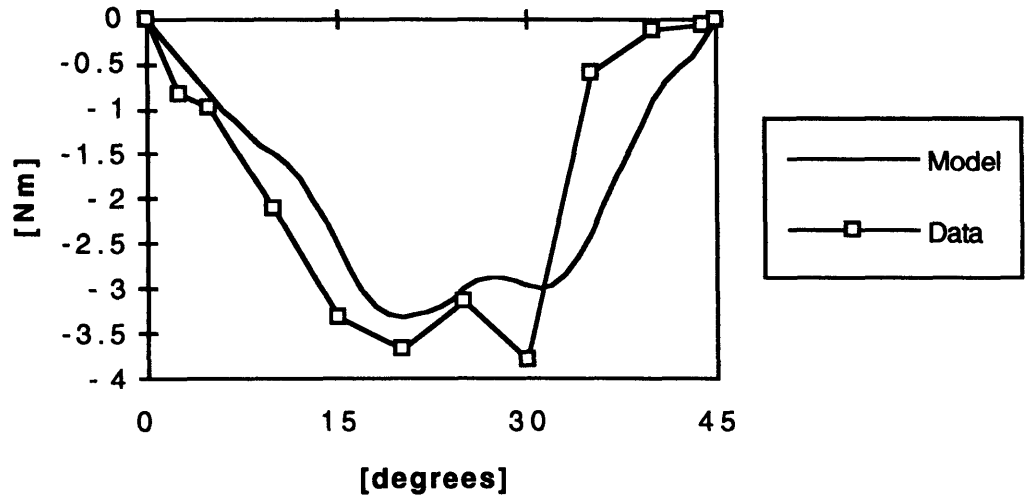

Figure 3.16: Comparison of instantaneous torque from nonlinear model and calculated data from experimental measurements at 14 Amperes DC. Shows typical peak torque uncertainty.

3.6 Transient Model

The model is completed by developing the equations governing the dynamic behavior of the VRM. The fifth order system representing the electrical and mechanical dynamics is given by

$$\frac{d\theta}{dt} = \omega \tag{3.11}$$

$$\frac{d\omega}{dt} = \frac{1}{J}(T_e - T_L - B\omega - C) \tag{3.12}$$

$$\frac{d\lambda_p(i_p,\theta)}{dt} = V - Ri_p, \text{ where } p = \text{a,b, and c.} \tag{3.13}$$

where $\omega \geq 0$ for these and all further equations. For this system, J represents the mechanical inertia, B and C are friction terms, T_L is a system load term, and V and R are the electrical terminal conditions which are identical for each phase. Note that the winding resistance, R, has been reintroduced to account for operation at high current levels where the voltage drop may no longer be negligible depending on the relative voltage supply level, V. The flux linkage is computed iteratively using Equation 3.13 and the proposed flux linkage model, Equation 2.1, which is repeated here for clarity,

$$\lambda = a_1(\theta)(1 - e^{a_2(\theta)i}) + a_3(\theta)i. \tag{3.14}$$

where the phase index has been dropped since they are assumed identical electrically with a mechanical phase shift of 60 degrees.

To simulate VRM dynamics using this model with a computer simulation, the system is approximated with a discrete model with a given position step size, $\Delta\theta$, so that

$$\Delta t = \frac{\Delta\theta}{\omega_k} \tag{3.15}$$

where ω_k is the VRM speed at time index k. In this simulation, the step transition is chosen to be $\Delta\theta$ rather than Δt to ensure comparable modeling precision at different speeds and conduction angles. Of course, for a real-time simulation the appropriate choice would be Δt. This is the case for the real-time observer model used in the experimental drive which is discussed in Chapter 6. The remaining discrete system

equations are formed using a forward Euler approximation. The discrete equations are given by

$$\theta_{k+1} = \theta_k + \Delta\theta, \tag{3.16}$$

$$\omega_{k+1} = \omega_k + \frac{\Delta\theta}{\omega_k J}(T_e - T_L - C) - \frac{\Delta\theta}{J}B, \tag{3.17}$$

$$\lambda_{k+1} = \lambda_k + \frac{\Delta\theta}{\omega}(V - Ri_{k+1}), \text{ and} \tag{3.18}$$

$$\lambda_k = a_1(\theta_k)(1 - e^{a_2(\theta_k)i_k}) + a_3(\theta_k)i_k, \tag{3.19}$$

where Equations 3.18 and 3.19 are solved iteratively and T_e is given by Equations 3.8 and 3.10 evaluated at i_k for each phase.

With this model, the VRM can be simulated for constant control angle operation. The difference equations are simulated with the terminal voltage given by

$$V = \begin{cases} V_b, & \theta_{on} \le \theta < \theta_{on} + \theta_{cond} \\ -V_b, & \theta_{on} + \theta_{cond} \le \theta < \theta_0 \\ 0, & otherwise \end{cases}, \tag{3.20}$$

θ_{on} is the rotor position when the phase FETs are turned on, $\theta_{on} + \theta_{cond}$ is when they are turned off and current falls off through the fly back circuit on the inverter, θ_0 is the position at which current returns to zero, and V_b is the DC supply voltage. A current chopping limit, i_{chop}, is also incorporated into the simulation to prevent overcurrent conditions and approximate the analog current limiting in the experimental VRM drive. It is a soft current limit which turns the phase back on when the current falls below a hysteresis limit, i_{hyst}.

The inertia and damping constants, J, C, and B in Equation 3.17 are determined through measurements on the experimental drive system. For the initial experiments there is no external load function so $V_L = 0$. The viscous damping is estimated by operating the experimental VRM at constant speed, calculating the average torque produced using the simulation, and then dividing by the VRM speed. The inertia and Coulombic friction values are determined from a spin down test. From steady-state operation, the electrical supply is shut off to the VRM and it is allowed to coast down to zero speed. The speed versus time trace of the spin down is fit to the following function

$$\frac{d\omega}{dt} = \frac{1}{J}(-B\omega - C). \tag{3.21}$$

which is Equation 3.12 with the electrical torque and load function terms set to zero. The curve fit produces estimates for *B/J* and *C/J*. Using the previous estimate for *B*, both *J* and *C* are determined uniquely.

Figures 3.17 through 3.19 show the simulated current, flux linkage, and torque for one phase over an electrical cycle for the operating conditions given in Table 3.2. Figure 3.20 shows the total VRM torque with phase shifted contributions from each phase. The current hysteresis observed in Figure 3.17 may be more than the specified hysteresis due to the step size of the simulation. Negative torque production is visible in Figure 3.19 because the phase is turned on before maximum misalignment. As described in the previous chapter, the torque produced tends to align the poles which for this condition is in the direction opposite to the rotor motion for conduction between θ_{on} and 45 degrees. The importance of relatively slow mechanical dynamics is apparent from the significant machine torque variation with position produced by the VRM as shown in Figure 3.20 . With fast mechanical dynamics the motor speed would vary noticably with the instantaneous torque while with the slow dynamics the motor speed is relatively constant over one electrical cycle even if a torque imbalance exist as is the case in this particular simulation. This torque ripple is characteristic of VRMs and is more pronounced for lower conduction angles.

Table 3.2: VRM Simulation for One Electrical Cycle.

Parameter	Units	Value
V_b	[Volts]	68
ω	[rpm]	2000
θ_{on}	[degrees]	32
θ_{cond}	[degrees]	65
i_{chop}	[Amperes]	20.0
i_{hyst}	[Amperes]	0.654
J	[kg.m^2]	0.00708
B	[kg.m^2/sec]	0.000531
C	[Nm]	0.252
T_L	[Nm]	0.0

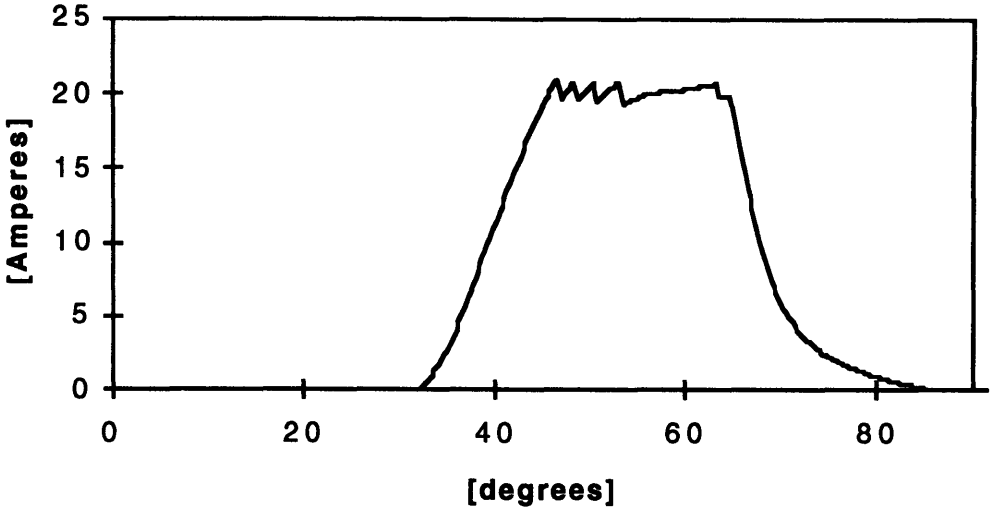

Figure 3.17: Current simulation for phase A at 2000 rpm.

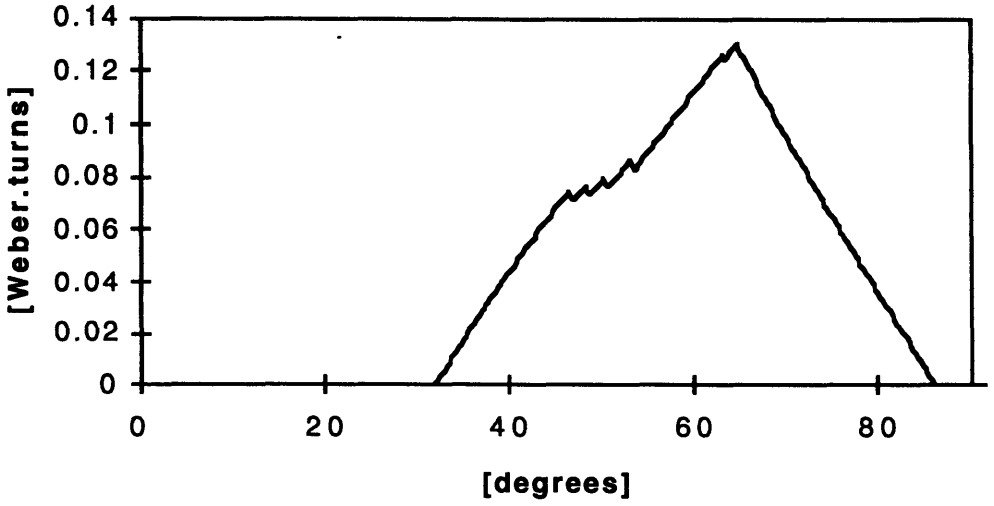

Figure 3.18: Flux linkage simulation for phase A at 2000 rpm.

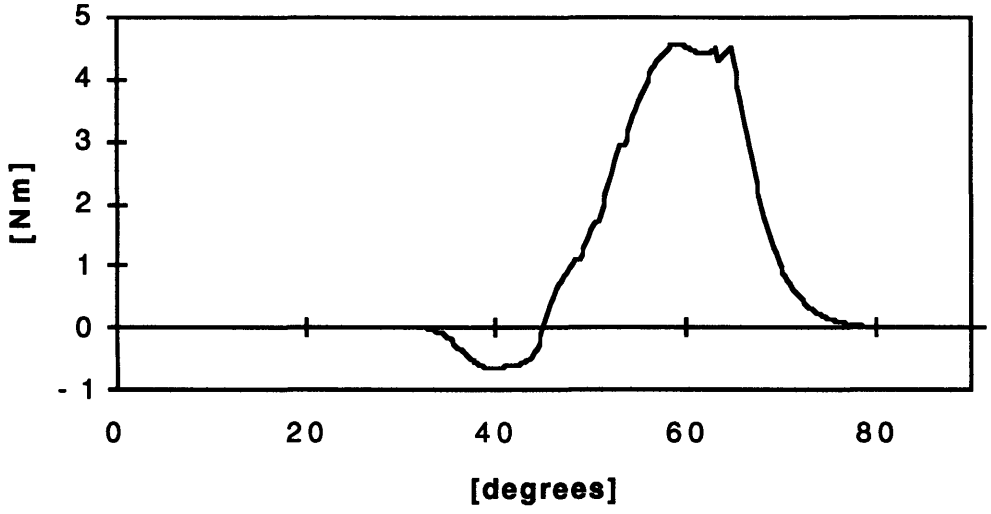

Figure 3.19: Instantaneous torque simulation for phase A at 2000 rpm.

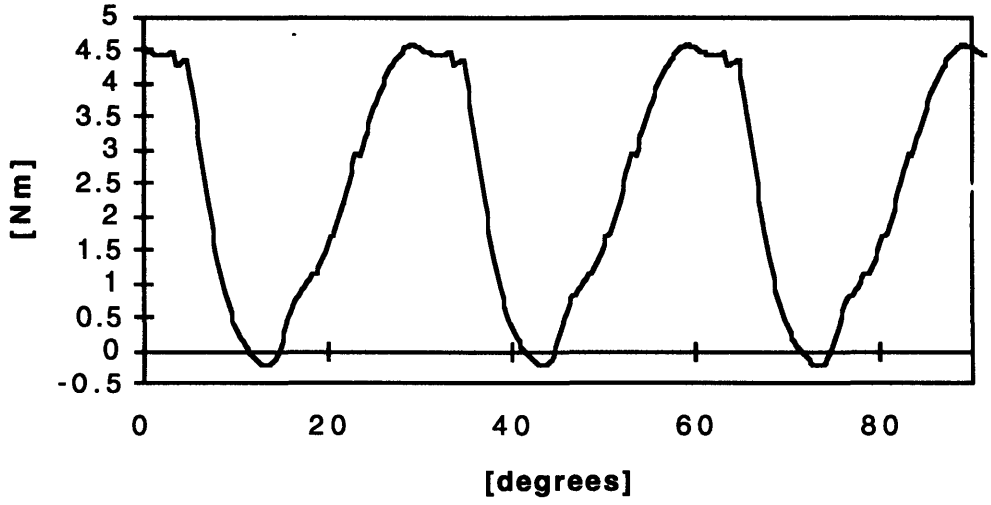

Figure 3.20: Instantaneous motor torque simulation at 2000 rpm. The motor torque is the sum of the three identical phase torques separated by phase angles of 30 degrees.

To conclude and further illustrate the simulation development, the discrete model is used to map the average torque of the candidate VRM at constant speed and various commutation angles. The average torque produced by the motor is determined by

calculating the torque produced at each conducting position step where the average torque is then given by

$$\langle T_e \rangle = \frac{P\Delta\theta}{\Theta_{EC}} \sum_{k=\frac{\theta_{on}}{\Delta\theta}}^{\frac{\theta_o}{\Delta\theta}} \tau(i_k) \tag{3.22}$$

where P is the number of phases and Θ_{EC} is the length of an electrical cycle. Since each phase contributes identically to the average torque only one phase torque τ is calculated and then multiplied by P = 3.

Figures 3.21 through 3.26 show the average motor torque produced for all potential θ_{on} and θ_{cond} in increments of 5 degrees. This shows how both negative and positive torque can be produced to accelerate and decelerate the VRM drive. Maximum positive torque output is achieved by turning on each phase prior to the point of maximum misalignment. Even though negative instantaneous torque is produced while $\theta < 45$ degrees, turning on the phases early allows the current to rise so that the positive instantaneous torque produced after $\theta > 45$ is more significant. Figure 3.27 summarizes this aspect showing that the maximum torque decreases with increasing motor speed. This follows intuition since at higher speeds the current has less time available to rise before $\theta > 45$ given a constant θ_{on}. This information is used to develop limits for the controller in the following chapter.

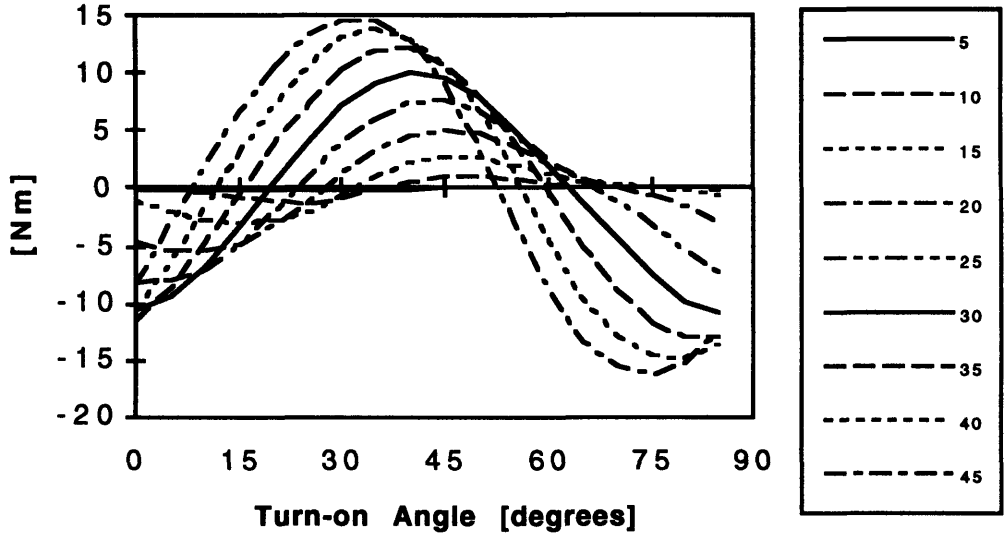

Figure 3.21: The modeled average torque produced as a function of turn-on and conduction angles at 2000 rpm.

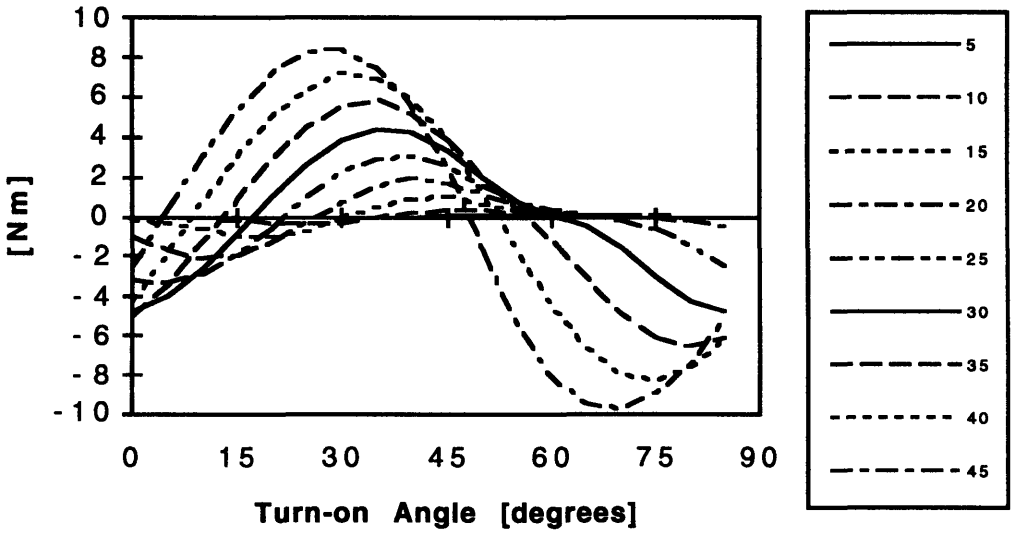

Figure 3.22: The modeled average torque produced as a function of turn-on and conduction angles at 4000 rpm.

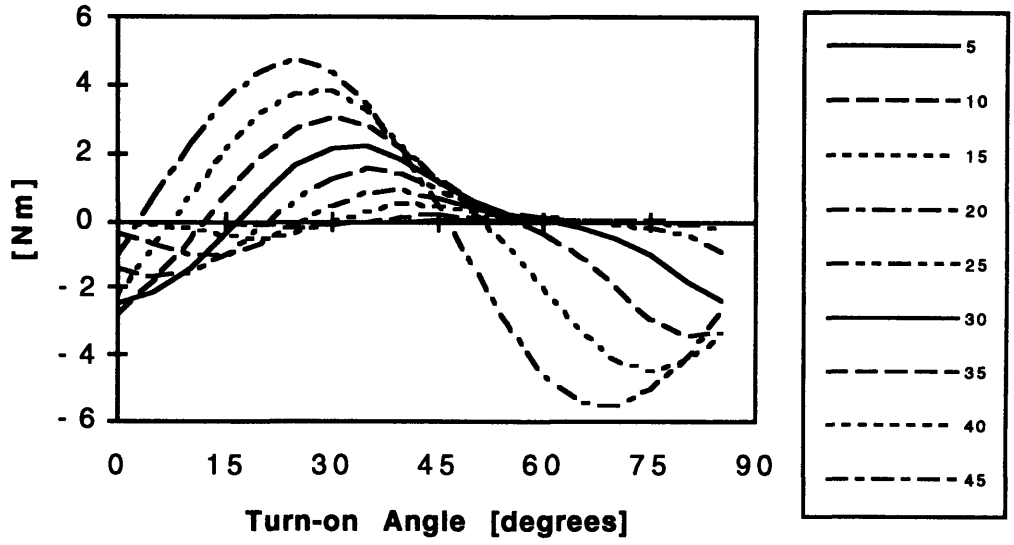

Figure 3.23: The modeled average torque produced as a function of turn-on and conduction angles at 6000 rpm.

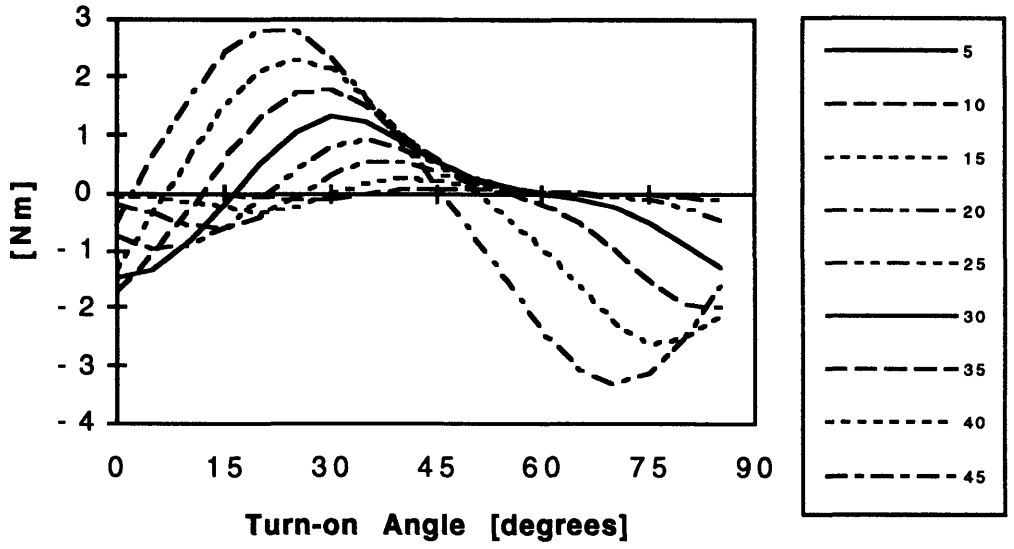

Figure 3.24: The modeled average torque produced as a function of turn-on and conduction angles at 8000 rpm.

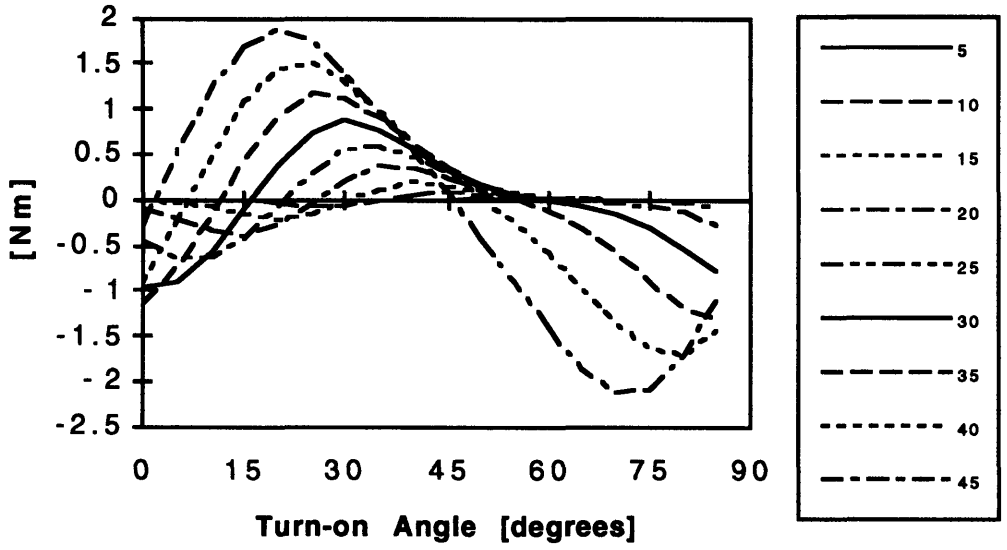

Figure 3.25: The modeled average torque produced as a function of turn-on and conduction angles at 10000 rpm.

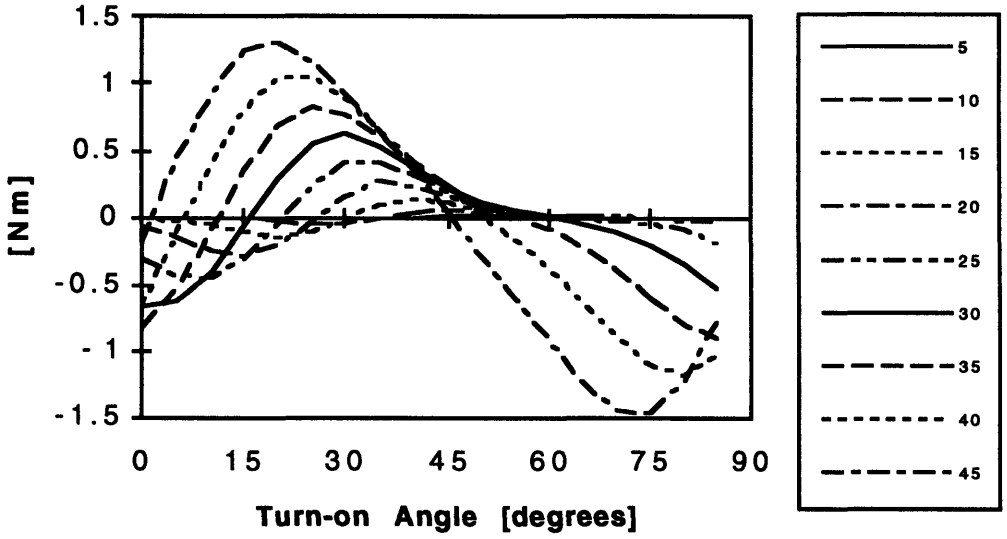

Figure 3.26: The modeled average torque produced as a function of turn-on and conduction angles at 12000 rpm.

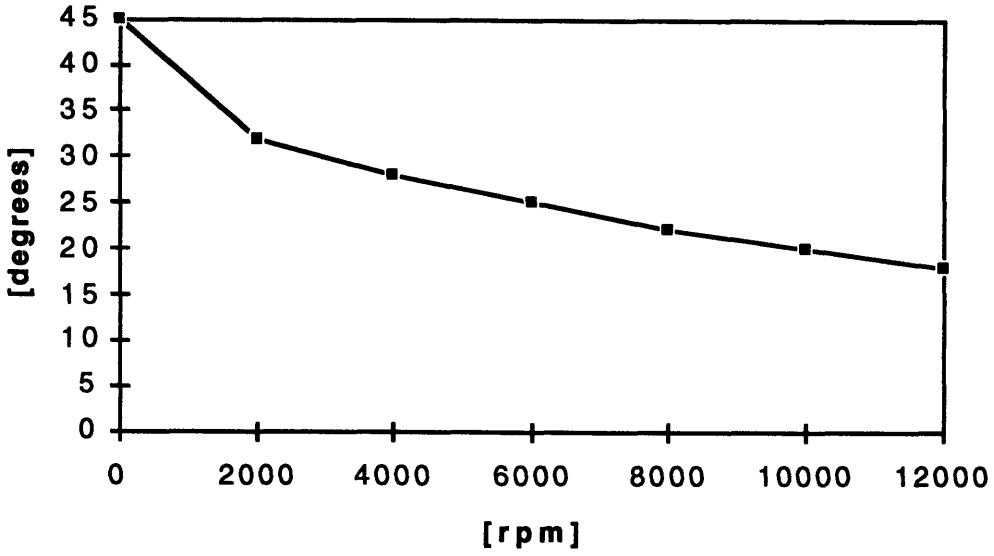

Figure 3.27: The turn-on angle which produces the maximum average torque with 45 degrees conduction.

3.7 Summary

This chapter presented the VRM model based on a nonlinear flux linkage - current relation and linear electrical terminal and mechanical dynamics. The model is the basis of both the drive simulation introduced in this chapter, and the observer design. Modeling methods and aspects that may affect the accuracy of the model, including the data measurement and curve fit accuracy, were discussed since this ultimately affects the stability of the controller and observer that will be discussed in the following two chapters.

The flux linkage model development from experimental measurements was presented first. The data was collected using AC terminal excitation on the phases. The DC flux linkage - current relationship was estimated from this data, and fit to a nonlinear function with position-dependent coefficients. The Marquardt expansion algorithm provided a reasonable method for fitting the coefficients to the supplied data. A smooth interpolation was provided by spline interpolation. This avoided the torque ripple inherent in using low order Fourier series representations. The resulting flux

linkage model showed a reasonable fit to the original data in Figure 3.9 with some inaccuracy in the curvature where saturation onsets and slight discrepancies in the bulk saturation slope. Furthermore independent force measurements indicated possible errors in the modeled peak torque.

VRM torque production was then developed based on an energy method calculation using the nonlinear flux linkage equation. In order to use the calculated model coefficients, the coefficient derivatives were also determined. The connect between smooth derivatives and an accurate torque model was discussed. The resulting calculated torque was compared to rough static experimental measurements of static motor torque from DC excitation to verify that the calculated torque profile had not been distorted.

The flux linkage and related torque models were then incorporated into a dynamic model for the transient speed drive system. The equations were discretized to a fixed $\Delta\theta$ step algorithm and restated. Finally performance maps of the VRM were presented to show the effects of varying θ_{on} and θ_{cond}, and motor speed, and to determine the θ_{on} corresponding to maximum average torque production. This information will be used in the presentation of the speed controller in the next chapter.

Chapter 4

THE DRIVE CONTROLLER

4.1 Introduction

The development of a controller for the VRM drive is presented in this chapter. It is tested in this chapter through simulation, and in the subsequent chapters through experimentation. The main objective of the controller design is to provide stable transient speed operation thus enabling the evaluation of the observer performance over a wide range of conditions. In this , the design is not specifically optimized for trajectory control, speed of response, or efficiency. The controller enables steady-state operation at speeds ranging from 0 to 10krpm, as well as both minor and major speed transients, by varying both the turn on and conduction angles, θ_{cond} and θ_{on}. Due to the nonlinear nature of the experimental drive hardware and the corresponding simulation model, stability analysis is conducted to establish gain boundaries based on the known operating regions.

The basic structure of the controller is linear and is driven by a speed error. To exercise control, the angles θ_{cond} and θ_{on} are modulated from reference values to drive the speed error asymptotically to zero. Limits on the controller actions are added to maintain operation within reasonable commutation angles and safe phase currents. The control of θ_{on} is simplified by setting the lower limit at the angle which maximizes torque production. Current limiting is provided in the experimental hardware and modeled appropriately in the simulation controller. These augmentations to the controller require further restrictions to maintain stability which is discussed below.

For the stability analysis and control gain selection, significant time scale separation between the electrical and mechanical time constants in the motor is assumed. By assuming that motor speed is approximately constant over an electrical cycle, the flux linkage - current transients can be ignored and average torque can be used for the

proportional plus integral gain selections. Specific torque-speed operating points are analyzed with respect to perturbations around dynamic equilibria to calculate gain boundaries for closed loop stability. Note that many of the equilibrium points may be unstable as can be inferred from the torque maps developed in Section 3.6. An operating point is open-loop unstable when an incremental increase in speed at constant commutation angles results in positive torque imbalance thus driving the motor speed even higher. Likewise the operating point is also unstable if a small decrease in speed results in a negative torque imbalance.

The analysis used here uses a similar approach to the design development of Torrey [3]. Torrey's method exploited results from VRM control literature and dynamic systems theory to represent the nonlinear plant as a linear controllable model. The analysis focused on modeling the electromagnetic interactions as functions of rotor position as developed in Chapters 2 and 3, and using time scale separation properties commonly employed in machine analyses with large system inertias.

4.2 Stability Analysis

The VRM controller is designed according to the structure posed in Figure 2.5. The error between a target motor speed, Ω, and the actual motor speed, ω, drives a proportional plus integral function of the speed error defined by

$$\hat{\omega} = \omega - \Omega \tag{4.1}$$

$$\hat{\omega}_{PI} = K_P \hat{\omega} + K_I \int \hat{\omega} dt \tag{4.2}$$

where K_P and K_I are the control law gains. The PI-compensated speed error, $\hat{\omega}_{PI}$, drives changes in the initial commutation angles, Θ_{on} and Θ_{cond}, given by

$$\hat{\theta}_{on} = \theta_{on} - \Theta_{on} = K_{on} \hat{\omega}_{PI}, \quad \theta_{on} \geq \theta_{on,min} \tag{4.3}$$

$$\hat{\theta}_{cond} = \theta_{cond} - \Theta_{cond} = K_{cond} \hat{\omega}_{PI}, \quad \theta_{cond} \leq 45^o \tag{4.4}$$

where K_{on} and K_{cond} are the turn on and conduction angle gains respectively.

The gain selection proceeds by performing a linearized stability analysis. The controller design is simplified by approximating the mechanical model of Equation 3.12 by

$$J\frac{d\omega}{dt} = \langle T_e \rangle - T_L - B\omega - C, \tag{4.5}$$

which relies on the assumption that the mechanical dynamics are slow compared to the electrical dynamics. Thus average torque, $\langle T_e \rangle$, can replace instantaneous torque as shown in Equation 4.5. In dynamic equilibrium, defined as constant speed operation, the torque load and output become balanced and therefore

$$0 = \langle T_e \rangle - T_L - B\Omega - C. \tag{4.6}$$

By assuming adequate knowledge of the load, the initial control angles are defined to produce the equilibrium required torque output. Then Equation 4.5 can be expanded around the equilibrium operating condition as given by

$$J\frac{d\hat{\omega}}{dt} = \left.\frac{\partial\langle T_e \rangle}{\partial\hat{\omega}}\right|_{\Theta_{on},\Theta_{cond}} \hat{\omega} + \left.\frac{\partial\langle T_e \rangle}{\partial\hat{\theta}_{on}}\right|_{\Omega,\Theta_{cond}} \hat{\theta}_{on} + \left.\frac{\partial\langle T_e \rangle}{\partial\hat{\theta}_{cond}}\right|_{\Omega,\Theta_{on}} \hat{\theta}_{cond} - \frac{\partial T_L}{\partial\hat{\omega}}\hat{\omega} - B\hat{\omega}, \tag{4.7}$$

thus producing a system equation linearized around Ω. Equation 4.7 is a function of three perturbation variables: $\hat{\omega}$, $\hat{\theta}_{on}$, and $\hat{\theta}_{cond}$. This is rewritten as a single second order equation in $\hat{\omega}$ by substituting Equations 4.2, 4.3, and 4.4 into Equation 4.7 and differentiating once to yield

$$\frac{d^2\hat{\omega}}{dt^2} + \frac{1}{J}\left[B + \frac{\partial T_L}{\partial\hat{\omega}} - \left.\frac{\partial\langle T_e \rangle}{\partial\hat{\omega}}\right|_{\Theta_{on},\Theta_{cond}} - K_{on}K_p\Psi\right]\frac{d\hat{\omega}}{dt} - \frac{1}{J}K_{on}K_i\Psi\hat{\omega} = 0, \tag{4.8}$$

where the function, Ψ, is defined as

$$\Psi = \left.\frac{\partial\langle T_e \rangle}{\partial\hat{\theta}_{on}}\right|_{\Omega,\Theta_{cond}} + \frac{K_{cond}}{K_{on}}\left.\frac{\partial\langle T_e \rangle}{\partial\hat{\theta}_{cond}}\right|_{\Omega,\Theta_{on}}. \tag{4.9}$$

Since Equation 4.8 is a second order differential equation, a necessary and sufficient condition for stability is given by

$$B+\frac{\partial T_L}{\partial \hat{\omega}}-\left.\frac{\partial \langle T_e \rangle}{\partial \hat{\omega}}\right|_{\Theta_{on},\Theta_{cond}} - K_{on}K_p\Psi > 0 \text{ and} \tag{4.10}$$

$$K_{on}K_i\Psi < 0. \tag{4.11}$$

4.3 Stable Closed-Loop Operating Regions

At this point, several observations are made to create sufficient conditions for stable gain selection. By restricting operation to turn on angles greater than the turn on angle which produces maximum torque, $\theta_{on,min}$, as given in Equation 4.3, it can be observed from the torque maps in Section 3.6 that decreasing the turn on angle and increasing the conduction angle will result in higher average torque. It would therefore be desirable to have the following conditions met for the control gains:

$$K_{on}, K_P, \text{ and } K_I > 0, \text{ and} \tag{4.12}$$

$$K_{cond} < 0. \tag{4.13}$$

Given these restrictions on the gains, sufficient conditions for stability are:

$$\Psi < 0 \tag{4.14}$$

$$\frac{\partial T_L}{\partial \hat{\omega}} > -B+\left.\frac{\partial \langle T_e \rangle}{\partial \hat{\omega}}\right|_{\Theta_{on},\Theta_{cond}} + K_{on}K_p\Psi \tag{4.15}$$

where $-B$ and $K_{on}K_p\Psi$ are negative by definition. The torque gradient with respect to speed is observed to also be negative by examining the torque maps of Section 3.6. Note that this will only be strictly true in the absence of current limiting which may be necessary at low speed operation. Therefore Equation 4.15 is satisfied by a wide range of system load torque functions even with moderate negative gradients.

For Equation 4.14 to be valid, the torque gradients with respect to $\hat{\theta}_{on}$ and $\hat{\theta}_{cond}$ must be examined. Figure 4.1 shows a portion of one of the torque maps with the equilibrium operating condition boundaries delineated by the hatched lines. The boundaries are established using the assumption that $\Theta_{on} \geq \theta_{on,min}$, and the additional assumption that the steady-state load torque will always be positive. The operating

points are further divided into two regions, 1 and 2, based on the sign of the torque gradient with respect to $\hat{\theta}_{on}$ where

$$\text{Region 1: } \frac{\partial\langle T_e\rangle}{\partial\hat{\theta}_{on}} > 0 \text{ and } \frac{\partial\langle T_e\rangle}{\partial\hat{\theta}_{cond}} > 0 \tag{4.16}$$

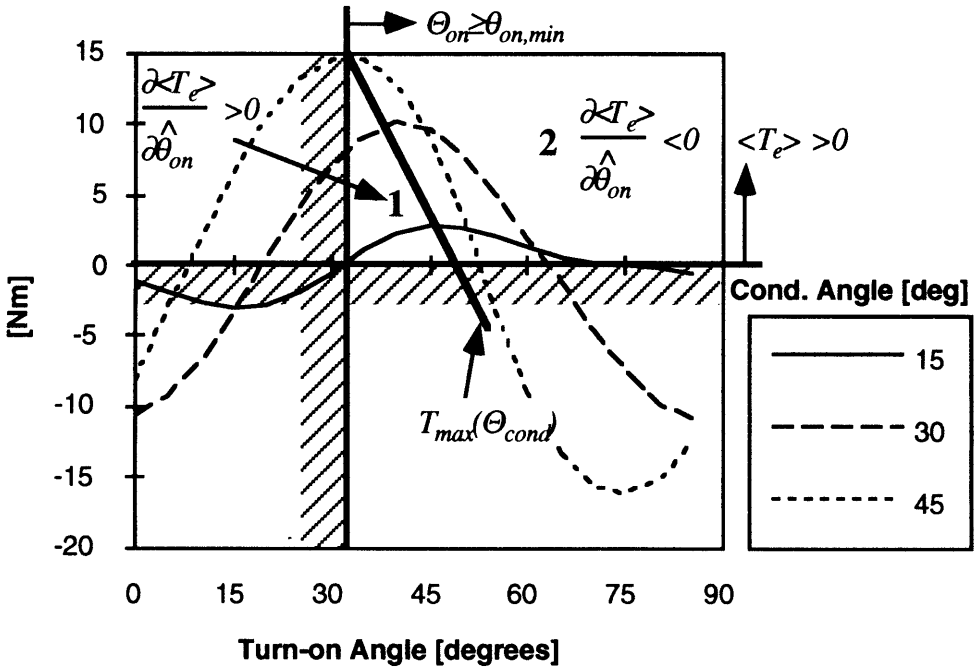

Figure 4.1: Equilibrium operating regions as a function of Θ_{on}, Θ_{cond}, and $\langle T_e\rangle$. Region 1 has positive torque gradients, and Region 2 has negative gradients.

$$\text{Region 2: } \frac{\partial\langle T_e\rangle}{\partial\hat{\theta}_{on}} < 0 \tag{4.17}$$

To insure $\Psi<0$, Region 1 establishes an upper bound on K_{cond}/K_{on} since both gradients are positive. In Region 2 the $\hat{\theta}_{cond}$ gradient can be either sign. A positive gradient is stabilizing, and a negative gradient sets the lower bound. The general gain restrictions for stability are given by

$$-\left.\frac{\frac{\partial\langle T_e\rangle}{\partial\hat{\theta}_{on}}}{\frac{\partial\langle T_e\rangle}{\partial\hat{\theta}_{cond}}}\right|_{\text{Region 2}} < \frac{K_{cond}}{K_{on}} < -\left.\frac{\frac{\partial\langle T_e\rangle}{\partial\hat{\theta}_{on}}}{\frac{\partial\langle T_e\rangle}{\partial\hat{\theta}_{cond}}}\right|_{\text{Region 1}} . \tag{4.18}$$

To determine the particular limits, the worst case partial derivatives were calculated for the candidate VRM. The simulation model was used to calculate the torque gradients using the following estimation form

$$\frac{\partial\langle T_e\rangle}{\partial\hat{\omega}} = \frac{\langle T_e(\Theta_{on},\Theta_{cond},\omega+\Delta\omega)\rangle - \langle T_e(\Theta_{on},\Theta_{cond},\omega-\Delta\omega)\rangle}{2\Delta\omega}. \tag{4.19}$$

The gradient ratios are most severe at the lowest steady-state operating speeds of interest which in this case was chosen to be 2000 rpm. Figure 4.2 shows the calculated partial ratios for Region 1, and Figure 4.3 shows the calculations for Region 2. The resulting bounds are

$$-1.72 < \frac{K_{cond}}{K_{on}} < -1.35. \tag{4.20}$$

The bounds of Equation 4.20 are wider when operating at higher speeds due to the lower gradients. Further examination of Figures 4.2 and 4.3 show that the upper limit occurs at minimum Θ_{on} with low Θ_{cond}, and the lower bound occurs at maximum Θ_{cond} with a higher Θ_{on}. Neither of these conditions are likely operating conditions. For instance when $\Theta_{cond} = 45$ the motor is accelerating and therefore $\Theta_{on} \approx \theta_{on,min}$, and the reverse is also true. Therefore the most limiting conditions will not occur in practical operation of the VRM and the stability boundaries will be wider.

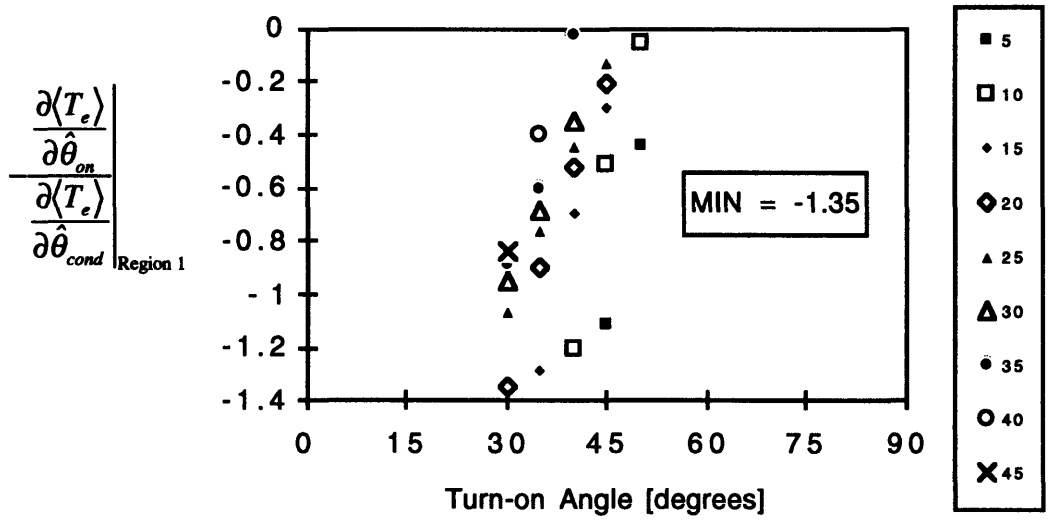

Figure 4.2: Stability limits for region of positive torque gradients at 2000 rpm. The most limiting case is at $\theta_{on,min}$.

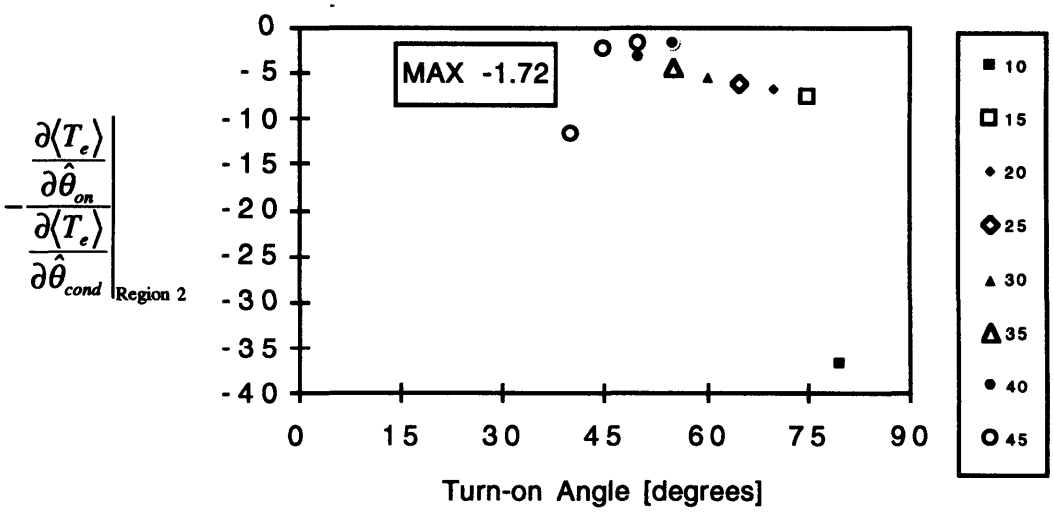

Figure 4.3: Stability limits for region of negative torque gradients at 2000 rpm. The most limiting case is at the maximum Θ_{cond}.

4.4 Implementation Issues

The controller is converted to a discrete time form to operate with the discrete simulation developed in Chapter 3 to produce a stabilized, transient speed simulation of the VRM. After establishing the performance predictions using the VRM simulation, the controller routine is then used for the experimental tests which are discussed in Chapter 6. Keeping in mind the design goal of the controller, several limitations are placed on the operation of the controller to maintain stability for discrete implementation in variable speed operation. Since the stability analysis was conducted based on perturbations near equilibrium, control increments for large setpoint changes are limited to prevent rapid changes in torque production and significant setpoint overshoot. In addition there are practical limits on the control angles which were introduced in the previous section.

The two control angle limits introduced in the previous section (Equations 4.3 and 4.4) are described here in more detail. The turn on angle is restricted to the following range:

$$\theta_{on,\min} \le \theta_{on} < \Theta_{EC}, \tag{4.21}$$

where the lower angle limit is a function of speed. For the candidate motor, the turn on limit was calculated and given by Figure 3.22. For the given controller design which retards the turn on angle during motor acceleration, retarding θ_{on} below this lower limit would result in lower $\langle T_e \rangle$ and therefore unstable operation. The upper limit is artificial representing the end of the current electrical cycle and beginning of the next. In practical applications that limit is never reached due to restrictions on incrementing θ_{on} when $\theta_{cond} = 0$.

The conduction angle is limited to

$$0 \le \theta_{cond} \le \frac{\Theta_{EC}}{2}. \tag{4.22}$$

The lower limit establishes the practical boundary that prevents turning off a phase before turning it on. The upper limit allows approximately equal time for currents to build up and then decay before the next electrical cycle turn on angle is reached.

When either control angle has reached a limit, the incremental control laws defined by Equations 4.3 and 4.4 cannot be satisfied and therefore the controller stability for the non-limited control angle must be reexamined. The partial derivative of torque with

respect to $\hat{\theta}_{cond}$ is zero when operating at the high θ_{cond} limit, therefore Equation 4.9 and the sufficient conditions for stability degenerate to

$$\frac{\partial \langle T_e \rangle}{\partial \hat{\theta}_{on}} < 0 \text{ when } \theta_{cond} = \frac{\Theta_{EC}}{2}. \tag{4.23}$$

Since θ_{cond} will only reach the maximum limit during motor acceleration, θ_{on} will be operating near $\theta_{on,\min}$ where the conditions of Equation 4.23 are always satisfied. When θ_{cond} is limited to zero, torque output for the motor is also zero, and changes in θ_{on} are therefore immaterial. When θ_{on} is equal to its maximum limit, the sign of Ψ cannot be controlled so this operating condition is avoided by choosing gains and initial control angles that will result in θ_{cond} reaching its minimum limit first. When $\theta_{on} = \theta_{on,\min}$ the partial derivatives are not zero because $\theta_{on,\min}$ varies with motor speed. The partial derivative can be represented by the expansion with respect to speed given by

$$\frac{\partial \langle T_e \rangle}{\partial \hat{\theta}_{on}} = \frac{\partial \langle T_e \rangle}{\partial \hat{\omega}} \frac{\partial \hat{\omega}}{\partial \hat{\theta}_{on}}. \tag{4.24}$$

Examination of the partial derivatives calculated from the torque maps and the turn on limits show that this quantity is always positive but smaller in magnitude than the original partial at constant speed used for the gain selections.

In addition to the limits on the control angles, the PI error command is restricted to prevent excessively large changes in the original control angles. The limits are chosen to allow sufficient accelerating and steady-state average torque for the worst case speed setpoint changes. These limits are represented by

$$-\hat{\omega}_{PI,max} \le \hat{\omega}_{PI} \le \hat{\omega}_{PI,max}. \tag{4.25}$$

Additionally, slew rate limiting for large setpoint changes was anticipated based on Torrey's investigation. Drive system experiments were conducted to determine a stable slew rate limit. The slew rate limit restricts the rate of change for θ_{on} and θ_{cond} to an absolute positive or negative limit at each controller update.

Discrete implementation of the controller is accomplished by transforming the incremental control equations to the analogous difference equations. For example, the integral term is replaced by the update rate of the controller, t_s, and is given by

$$\hat{\omega}_{PI_k} = K_P \hat{\omega}_k + K_I \sum_{j=1}^{j=k} \hat{\omega}_j t_s. \tag{4.26}$$

Integrator windup is prevented by disabling the integrator function in Equation 4.2 when $\hat{\omega}_{PI}$ is limited.

Due to the nonlinear relationship between the control angles and equilibrium operation, additional measures are employed to reduce possible oscillatory responses. By the structure of the controller, the initial control angles are related by proportional constants and the error command to the final equilibrium angles. Therefore the equilibrium response will be an oscillation around the new speed setpoint in which the steady-state control angles are based solely on an integrator error that results in proportionally shifted control angles that produce equilibrium torque. To reduce the magnitude of oscillations the integrator is chosen to have slow response.

4.5 Performance Evaluation

Performance of the controller is evaluated in the context of the design goal. The controller was designed to provide stable transient speed control for both large setpoint changes and steady-state operation in the presence of modeling errors. Based on experience with the simulation and the stability limits given by Equations 4.12, 4.13, and 4.20 appropriate gains were selected for the simulation. The control parameters chosen for the transient speed simulation are shown in Table 4.1.

Table 4.1: Controller parameters selected for the VRM simulation.

Parameter	Units	Value
K_{on}	[sec]	0.5
K_{cond}	[sec]	0.5
K_P	[]	0.5
K_I	[1/sec]	-0.75
$\hat{\omega}_{PI,max}$	[rad/sec]	50.0
$\hat{\omega}_{rate\ lim}$	[rad/sec/update]	4.0
t_s	[msec]	4.0
J	[kg.m^2]	0.00708
B	[kg.m^2/sec]	0.000531
C	[Nm]	0.252
T_L	[Nm]	0.0

The gains, speed error command limit, and slew rate limit were selected based on experience with the simulation within the limits prescribed by the above analysis. The inertia and load constants are the same as given for the steady-state simulations in Chapter 3.

The transient simulation is executed by inputting initial conditions, the final speed setpoint, and the controller update rate. Data is collected and stored at regular intervals. Figures 4.4 through 4.7 show the simulation response for a large transient from an initial speed of 2000 rpm to a new setpoint of 3500 rpm. The initial conditions for this transient are shown in Table 4.2. For this particular transient the load torque is set to a viscous plus Coulombic term ($B\omega + C$). The load function is set to zero ($T_L = 0$) The resulting typical speed response shown in Figure 4.4 is asymptotically stable with a settling time of approximately 3.5 seconds measuring from the point when the command signals are no longer limited. It is also observed that the response is heavily damped due primarily to the slow integration term and extremely slow mechanical time constant of the drive system (J/B ~ 10 sec). In Figure 4.5 error command limiting is exhibited due to the large initial setpoint error. The error command limit is set high enough so that the initial control angles plus the control angle gains times the maximum error command will allow for the control angles to be modulated sufficiently to balance the load torque at the new setpoint. The control angle incremental changes are shown in Figures 4.6 and 4.7. Note that the turn on angle immediately tracks the minimum turn on angle limit until the new setpoint is reached. The transient shows that the minimum turn on angle decreases with increasing motor speed. The conduction angle is equal to the maximum limit initially for the same reason, and remains limited due to the remaining negative error command. The error command will always remain negative for this particular transient because the turn angle necessary for equilibrium operation is lower than the initial turn angle supplied for the simulation.

Table 4.2: Simulation parameters for transient controller simulation shown in Figures 4.4 through 4.7.

Parameter	Units	Value
V_b	[Volts]	68.0
$\theta_{on}(0)$	[degrees]	32.0
$\theta_{cond}(0)$	[degrees]	13.5
$\omega(0)$	[rad/sec]	2000
Ω	[rad/sec]	3500

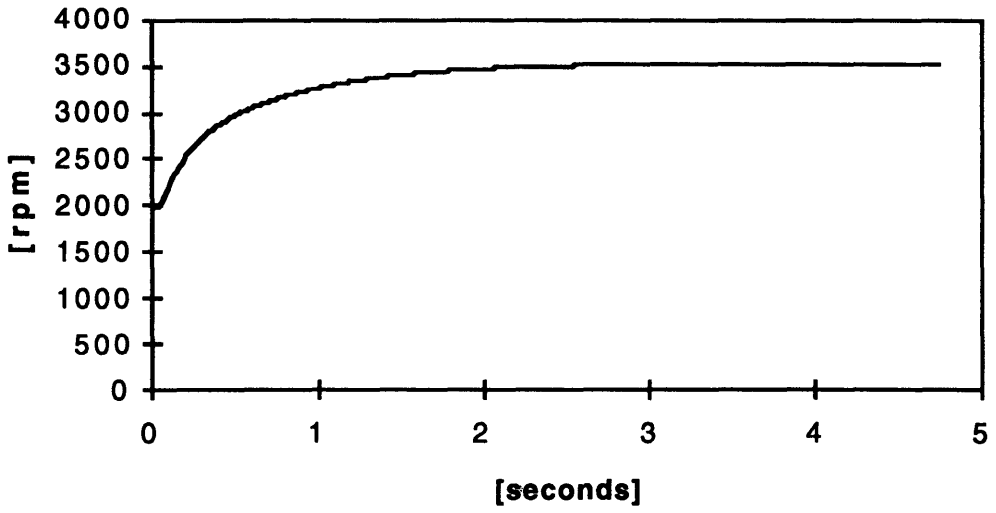

Figure 4.4: VRM simulation transient speed response to setpoint change from 2000 rpm to 3500 rpm.

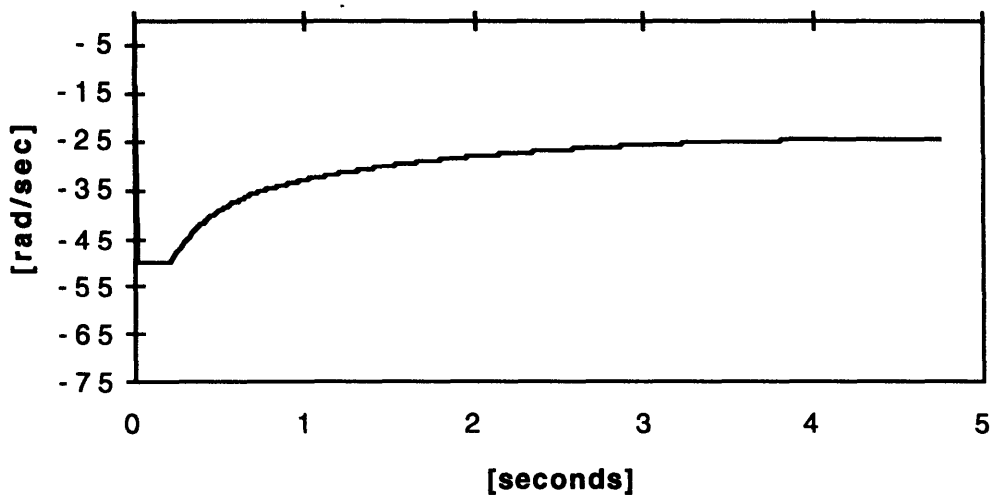

Figure 4.5: VRM simulation PI error command transient response to setpoint change from 2000 rpm to 3500 rpm.

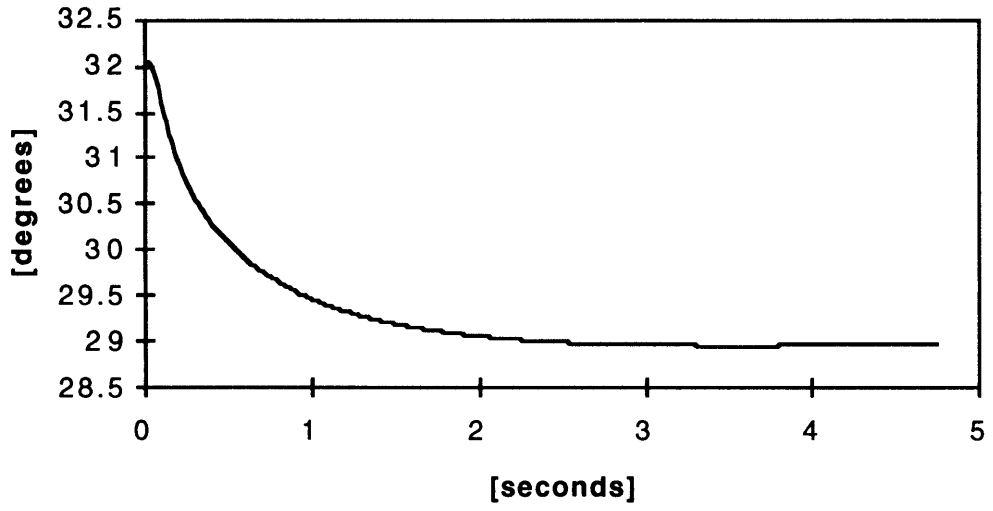

Figure 4.6: VRM simulation turn on angle transient response to setpoint change from 2000 rpm to 3500 rpm.

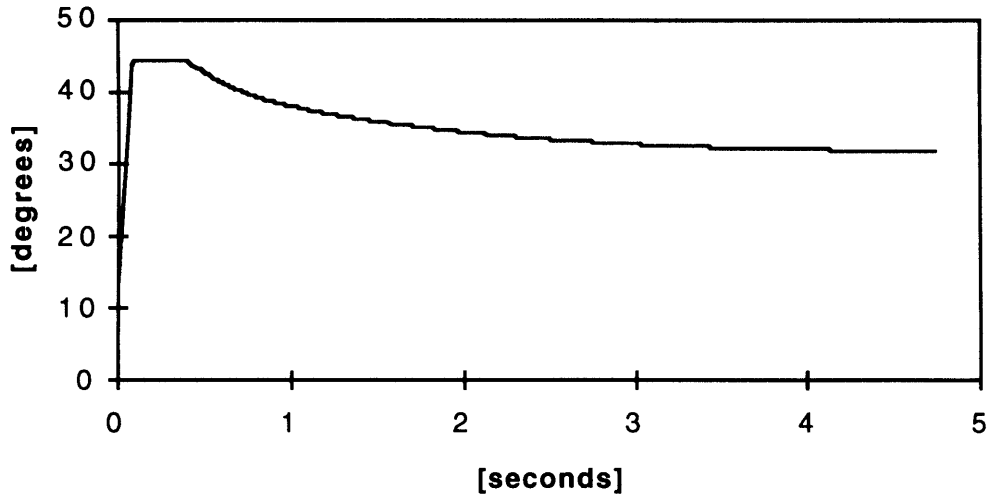

Figure 4.7: VRM simulation conduction angle transient response to setpoint change from 2000 rpm to 3500 rpm.

The transient controller simulation showed a stable damped speed response with gains chosen through experimentation given absolute limits for stability. The controller also operated with limits on the speed error command, the slew rate of the control angles, and ultimate maximum and minimum limits on the control angles. The controller performance is revisited in the discussion of the experimental results in Chapter 6. It will be shown there that the operating characteristics of the experimental drive system dictated different restrictions on the controller action. The controller simulation are run again with the new restrictions for direct comparison to the experiments in Chapter 6.

4.6 Summary

The development of a speed controller was presented in this chapter. Stability analysis was conducted by developing a linearized system equation around an equilibrium speed. The general results were applied to the experimental VRM model to select gains for the control parameters and compensation terms. Finally the controller was implemented with restrictions into the VRM simulation, and the transient performance was demonstrated.

The stability analysis made use of the time scale separation between the mechanical and electrical time constants of the VRM drive. Since drive systems typically have large inertias, the mechanical time constants are many orders of magnitude higher than the time constants associated with the electrical variations, which occur within one electrical cycle of machine rotation. For a 6-4 VRM rotating at even just 2000 rpm the entire cycle over which the variable torque production repeats is only 7.5 milliseconds. For the candidate motor, the mechanical time constant is 13.3 seconds. This separation allows the linearized analysis to proceed using time-average torque.

The general linearized solution is potentially unstable. Depending on the sign of the average torque gradients with respect to the control angles and speed, the drive system will be unstable in open-loop. By restricting the control angles as a function of speed and within practical operating limits, a single set of control gains are chosen. The VRM is capable of conducting large transients over the full range of desired speed using this control form because the restrictions are set to ensure a sufficient torque production range.

The simulation shows that this design methodology is successful when integrated with the electromagnetic model of the previous chapter. The functioning of the control

angle limits and dynamic compensation resulted in an damped transient response. Though the compensation speed is relatively slow, this is sufficient for the research purposes. In the next chapter, the observer is developed, and it will be shown how the observer dynamics are designed to be significantly faster than the controller so that observer errors die away before control action is affected.

Chapter 5

THE MOTION ESTIMATOR

5.1 Introduction

The dynamic VRM model and controller developed in the previous two chapters form the structure around which the motion estimator is designed and evaluated. This chapter presents the design of the estimator, or observer, as it is often referred to in the controls literature. The observer structure introduced in Chapter 2 is used to develop the optimal estimation solution for a steady-state Kalman filter. Using the experimental VRM parameters, stable gains are calculated for the observer. Implementation issues are discussed with respect to incorporating the observer into the transient speed simulation, and then the simulation results are presented.

Section 2.5 explained how an observer structure can be applied to VRM control problems. Instead of measuring all the plant states required for control, these states are estimated by developing a model of the plant, the estimator, and correcting the estimator state errors based on output measurements from the actual plant. Output measurements are compared to the corresponding estimator outputs, scaled by an estimator gain vector, and then fed back into the estimator state model as shown in Figure 2.7. Stable gains must be selected for the innovation terms as described in Section 2.5. There are well-developed theories for both full-state and reduced-order observer design methods as described in Luenberger [11] and Franklin, Powell, et. al. [12] and [13]. Since the system model for the VRM observer application is multi-input multi-output, or MIMO, the steady-state Kalman filter design method was selected which provides a systematic process for uniquely specifying the gains as opposed to pole-placement techniques which are typically more appropriate for single-input single-output, or SISO, applications. This method optimizes the estimator gains to minimize the effects of process and measurement noise. The Kalman filter assumes that the magnitude of the

process and measurement noise is known and is distributed in a Gaussian manner with no time correlation.

For the VRM application the plant output that is used to drive the observer innovations is the rotor position. In this implementation the position state is not measured directly, but is derived from phase current measurements. The nonlinear inversion is obtained by measuring phase currents and using the VRM model developed in the previous chapters to determine the angle corresponding to the current. For a given current and phase there are two possible solution for the rotor position within each electrical cycle, so reasonable assumptions are made in order to eliminate one solution. Furthermore, the observer is designed to interpolate without the innovation terms between each current measurement, thus the observer gains must be designed by taking into account the N state transitions between each measurement.

Having calculated the appropriate gains, the observer performance is evaluated through the transient simulation. Since the simulated plant and observer models are identical, the observer stability is demonstrated by imposing initial state errors for position and speed. Thus stability is shown when the errors decay asymptotically. The resulting transients correspond to the expected decay rates and steady state errors for the given observer design. Their adequacy for transient speed control are also discussed.

5.2 Steady-state Kalman Filter Design

The Kalman filter design method follows directly from modern observer and optimal control theories. The full state, or identity, observer seeks to duplicate the plant state dynamics with a linear state-space model so that all the states are available for feedback control purposes. In general the open-loop plant model may be either stable or unstable similar to the real plant. Errors between the plant and observer states due to initial condition differences, disturbances, or incomplete modeling of the dynamic behavior may never decay and in fact increase unbounded at a rate determined by the eigenvalues of the original plant and plant model. The state estimate errors are driven asymptotically to zero using a linear function of the difference between the available plant outputs and the corresponding estimated outputs. The gain matrix for the innovation terms can be arbitrarily selected to cause the observer to converge to the plant states with specified decay characteristics. The only stipulation being that the plant states are observable.

Various methods have been developed for selecting the observer gains. Optimal control theory bases the gain choice on maximizing, or minimizing, an artificial 'cost' function. The design of the cost, or objective, function is usually tailored to performance characteristics of interest. The Kalman filter design is a particular objective function structure which assumes that the plant being followed by the observer has inaccuracies that can be modeled as noise in both the process and measurement stage. The theory is significantly simplified for linear system representations where the noise is assumed to have no time correlation. The optimal observer gains are then chosen by minimizing the state errors in the presence of expected process and measurement noise levels. Due to initial condition differences, the general solution is time-varying, but the solution which minimizes steady-state errors is obtainable through a recursive process. It is the limit of the time-varying solution as $t \rightarrow \infty$. The following development of the steady-state Kalman filter identifies the plant and observer structures, and then applies this structure to the VRM under consideration.

To develop the discrete-time estimator, the plant is represented using the conventional state-space form

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{F}\mathbf{x}(t) + \mathbf{G}u(t), \tag{5.1}$$

The output vector is given by

$$y(t) = \mathbf{H}\mathbf{x}(t), \tag{5.2}$$

where the measurements do not depend on the input vector (i.e. $\mathbf{J}u(t) = 0$).

To keep the development more generally applicable the system representation is converted to the discrete time domain. This is appropriate since both the simulation and the experimental observers are implemented using a digital computer. Of course, a small enough update rate may be selected so that the mechanical time constants of interest may be adequately represented in continuous time. The reduced second-order system can be represented in the discrete time domain by

$$\mathbf{x}(k+1) = \Phi\mathbf{x}(k) + \Gamma u(k), \tag{5.3}$$

$$y(k) = \mathbf{H}\mathbf{x}(k), \tag{5.4}$$

assuming an equivalent zero-order hold on the input vector, $u(t)$, over the sampling interval.

This second-order discrete representation is used to pose an estimator of the form

$$\bar{\mathbf{x}}(k+1) = \Phi\bar{\mathbf{x}}(k) + \Gamma u(k) + \mathbf{L}_p\big(y(k) - \bar{y}(k)\big), \tag{5.5}$$

$$\bar{y}(k) = \mathbf{H}\bar{\mathbf{x}}(k), \tag{5.6}$$

where $\mathbf{L}_p$ is the estimator, or innovation, gain matrix which corrects the predicted state transitions based on current measurement errors. This is called the predictor observer form.

The gain selection is accomplished using steady-state optimal control methods to produce the Kalman filter. Assume the real plant can be represented by a discrete system of the form

$$\mathbf{x}(k+1) = \Phi\mathbf{x}(k) + \Gamma u(k) + \Gamma_1 w(k), \text{ and} \tag{5.7}$$

$$y(k) = \mathbf{H}\mathbf{x}(k) + v(k), \tag{5.8}$$

where $w(k)$ and $v(k)$ represent the process and measurement noise respectively. It is assumed that the noise vectors have the form of white noise (i.e. a frequency spectrum that is both constant and infinite [13]). The process and measurement noise covariances are defined by

$$\mathbf{R}_w \equiv \varepsilon\big\{\mathbf{w}(k)\mathbf{w}^T(k)\big\}, \text{ and} \tag{5.9}$$

$$\mathbf{R}_v \equiv \varepsilon\big\{\mathbf{v}(k)\mathbf{v}^T(k)\big\}. \tag{5.10}$$

The steady-state optimal solution which minimizes the state estimate errors given the assumed forms for process and measurement noise is then given by

$$\mathbf{L}_p = \Phi\mathbf{M}\mathbf{H}^T\big(\mathbf{H}\mathbf{M}\mathbf{H}^T + \mathbf{R}_v\big)^{-1} \tag{5.11}$$

where $\mathbf{M}$ represents the eigenvectors resulting from the steady-state solution of the Riccati equation. The solution for $\mathbf{M}$ depends on Φ, $\mathbf{H}$, $\mathbf{R}_v$, $\mathbf{R}_w$, and Γ_1, and is found by iteratively solving the algebraic form of the Riccati equation given by

$$\mathbf{M} = \Phi\Big[\mathbf{M} - \mathbf{M}\mathbf{H}^T\big[R_v + \mathbf{H}\mathbf{M}\mathbf{H}^T\big]^{-1}\mathbf{H}\mathbf{M}\Big]\Phi^T + \Gamma_1 R_w \Gamma_1^T. \tag{5.12}$$

These constant estimator gains are also referred to as the Discrete Linear Quadratic Estimator (DLQE) solution which implies a Gaussian noise distribution for the process and measurements. For more detailed development of the DLQE see Franklin, Powell, et. al. [12, 13].

The above development is now applied to the VRM drive system. As proposed in Chapter 4, the time scale separation between the mechanical and electrical time constants of the VRM allows the fifth-order model of Equations 3.11 through 3.13 to be represented by

$$\frac{d}{dt}\mathbf{x}(t)=\begin{bmatrix}0 & 1\\ 0 & -\frac{B}{J}\end{bmatrix}\mathbf{x}(t)+\begin{bmatrix}0\\ \frac{1}{J}\end{bmatrix}\left(\langle T_e(t)\rangle - T_L(t) - C\right), \tag{5.13}$$

$$\mathbf{x}(t)=\begin{bmatrix}\theta\\ \omega\end{bmatrix}, \tag{5.14}$$

$$y(t)=[1 \quad 0]\mathbf{x}(t), \tag{5.15}$$

where the electrical dynamics are implicit in the definition of the average torque, $\langle T_e(t)\rangle$.

The discrete time state-space matrices are calculated by integrating Equation 5.13 using the method of variation of parameters [13]. The discrete system is given by

$$\mathbf{x}(n+1)=\mathbf{A}\mathbf{x}(n)+\mathbf{B}u(n), \tag{5.16}$$

$$y(n)=\mathrm{X}\mathbf{x}(n), \tag{5.17}$$

$$\mathbf{A}=\begin{bmatrix}1 & \frac{J}{B}\left(1-e^{-\frac{B\Delta t_n}{J}}\right)\\ 0 & e^{-\frac{B\Delta t_n}{J}}\end{bmatrix}, \tag{5.18}$$

$$\mathbf{B}=\int_{t_n}^{t_{n+1}}\begin{bmatrix}\frac{1}{B}\left(1-e^{-\frac{B(\sigma-t_n)}{J}}\right)\\ \frac{1}{J}e^{-\frac{B(\sigma-t_n)}{J}}\end{bmatrix}d\sigma=\begin{bmatrix}\frac{1}{B}\left(\Delta t_n-\frac{J}{B}\left(1-e^{-\frac{B(\Delta t_n)}{J}}\right)\right)\\ \frac{1}{B}\left(1-e^{-\frac{B(\Delta t_n)}{J}}\right)\end{bmatrix}, \tag{5.19}$$

$$\mathrm{X}=[1 \quad 0], \text{ and} \tag{5.20}$$

$$\mathbf{x}(n)=\begin{bmatrix}\theta_n\\ \omega_n\end{bmatrix}, \tag{5.21}$$

$$\Delta t_n = t_{n+1}-t_n. \tag{5.22}$$

where different matrix names (A, B, and X) have been used to differentiate these quantities from the earlier general presentation (Φ, Γ, and H). Also a different discrete time index, *n*, is being used to avoid confusion.

The observer design is further complicated due to the sparse measurement sampling. As described in the previous section, the observer innovations calculated from the VRM output measurements only occur once per phase per electrical cycle with the observer model interpolating between each measurement. By assuming that there are N observer model steps between each measurement of the output vector, *y(n)*, then the observer model can be described as 2xN order system given by

$$\begin{aligned}
\bar{\mathbf{x}}(n+1) &= \mathbf{A}\bar{\mathbf{x}}(n) + \mathbf{B}u(n) + \mathbf{K}_p\big(y(n) - \bar{y}(n)\big) \\
\bar{\mathbf{x}}(n+2) &= \mathbf{A}\bar{\mathbf{x}}(n+1) + \mathbf{B}u(n+1) + 0 \\
&\vdots \\
\bar{\mathbf{x}}(n+N) &= \mathbf{A}\bar{\mathbf{x}}(n+N-1) + \mathbf{B}u(n+N-1) + 0
\end{aligned} \qquad (5.23)$$

where $\mathbf{K}_p$ is the VRM observer gain matrix. Figure 5.1 is a schematic representation of the relative sample rates of the measurements and observer model interpolations.

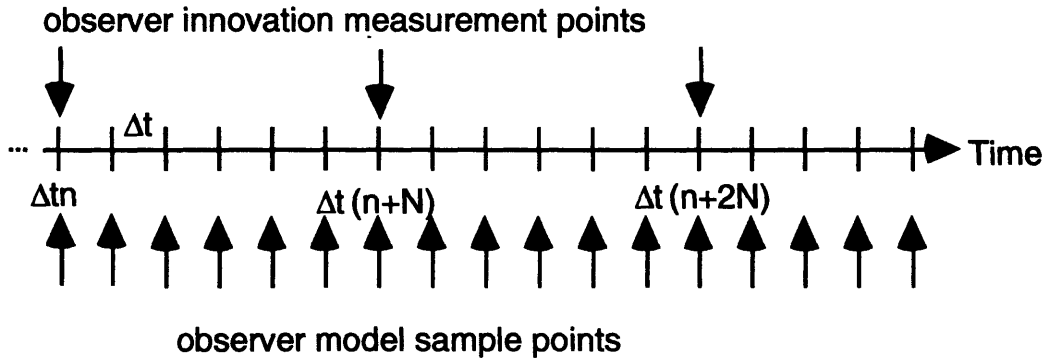

Figure 5.1: Schematic representation of the sample rates for the observer model calculations and measurement innovation inputs.

For the purpose of selecting the estimator gains, this model is reduced to a single second-order system by

$$\bar{\mathbf{x}}(m+1) = \mathbf{A}^N\bar{\mathbf{x}}(m) + \big(\mathbf{A}^{N-1} + \mathbf{A}^{N-2} + \ldots + \mathbf{I}\big)\mathbf{B}u(m) + \mathbf{A}^{N-1}\mathbf{K}_p\big(y(m) - \bar{y}(m)\big), \qquad (5.24)$$

$$\Delta t_m = N\Delta t_n, \qquad (5.25)$$

with step intervals Δt_m instead of Δt_n. It is assumed that $u(m)$ represents the average input over the N observer model updates between each measurement. This is reasonable because assuming that the mechanical dynamics are slow compared to the time between observer measurements. This is equivalent to the previously stated assumption that the mechanical dynamics are slow when compared to the electrical dynamics. The system which is used to develop the observer gain matrix is then

$$\bar{\mathbf{x}}(m+1) = \mathbf{A}^N \bar{\mathbf{x}}(m) + \left(\mathbf{A}^{N-1} + \mathbf{A}^{N-2} + \ldots + \mathbf{I}\right)\left(\mathbf{B}u(m) + \mathbf{B}_1 w(m)\right) \tag{5.26}$$

where $\mathbf{B}_1$ is the assumed form of the VRM process noise distribution matrix.

Since output measurements only occur every N steps, the VRM output vector, X, is still applicable. The DLQE solution is then calculated by making the following substitutions into Equations 5.11 and 5.12

$$\begin{aligned} \Phi &\leftarrow \mathbf{A}^N \\ \Gamma_1 &\leftarrow \left(\mathbf{A}^{N-1} + \mathbf{A}^{N-2} + \ldots + \mathbf{I}\right)\mathbf{B}_1 \\ \mathbf{H} &\leftarrow \mathbf{X} \\ R_{v,w} &\leftarrow R_{v,w} \\ \mathbf{L}_p &\leftarrow \mathbf{K}_p \\ k &\leftarrow m \end{aligned} \tag{5.27}$$

5.3 Stability Analysis

To apply the Kalman filter to the experimental VRM model, the discrete observer parameters and noise covariances are calculated to be those given in Table 5.1. The motor parameters are known from Chapter 3,but selection of the noise covariances require some explanation. The measurement covariance was chosen to be the square of the LSB precision of the sensing system. Since there is no well-defined estimate of the average torque modeling accuracy, a rough estimate of 10% of the maximum available torque was employed as a worst case based on the flux-linkage and torque measurements discussed in Chapter 3.

The performance of the observer is characterized by the behavior of the state errors given by

$$\tilde{\mathbf{x}} = \mathbf{x} - \bar{\mathbf{x}}, \tag{5.28}$$

$$\tilde{\mathbf{x}}(k+1) = \left[\Phi - \mathbf{L}_p\mathbf{H}\right]\tilde{\mathbf{x}}(k). \tag{5.29}$$

If all the eigenvalues of Equation 5.29 are within the unit circle of the z-plane, the observer will be asymptotically stable. The rate of decay will depend on the $\mathbf{L}_p$ determined from the Kalman filter design above. Ideally verification with the VRM simulation and experimental drive will show that the state errors decay within times that are short compared to the mechanical dynamics so that transient observer errors will not substantially affect the speed controller regulation precision.

Given the noise estimates in Table 5.1 the closed-loop poles of the system given by Equation 5.24 still depend on the number of steps, N, between innovations. Since a measurement is made each time a phase is turned on, the time between innovations corresponds to a rotor position of 30 degrees, or one twelfth of a revolution. For the experimental VRM, with a top speed of 10 krpm and a digital cycle time of 250 μsec, the resulting range for N is $N \geq 2$. Figure 5.2 shows a plot of the closed loop poles from the Kalman filter design as a function of N. The closed loop observer poles are given by the roots of the characteristic equation

$$\left|z\mathbf{I} - \Phi + \mathbf{L}_p\mathbf{H}\right| = 0. \tag{5.30}$$

Since a constant set of gains is desired for the VRM, it is necessary to choose a gain that will only be optimal at N and time interval, Δt. Even if N was held constant over the VRM speed range, the time interval would then change and a similar issue would be involved. Figure 5.3 shows the closed pole locations as N varies after having fixed the Kalman filter gains for the N=10 solution. From this it is concluded that the observer will be stable for all reasonable values of N.

Since the sample rate varies with N based on VRM speed, the corresponding decay rates cannot easily be compared in the z-plane. Figure 5.4 shows the corresponding continuous time poles which show how the error decay rates vary with N given the Kalman filter design. The z-plane poles are mapped to the s-plane using the relation

$$s_{1,2} = \frac{\ln z_{1,2}}{T} \tag{5.31}$$

where the sample period is $T = N\Delta t$. The s-plane pole plot predicts settling times between 40 and 50 msec using the approximation, 4.6/Re{s}.

Table 5.1: State-space matrices calculated for discrete VRM observer.

Matrix	Coefficients
Φ	$\begin{bmatrix} 1.0 & 0.0002 \\ 0.0 & 1.0 \end{bmatrix}$
Γ	$\begin{bmatrix} 0.0 \\ 0.035 \end{bmatrix}$
$\mathbf{L}_p$	$\begin{bmatrix} 0.37 \\ 32 \end{bmatrix}$
$\mathbf{R}_w$	1.5e-5 $[deg]^2$
$\mathbf{R}_v$	0.64 $[Nm]^2$
poles z1, z2 (N=10)	$0.78 \pm 0.18i$

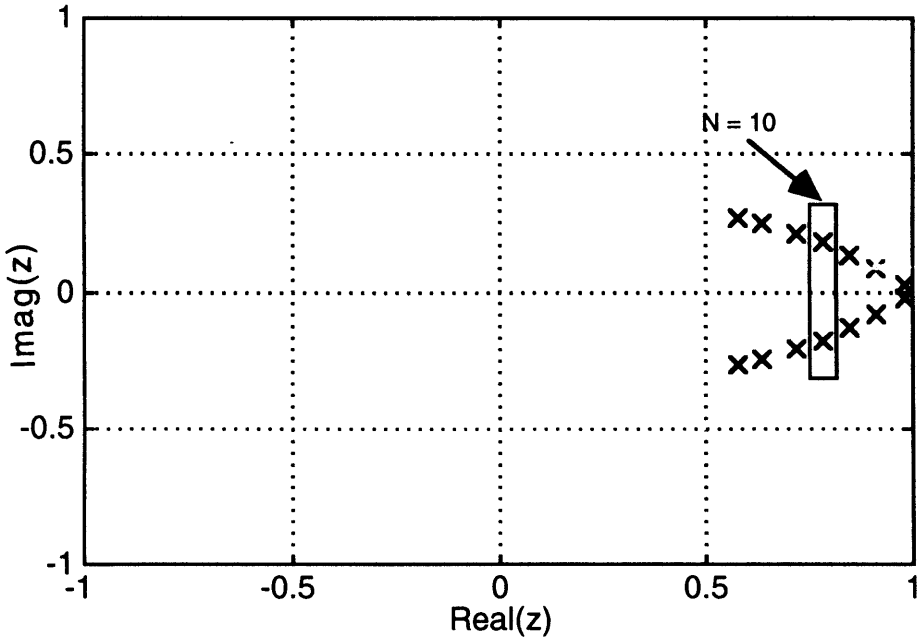

Figure 5.2: Closed-loop observer poles in the z-plane as a function of N (the number of steps between output vector innovations) for $2 \leq N \leq 20$.

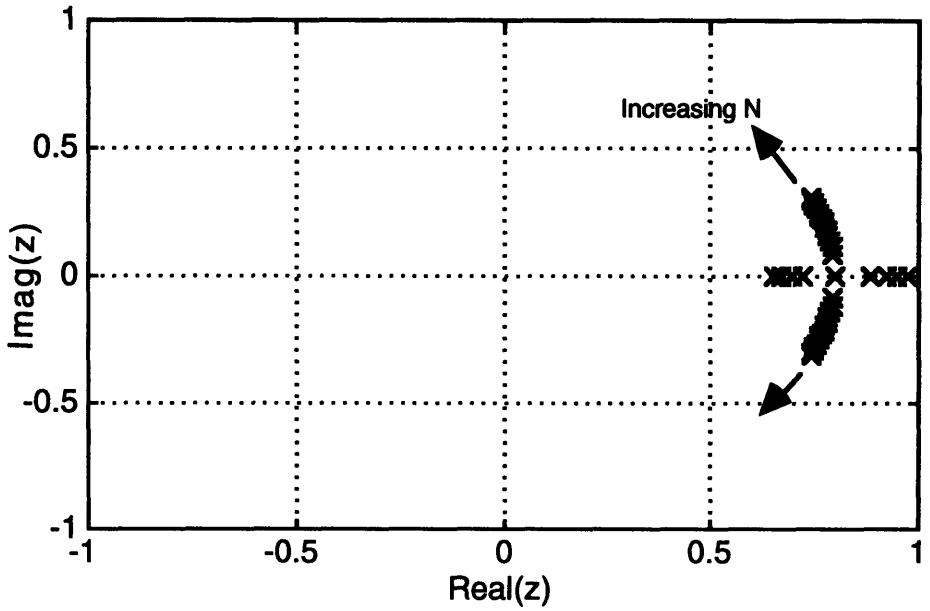

Figure 5.3: Closed-loop observer poles in the z-plane as a function of the number of interpolating steps, N, given fixed Kalman filter gains designed for N = 10.

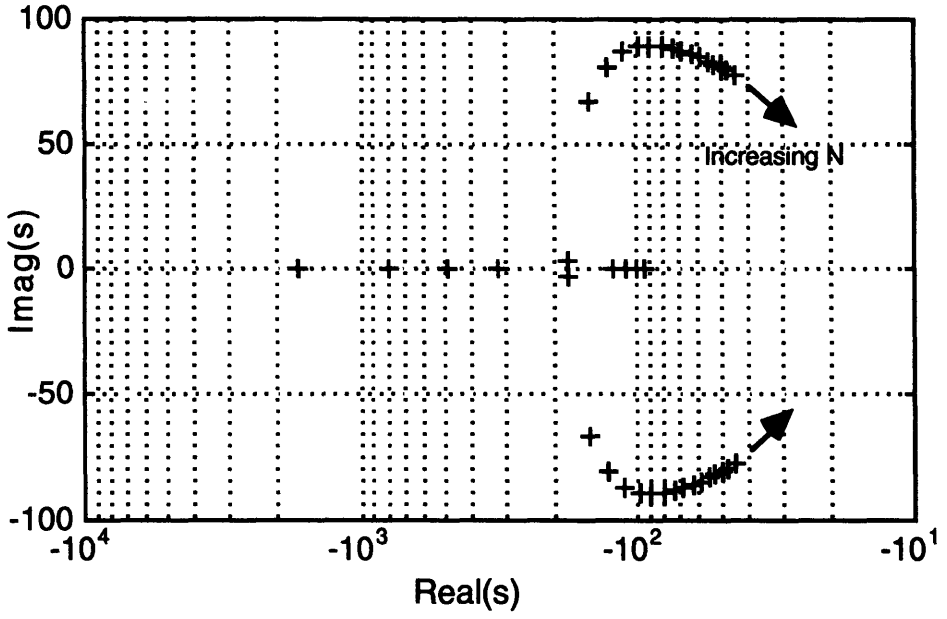

Figure 5.4: Continuous time mapping of the discrete closed-loop observer poles of Figure 5.3 into the s-plane. Predicts a settling time of approximately 40 to 50 msec.

5.4 Implementation Without Position Sensing

Up to this point in the development, it has been assumed that the output vector, $y(k)$, is available to the observer. From Equations 5.4, 5.21, and 5.20 the output vector is simply the instantaneous rotor position. Since the objective is to eliminate the position sensor, the output vector is not directly available to the observer model. The position is inferred from measuring the current in each phase a short time after turning it on. Since there is a direct relationship between the measured current and the instantaneous rotor position, reconstruction of the output vector is accomplished by determining the position - current relation for the given supply voltage and time elapsed since turning on the phase.

Equation 2.1 proposed a nonlinear flux linkage model that depends on rotor position and phase current. Stated generically, this model is given by

$$\lambda = f(i, \theta). \tag{5.32}$$

This equation is not directly invertable to solve for the rotor position because the dependent variable, θ, is not separable, so it is necessary to search for the position roots given the flux linkage and current to numerically determine the inverse relationship, $f^{-1}(i, \lambda)$. The current is directly measured, and the flux linkage is calculated assuming that the flux is zero when the phase is turned on so that

$$\lambda \approx V_b \Delta t_{obs}, \tag{5.33}$$

where Δt_{obs} is the time delay after turning on the phase when the current is measured. For short time delays the resistive drop due to current is negligible compared to the supply voltage, V_b, and is therefore neglected. Since $f(i, \theta)$ is monotonic with respect to θ, the position can then be found by a searching algorithm that finds successively smaller bounds on θ which the solution must be between. The estimated rotor position resulting from this iterative procedure can then be represented by

$$\theta = f^{-1}(I, V_b \Delta t_{obs}) \tag{5.34}$$

where I is the measured current. Figure 5.5 shows the resulting relationship using the VRM simulation for a given supply voltage and measurement delay from alignment to maximum misalignment. Equation 5.34 has two solutions which are symmetric about alignment between the rotor and excited pole faces. The observer therefore incorporates a rule for discriminating which solution is correct. A simple method is to pre-calculate the observer rotor position without the innovation term and select the solution with the

minimum error. If the choice turns out to be incorrect, the model and plant states will eventually diverge enough to become self-correcting. This will be discussed further in the context of the experimental VRM results.

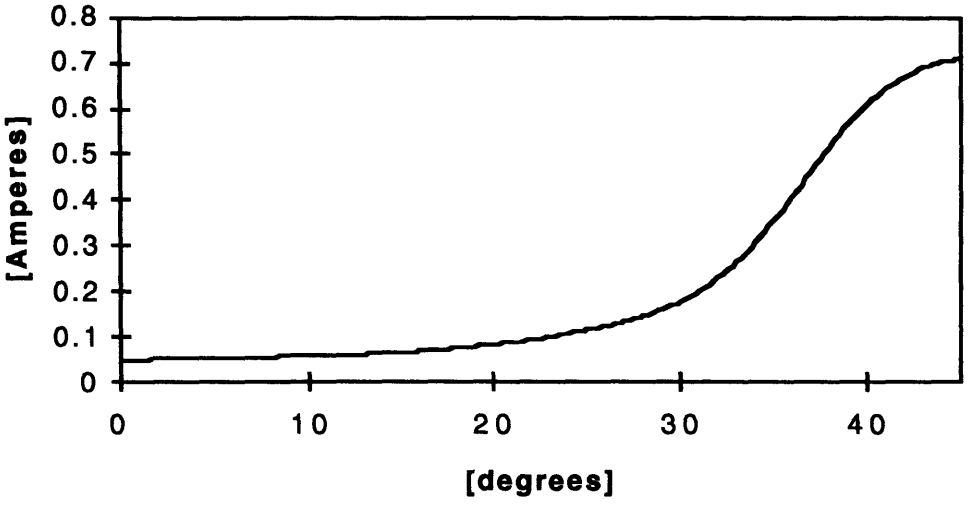

Figure 5.5: Rotor position versus current for $V_b = 30V$ and $\Delta t_{obs} = 83$ μsec.

5.5 Performance

The performance of the estimator is evaluated by incorporating the observer model into the transient simulation. Since both the estimator and the simulated plant are based upon the same dynamic model, errors must be injected in order to evaluate the estimator's stability and disturbance rejection capabilities. A simple way to capture this is to set up the observer model with initial condition errors.

Figures 5.6, 5.7, and 5.8 show the response of the observer to an initial error of 5 degrees and 200 rpm. The simulation parameters are given in Table 5.2. The step shifts in the observer occur each time a phase commutates, and a new current measurement is made. Between output measurements the observer model interpolates using the same model as the simulated plant. The position error correction at each innovation in Figure 5.8 corresponds directly to the observer position gain term, 37%, from Table 5.1. The steady-state error variation is due to a combination of simulating the observer open-loop from the speed control loop, and the granularity of the current measurement times. In

the simulation the measurements are asynchronous with respect to the simulation updates which are calculated at regular $\Delta\theta$ intervals. All such errors, though, remain bounded. The settling time is approximately 30 msec which agrees with the s-plane pole predictions given in Figure 5.4. This is 50 to 100 times faster than the mechanical dynamic response as demonstrated in Figures 4.4 through 4.7.

Table 5.2: Simulation parameters for the off-line observer error decay shown in Figures 5.6 through 5.8.

Parameter	Units	Value
V_b	[Volts]	68.0
$\theta_{on}(0)$	[degrees]	29.0
$\tilde{\theta}(0)$	[degrees]	5.0
$\tilde{\omega}(0)$	[rpm]	200.0
$\theta_{cond}(0)$	[degrees]	15.5
$\omega(0)$	[rad/sec]	3500
Ω	[rad/sec]	3500

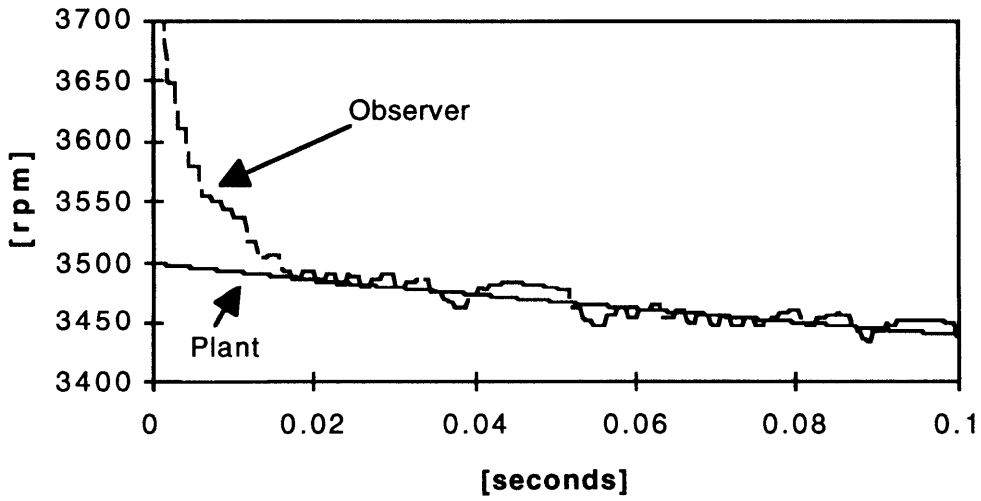

Figure 5.6: Off-line observer transient speed simulation showing both the simulation plant and estimated motor speeds.

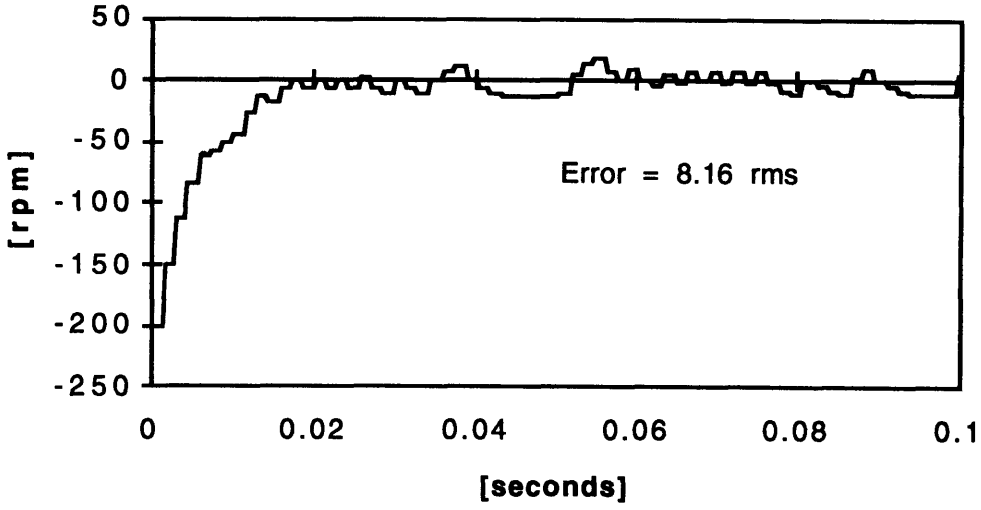

Figure 5.7: Off-line observer transient speed simulation showing estimated speed error (plant - observer).

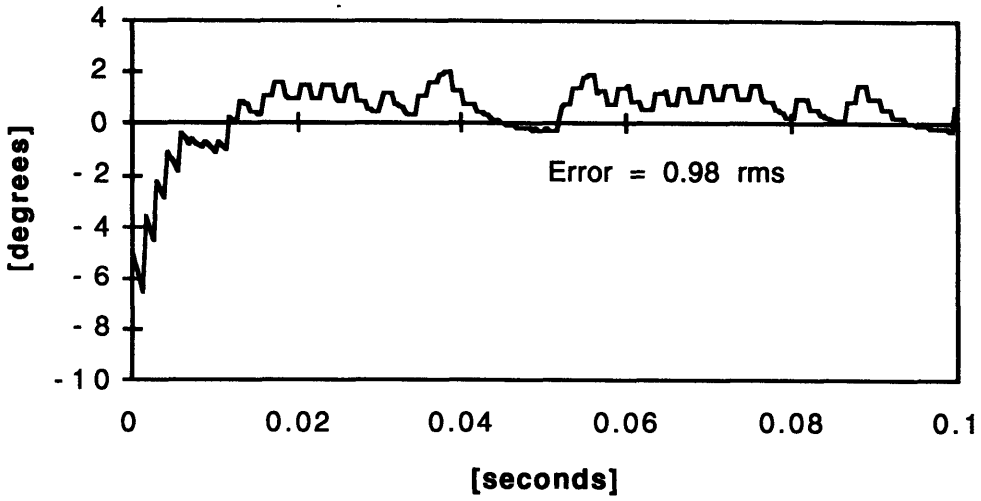

Figure 5.8: Off-line observer transient speed simulation showing estimated rotor position error (plant - observer).

To examine the effects of the observer on the control loop, the imposed state error simulation is repeated with the 'on-line' observer which provides the position and

speed feedback for the controller algorithm. Table 5.3 lists the parameters for this simulation, and Figures 5.9, 5.10, and 5.11 show the resulting transient behavior. As predicted by Eigenvalue Separation Theorem, the decay rate of the observer errors is the same as the off-line observer and therefore independent of the control loop.

Table 5.3: Simulation parameters for the on-line observer error decay shown in Figures 5.9 through 5.11.

Parameter	Units	Value
V_b	[Volts]	68.0
$\theta_{on}(0)$	[degrees]	29.0
$\theta_{cond}(0)$	[degrees]	15.5
$\tilde{\theta}(0)$	[degrees]	5.0
$\tilde{\omega}(0)$	[rpm]	200.0
$\omega(0)$	[rad/sec]	3500
Ω	[rad/sec]	3500

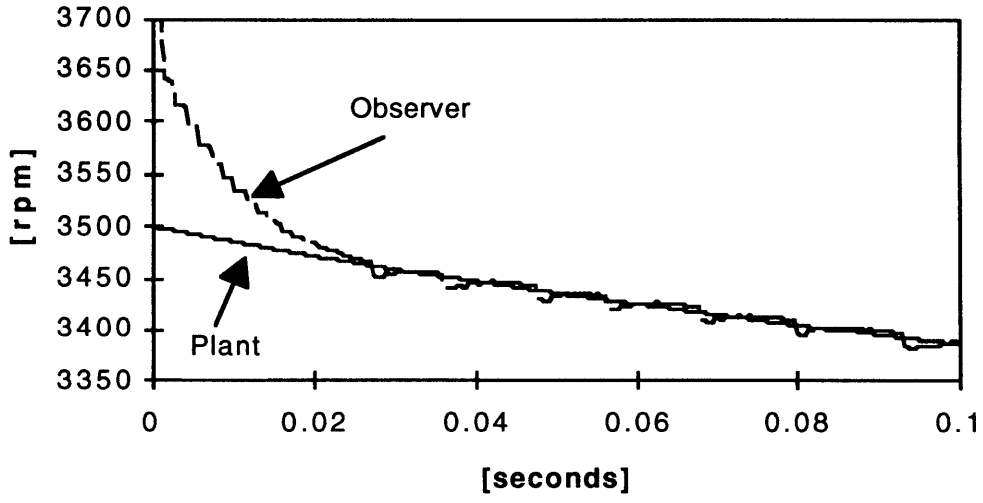

Figure 5.9: On-line observer transient speed simulation showing both the simulation plant and estimated motor speeds.

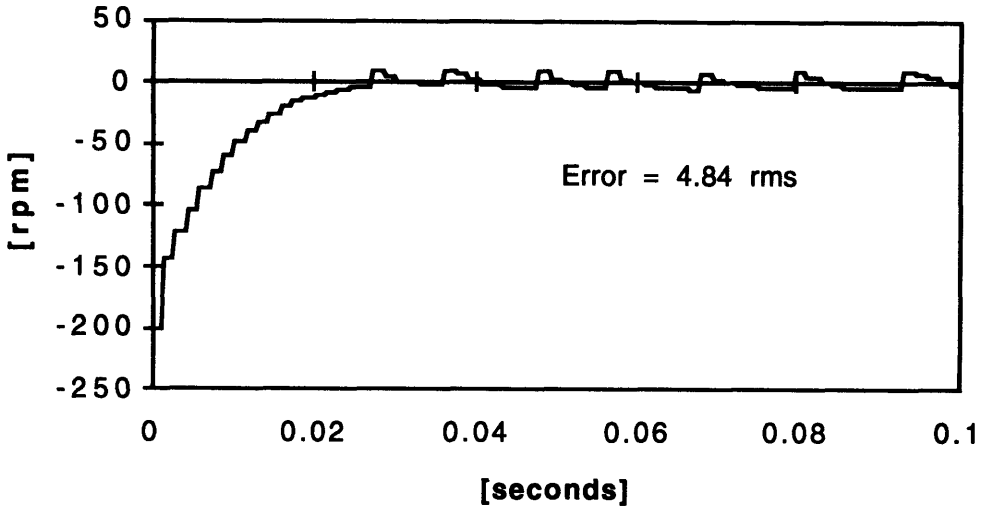

Figure 5.10: On-line observer transient speed simulation showing estimated speed error (plant - observer).

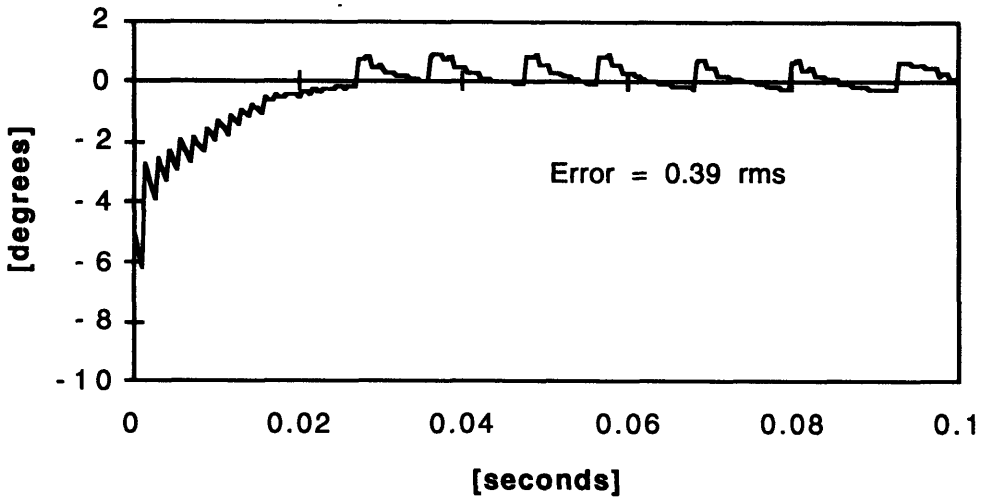

Figure 5.11: On-line observer transient speed simulation showing estimated rotor position error (plant - observer).

As a final demonstration, Table 5.4 lists the simulation parameters for a large speed setpoint change with the on-line observer. Figure 5.12 shows the speed response with

the observer feedback compared directly to the response with mechanical-state plant feedback that was shown in Figure 4.4. As above, the controller response rate is not affected by the addition of the observer dynamic loop. The minor difference visible in Figure 5.12 is due to the different torque that is produced by commutating based on the observer position state rather than the actual motor position.

Table 5.4: Simulation parameters for the speed setpoint change with on-line observer feedback shown in Figure 5.12.

Parameter	Units	Value
V_b	[Volts]	68.0
$\theta_{on}(0)$	[degrees]	32.0
$\theta_{cond}(0)$	[degrees]	13.5
$\tilde{\theta}(0)$	[degrees]	0.0
$\tilde{\omega}(0)$	[rpm]	0.0
$\omega(0)$	[rad/sec]	2000
Ω	[rad/sec]	3500

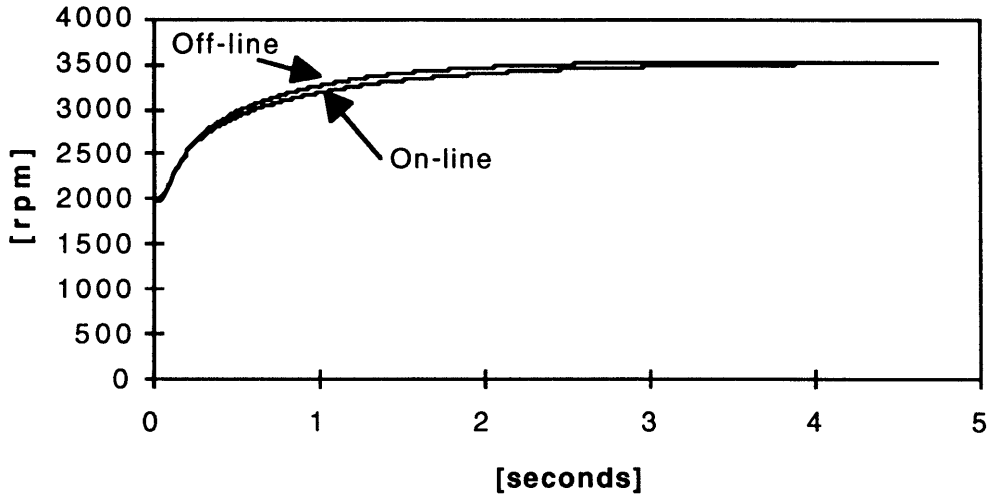

Figure 5.12: Transient speed simulation showing response to a speed setpoint change with observer feedback (on-line) and plant feedback (off-line).

5.6 Summary

This chapter developed the mechanical-state observer and position measurement process for the transient VRM simulation. The observer gains were chosen based on a Kalman observer design methodology which minimizes estimated state errors. Extensions of the second-order observer model were discussed to incorporate the dynamics of observer interpolation between phase current measurements. The 2xNth order model was used to solve for the steady-state Kalman filter gains. The position inversion process from current measurements using the $\lambda - i - \theta$ relationship was also explained. By measuring current at a known flux-linkage shortly after turning on a phase, the position is calculated within an electrical cycle of the VRM. Finally, the stable performance of the observer using the VRM simulation was demonstrated. The observer errors decayed asymptotically to zero with small fluctuations due to the discrete nature of the simulation. Though the decay rate is not input to the Kalman filter design specification, it was calculated to show that errors decay within times that are short when compared to the mechanical dynamics of the VRM.

The decision to use a mechanical-state observer with an optimal estimation gain design is appropriate for the VRM application. In particular, the Kalman filter structure allows consideration of modeling errors in a convenient and reasonable manner. The measurement noise corresponds to the precision of the current sensing system. Gaussian process noise is indicative of imprecise firing of the control angles due to the timing constraints of an experimental VRM drive.

The issue of variable speed operation complicates the observer development. It requires consideration of variable optimal gains. To simplify the implementation, the choice had to be made between using a variable time step or a variable number of interpolations between measurements. Varying the number of interpolations has the advantage of maximizing the available data when a hard time step limit exists. This limit becomes more significant when considering that the simulation must be converted to a real-time observer for the experimental VRM.

The experimental VRM is the subject of the next chapter. The issues of timing, processor speed, and sensing accuracy will become more apparent in the real-time application of this observer. Following development of the hardware and software systems, the real-time performance of the observer will be compared to the simulation showing similar results.

Chapter 6

VRM DRIVE EXPERIMENT

6.1 Introduction

This chapter presents the design of the experimental VRM drive, and then compares the transient behavior of the drive system to the simulations from the previous chapters. The drive employed the variable-reluctance motor from which the $\lambda - i - \theta$ relationship was developed for the simulation program. Furthermore, the controller and observer used in the simulations were also used in the experimental drive. The performance and accuracy of the motor, controller, and observer simulations could then be directly compared to corresponding experiments using this drive system. To fully corroborate the simulation results, the VRM drive was designed for transient speed operation at power levels sufficiently high to demonstrate the stability of the controller and observer in a region of nonlinear flux linkage with limited sampling. The results presented in this chapter corroborate the predictions from the simulations regarding stability and transient performance for the observer. Limitations encountered during the experimentation are also discussed and then incorporated back into the simulation, so direct comparisons could be performed.

The experimental VRM was initially introduced in Section 2.2. After establishing the geometry and winding configuration, a mathematical simulation of the electrical characteristics of the motor was developed in Chapter 3. This simulation was the basis for the design of the controller and observer which were incorporated into the motor simulation and evaluated in Chapters 4 and 5.

The complete drive included the VRM, drive electronics, a controller, and a DC motor acting as a programmable load. Figure 6.1 shows the drive layout. Figures 6.2 and 6.3 are photographs of the VRM drive. The controller was implemented in software on a Texas Instruments TMS320C30 DSP system board installed in a desktop

computer. Keyboard commands were interpreted by a PC-resident interface program and transmitted to the DSP board. The DSP board then issued commands to the inverters, and received feedback signals from Hall effect current sensors, a position encoder, and an over-current protection circuit.

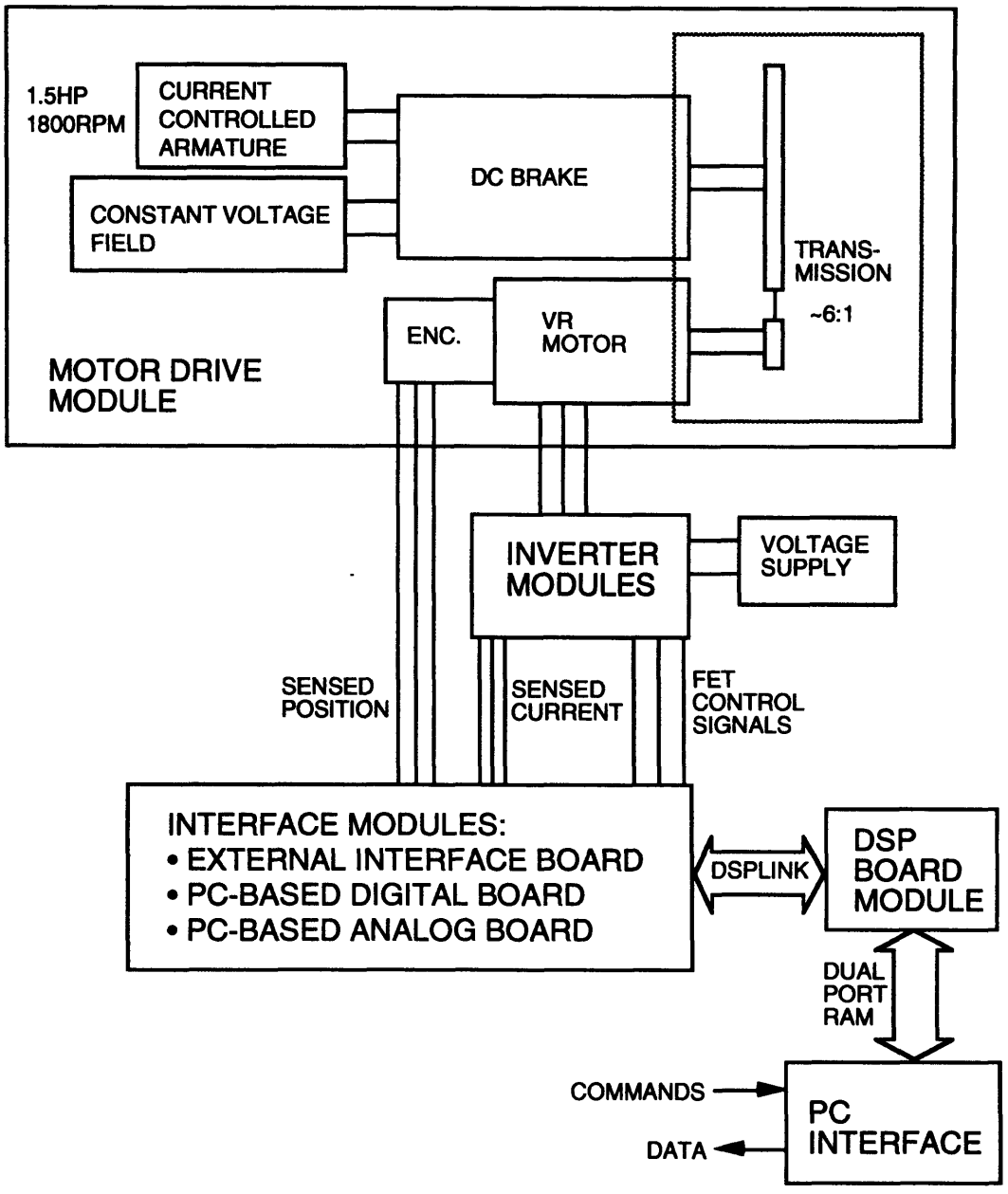

Figure 6.1: VRM experimental drive system.

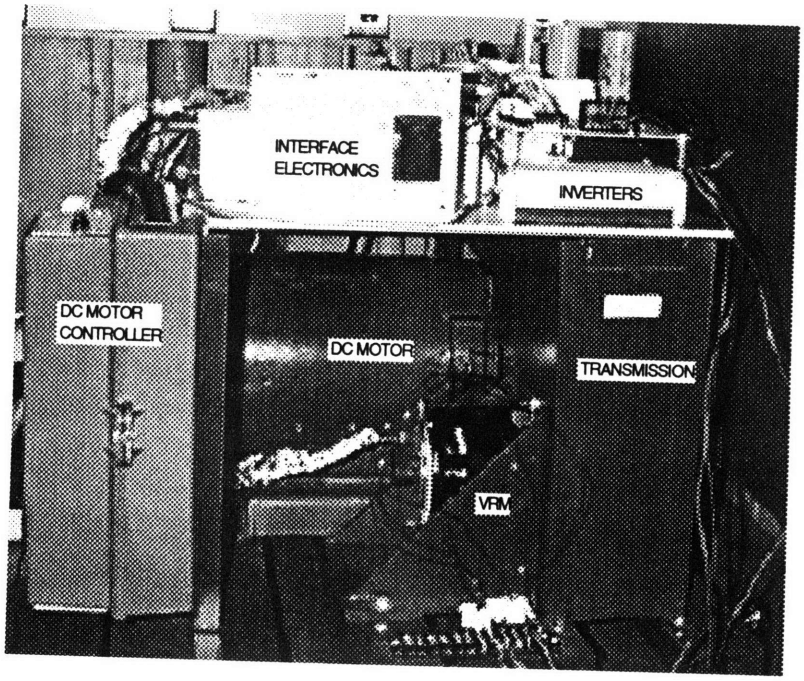

Figure 6.2: Drive system VRM and DC load.

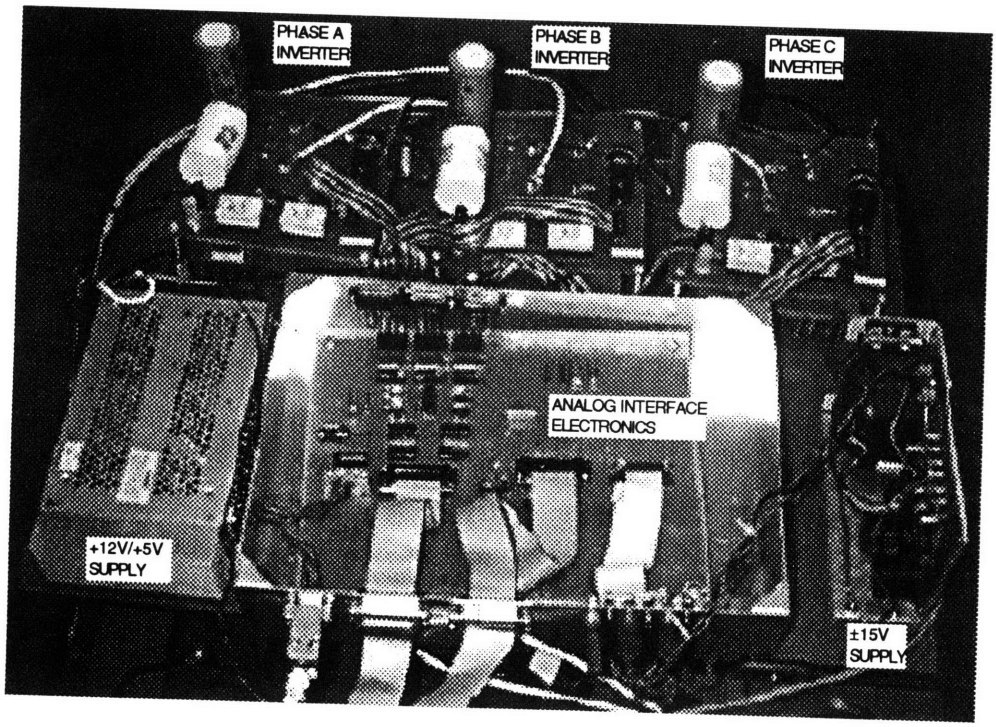

Figure 6.3: VRM drive system inverters, analog interface board, and signal supplies.

This system configuration provided the control and data acquisition flexibility necessary to evaluate the simulation. In a production drive system, computation would more likely be limited to analog hardware, digital timing circuits, and EPROM lookup tables for control angles versus motor speed. By keeping control angle selection entirely in software, various control strategies were easily considered. In addition, examination of the observer performance and troubleshooting was aided by this configuration.

Though the laboratory tests conducted only evaluated the VRM as a motor, the drive is fully capable of reverse operation with the DC motor acting as a motor and the VRM generating. In principle, the observer operation should be identical for generator operation. As discussed in Chapter 5, though, sensing accuracy varies with the relative measurement position, so the generator turn on angles may result in less precise output regulation.

6.2. Drive Electronics

The drive electronics consists of an internal digital logic board, an external interface board, three inverter boards, and a four-channel ADC board as shown in Figures 6.4 through 6.6. The PC-resident digital logic board communicates commands and digital feedback signals between the DSP board and the external interface board. Its specific functions include: inverter ON/OFF commands, position sensor decoding, index pulse detection, and encoder circuit counter reset commands. The external interface board processes commands from the internal digital board, and converts the current feedback signals from each of the three phase inverter boards to scaled voltage which can be read by the ADC board. The external board also performs an independent current limit lockout for each phase.

The position sensor decoding circuit on the internal digital board uses sequential logic gates and the DSP clock signal to output a 12-bit position value to the DSP in two words: a 4-bit MSW, and an 8-bit LSW. The decoding circuit is shown in Figure 6.4, and in greater detail in Figure 6.7.

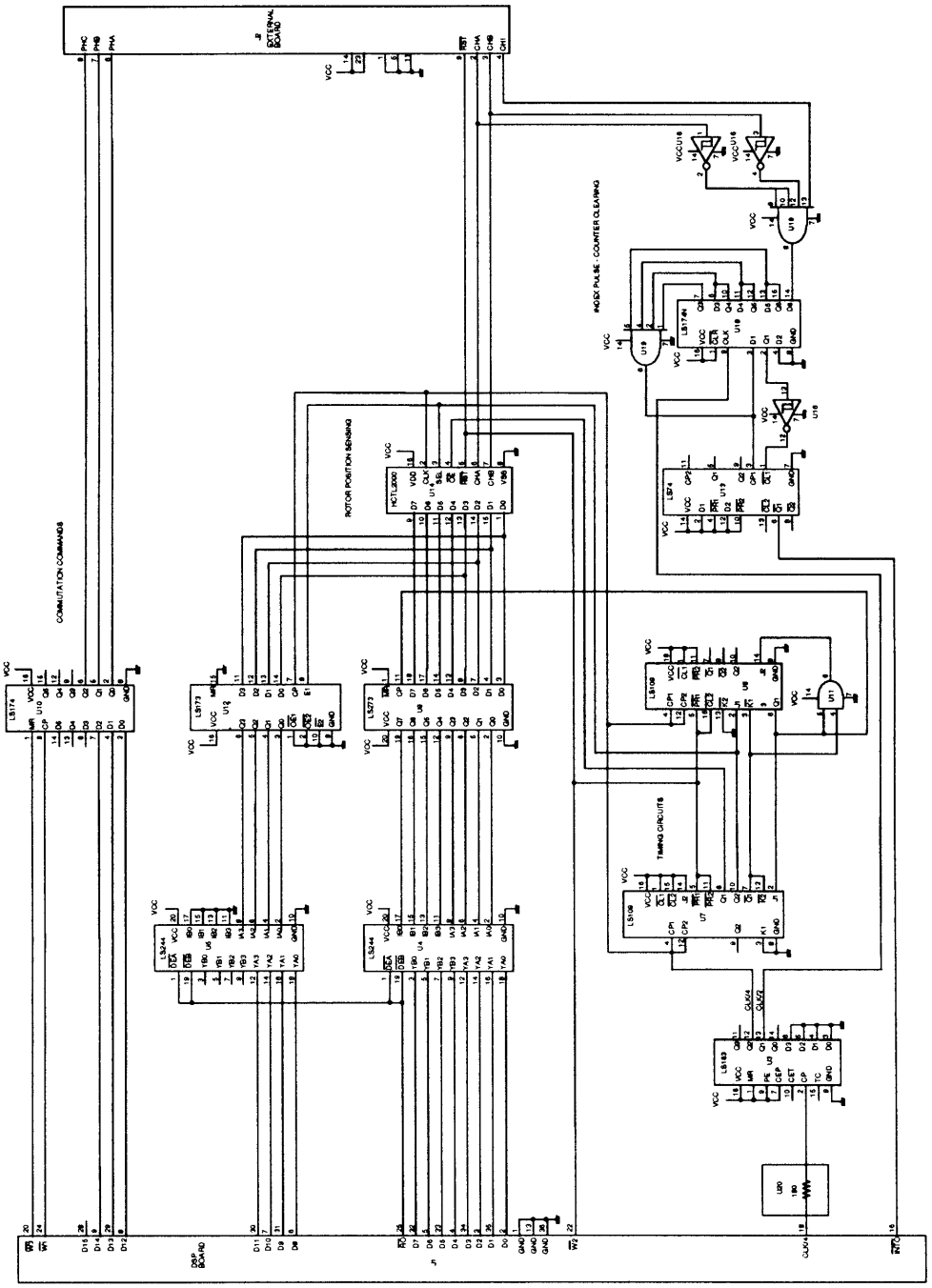

Figure 6.4: Internal digital board.

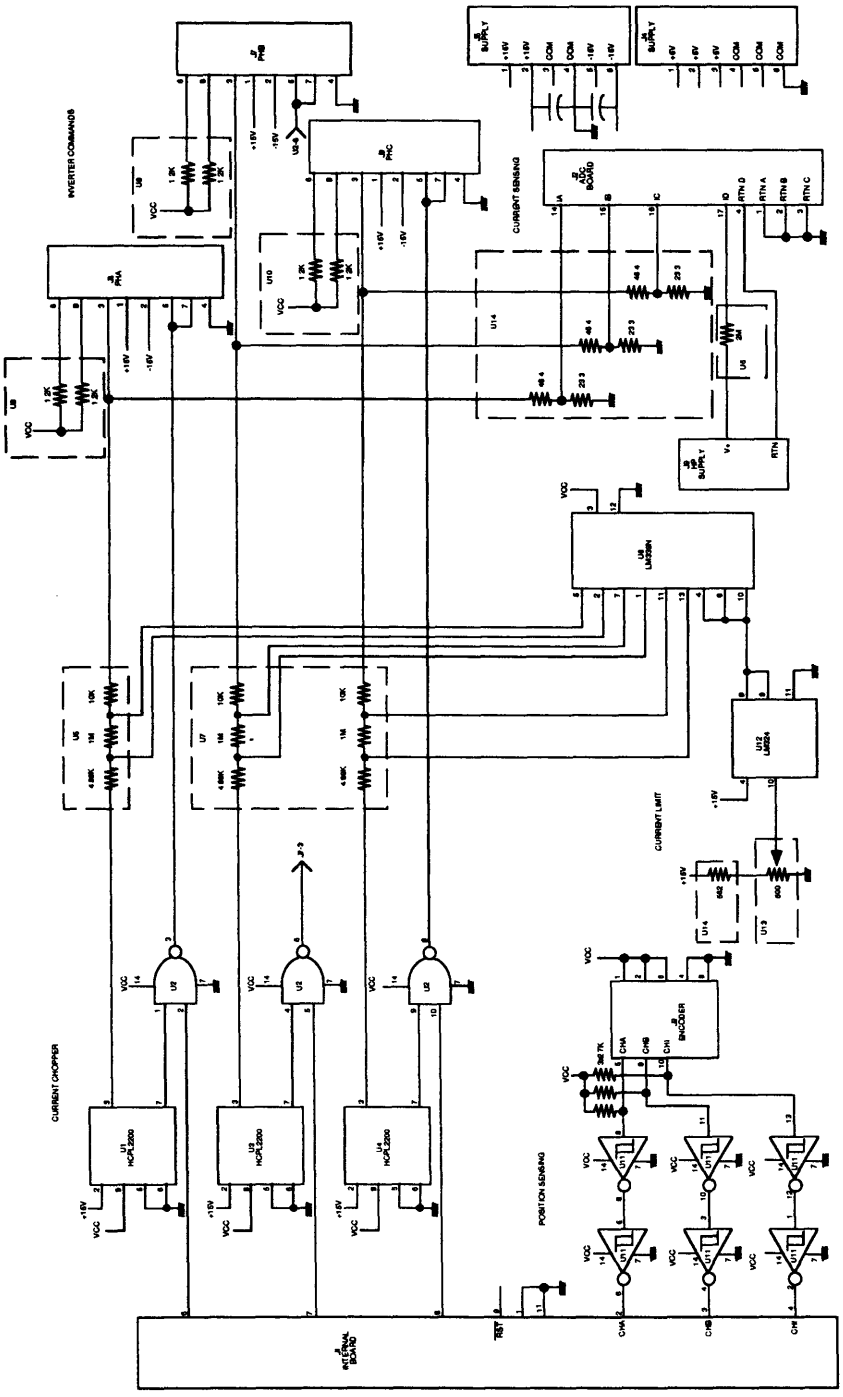

Figure 6.5: External interface board.

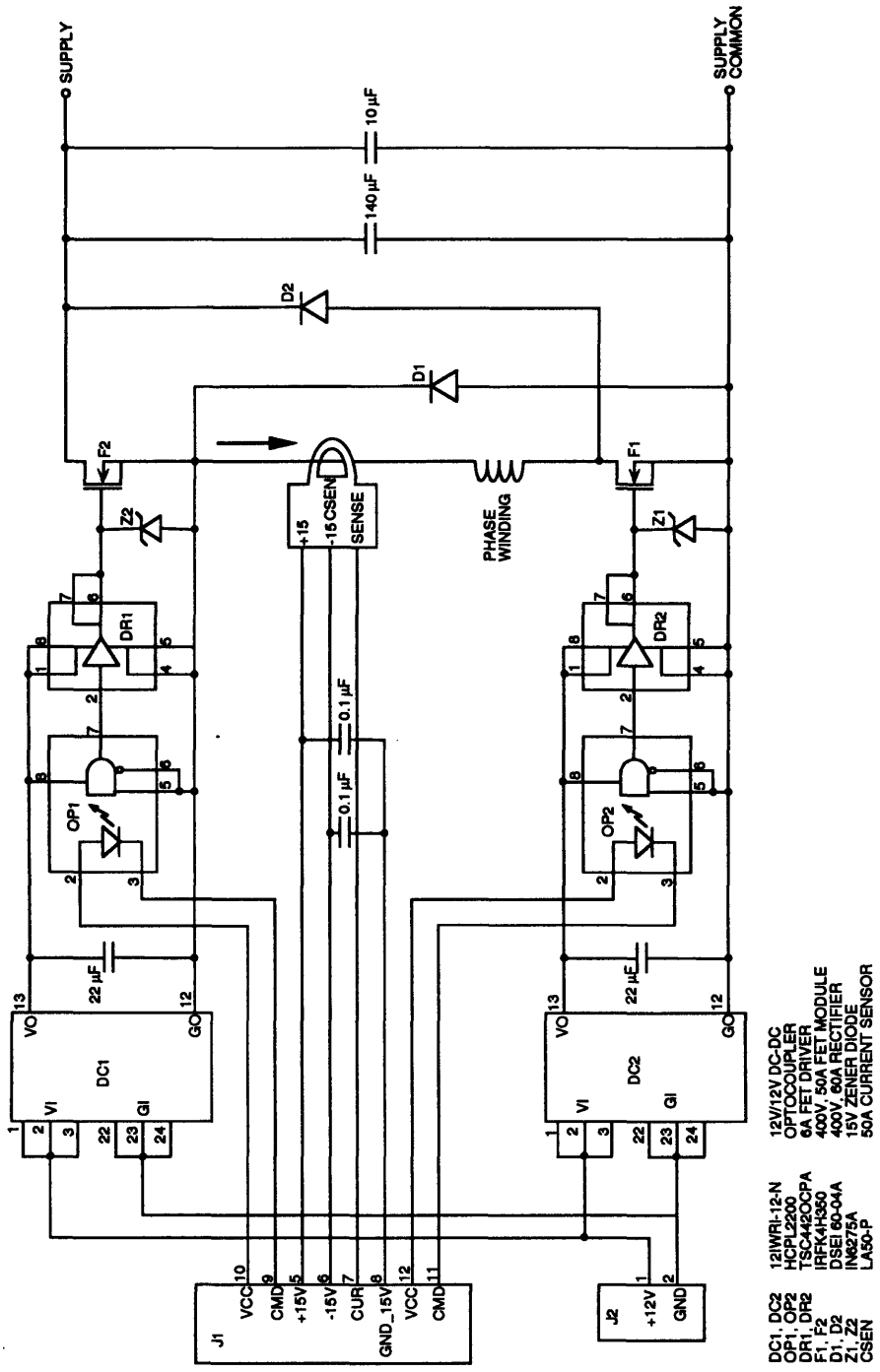

Figure 6.6: Inverter board.

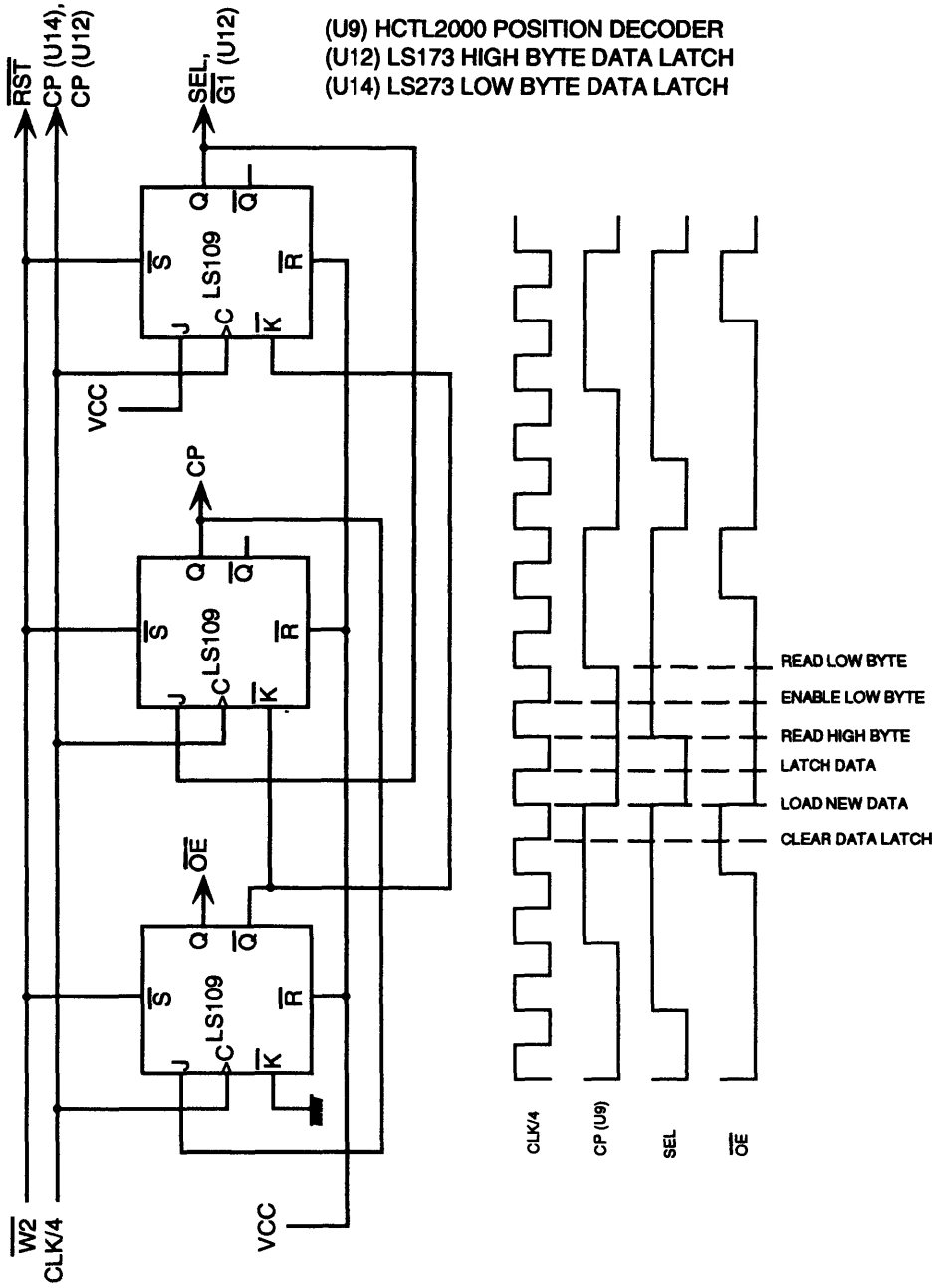

Figure 6.7: Functional details of the position decode control circuit for the HP encoder chip, HCTL2000, on the internal digital board shown in Figure 6.4.

Based on the HCTL2000 position decoder chip, this circuit latches the position data within the decoder chip and then outputs the MSW and LSW in sequence. The D-type flip flops latch the data until the next position is decoded. The logic is based on the external clock signal from the DSP which is four times the system clock period of 60 nsec. The position decoding circuit uses a divide by four circuit on the external DSP clock signal, therefore one decode sequence takes 4 x 4 x 60 nsec = 960 nsec. The DSP software asynchronously receives the latched position data using a read command to two tri-state buffers on the internal board.

The sequential logic is also used to fix the index pulse width at 120 nsec for an interrupt input to the DSP. The hardware interrupt is used by the DSP software to calculate speed averaged over one mechanical revolution. The DSP interprets an interrupt when the input level is low. Since the pulse width is determined by the speed with which the encoder wheel passes the detector, the index pulse is digitally processed to a fixed width of two DSP clock cycles to prevent multiple interrupts at low speeds. Figure 6.8 shows this circuit function in more detail. After the index pulse is detected by the DSP, the interrupt service routine commands a hardware write output which clears the HCTL2000 position counter for the next motor revolution. This prevents any cumulative position counting errors from occurring. The position decoder could have been wired directly to the index circuit to perform the counter clearing function in hardware, but the software control allows for more flexible troubleshooting of the counting circuit with the penalty of a small offset in the detected position. In this way, the counting errors can easily be determined by preventing counter clearing in software.

Current limiting is provided on the external interface board to prevent over-current conditions in each phase winding. This circuit is shown in Figure 6.9. Current is sensed on each phase inverter board using the LA50-P sensor with a current gain of 1000x. The motor phase is wound three times through the sensor resulting in an overall gain of $I_S = 0.003 I_P$. The sensed current signal, I_S, is input to a comparator circuit resulting in a non-inverting input voltage, V_+, proportional to the phase winding current. The inverting input reference, V_-, is established by a potentiometer and follower to set the current limit. When the comparator output is low, the optocoupler is on which allows the DSP commutation command to pass through the inverted AND gate. When the comparator output is high, the phase command is inhibited and the inverter is shut down. Circuit analysis shows that the state of the comparator output results in a hysteresis current limit which inhibits commutation when $I_P > (V_- / 0.205)$ and re-enables commutation when $I_P < (V_- / 0.205) - 0.654A$. The hysteresis allows more than sufficient time for the power FETs to fully reset before being turning back on.

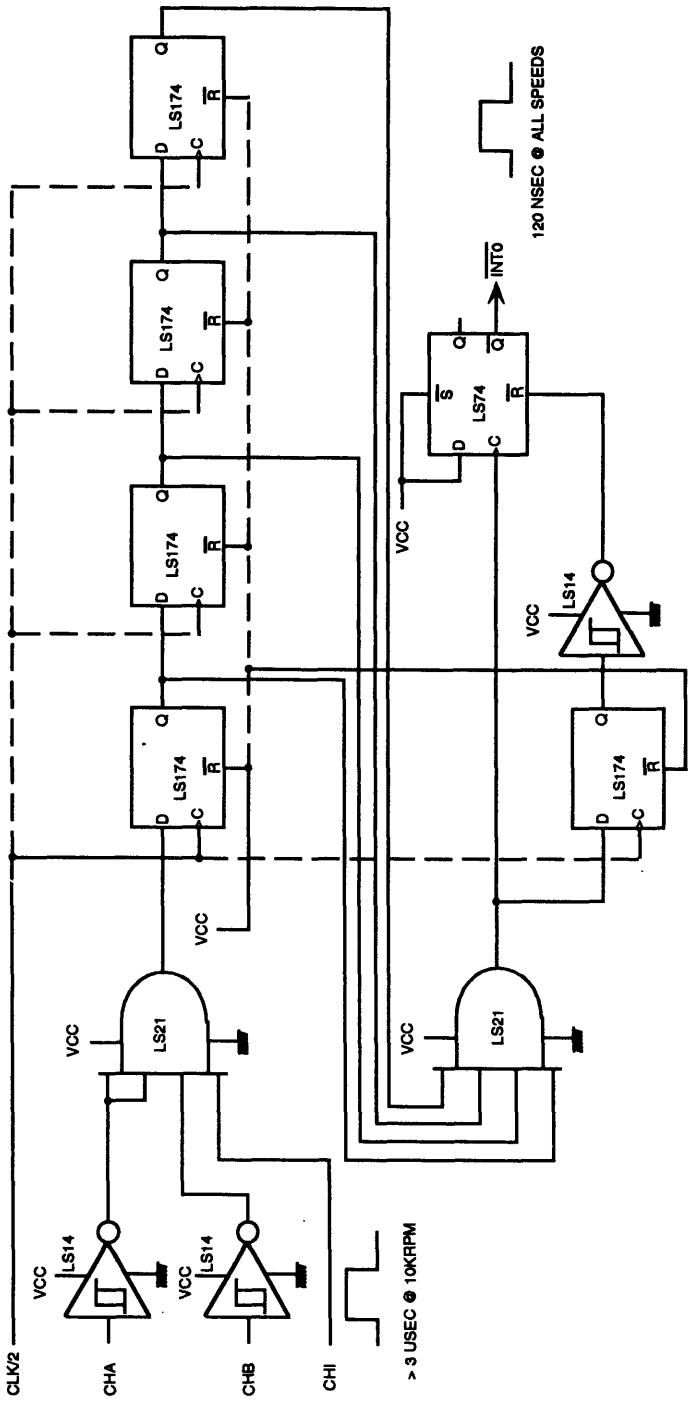

Figure 6.8: The index pulse logic from the internal digital board shown in Figure 6.4.

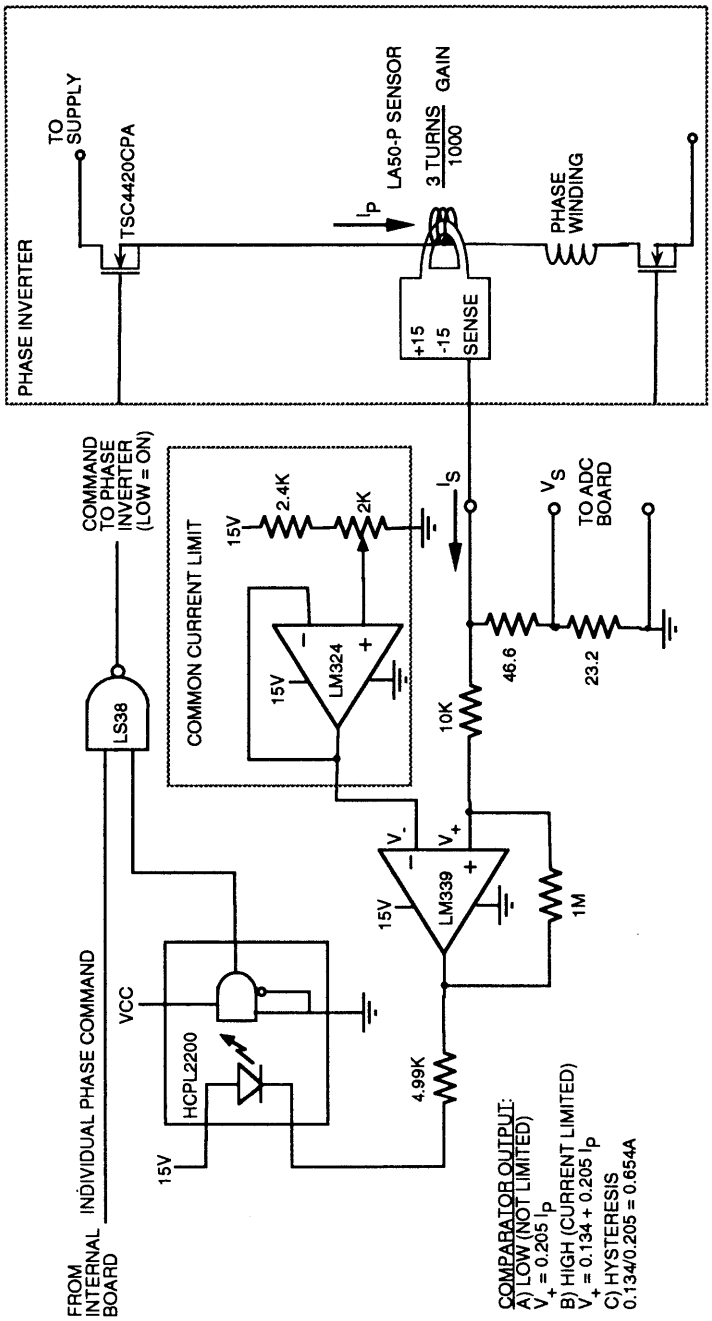

Figure 6.9: Functional details of the phase current limit circuit shown on the external board in Figure 6.5.

6.3 Controller

The controller for the experimental system is implemented using a PC-resident DSP system board coded in C. The user is provided with a control interface to start and stop the motor, change control angles when not operating in speed control mode, enable and disable estimator calculations, and switch both commutation and speed control between the sensed states (position encoder and index pulse calculated speed) and the estimated states (using the current feedback). The interface also displays real-time parameter values read from the dual port memory shared by the DSP board and the PC.

The DSP system board is from the Spectrum Corporation, and is based on the TMS320C30 DSP processor from Texas Instruments. The board provides an address, data, and control line external interface called the DSPLINK. This DSPLINK is used to communicate with the internal digital board (commutation control and encoder sensing), and the analog board (current and supply voltage feedback). With a processor speed of 33mHz, the C30 processor is able to execute over 4000 instructions between each commutation command for the given 6/4 VRM operating at 10,000 rpm. This is sufficient for the software control requirements. The board also provides two timers, two software interrupts, two hardware interrupts, and an external 8.25MHz (120 nsec) clock.

Figure 6.10 shows a flowchart of the software control program. Note that the main loop runs continuously to maximize the data collected within an electrical cycle. A software interrupt is used to start the motor spinning open-loop in the right direction using a specified commutation time interval. The motor speed calculation from the index pulse is accomplished with a hardware interrupt service routine which interrupts the main routine when an index pulse is detected. After the motor is spinning, commutation is controlled by the main loop default mode using the position encoder and fixed control angles. The control mode changes are initiated when new commands are received from the PC user interface.

When the observer logic is enabled, the observation is accurately controlled by starting a timer when a phase is turned on. The software waits in a loop until the observation time is reached, and then the current measurement is taken. The observer then calculates position from a current-position lookup table that was generated by the PC control program for a fixed observation time delay. The observation time delay is selected by the user during program initialization. Finally the states are updated using the new observer position error, and the calculated electrical torque. Calculating the torque output by the motor is computation-intensive, and so, is calculated asynchronously by the PC control program using the control angles and the appropriate

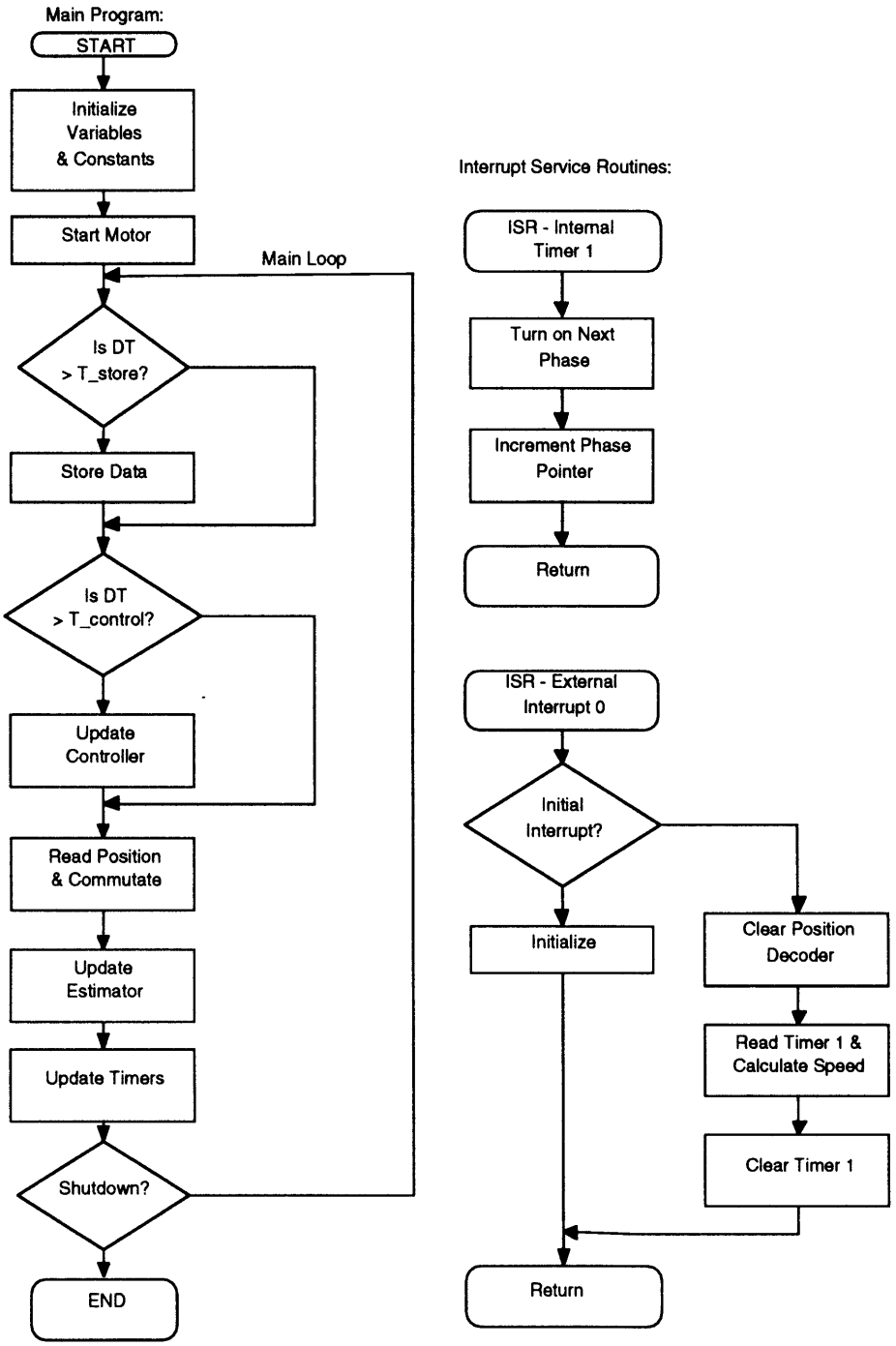

Figure 6.10: Software flowchart.

state vectors. The calculated average torque is then transferred to the DSP main loop as it is updated by the PC. The inherent time delay between average torque updates does not introduce significant errors since the mechanical dynamics of the system are significantly slower than the torque computation time.

6.4 Experimental Performance

Experimental tests were conducted using the VRM drive to check the predictions of the simulations in Chapters 3 through 5. This section begins by reviewing the important features of the simulations that were demonstrated in the previous chapters. Then the experimental test configurations that were used to test whether the simulations accurately predicted experimental performance are outlined. Before presenting the experimental transients, though, additional controller restrictions that were necessary to operate the VRM drive are discussed.

In the previous chapters simulations were presented showing the instantaneous torque, flux linkage, and current over an electrical cycle. The controller simulations demonstrated the stability and transient performance for speed setpoint changes. This was followed an independent evaluation of the observer using simulations that showed stable error rejection and decay.

In this section, experimental tests are presented that were used to evaluate the simulation predictions of these above features. Additional simulations are also presented as necessary to change the predictions to reflect restrictions that were imposed on the experimental drive. The experiments and simulations that are presented are as follows:

1. Experiment with on-line observer feedback at regulated speed with imposed initial errors. This experiment is used to evaluate the error rejection and stability of the observer. It is compared to an off-line observer simulation from Chapter 5.

2. Experiment with on-line observer feedback for a transient speed setpoint change. This experiment demonstrates the stability and trajectory of the controller for large speed errors. Various control limitations are encountered and discussed.

3. Experiment with measured current during one electrical cycle at regulated speed. This experiment demonstrates the accuracy of the electromagnetic motor model

used in the simulations and the experimental observer. It is compared directly to the simulation predictions.

4. Simulation with on-line observer feedback for a transient speed setpoint change including controller restrictions. This simulation incorporates the control restrictions necessary for the experimental drive, and is used to compare directly to Experiment #2 above.

5. Experiment showing maximum speed transient with on-line observer feedback. This experiment is used to demonstrate the maximum possible speed with the experimental system and the limitations that prevented reaching the design speed for the drive.

6. Simulation at the design speed using the on-line observer feedback. This simulation demonstrates the predicted performance at the design speed 10 krpm.

In the experiments and simulations that are compared in this section, all the experiments incorporate an on-line observer which provides the feedback necessary for the transient speed controller. Some of the simulations conducted used an off-line observer where transient speed control feedback was provided by simulated motor position and speed. It was shown in Chapter 5, that the interaction between the controller closed-loop dynamics and the observer closed-loop dynamics was minimal so that it is not necessary to prove that the Eigenvalue Separation Theorem [14] holds by repeating simulations with both on-line and off-line observers.

The controller and observer were tested in various operating modes to investigate their stability. Some basic operating restrictions were developed to ensure stable operation. First, it was observed that the position sensing based on current was less accurate near misalignment. This is easily understood by referring to Figure 5.5 which shows the variation of sensed current with position. Near misalignment at 45 degrees the slope approaches zero, thus small changes in sensed current could result in significant shifts in interpreted position. Tests showed that the observer could be used to control the drive in a stable manner as long as the turn on angle was not driven near 45 degrees from either a higher or lower initial turn on angle. Operation near misalignment resulted in the observer often correcting the estimated position in the wrong direction since it could not distinguish between 40 and 50 degrees. Since the full-range of positive torque output could be achieved with operation below 45 degrees this restriction was imposed. The controller was modified for the experimental drive to force θ_{on} to $\theta_{on,min}$ and modulate θ_{cond} for torque control. For cases where fast deceleration is necessary an algorithm could be developed to allow transient operation

through 45 degrees. This is a potential area for further development which is discussed in the final chapter.

Another operating restriction introduced in the controller development was slew rate limiting. The controller analysis developed gains for stable operation around steady-state operating points. In order to ensure stable operation for large speed setpoint changes a slew rate limit was implemented in terms of the controller speed error input as described in Section 4.4. This limit was determined experimentally and is noted with the test results that follow.

Figures 6.11 through 6.13 show the response of the experimental observer and the corresponding observer simulation from Figures 5.9 through 5.11 to initial state errors. In both the experiment and the simulation, the observer is 'on-line' providing the speed and position feedback necessary for the controller. The experiment parameters are given in Table 6.1. The figures show that both position and speed error states decay exponentially. The steady-state position error variation, 1.03 degrees rms, shown in Figure 6.11 is due to a combination of the accuracy of the current sensing and the position inversion algorithm. The actual position and speed are determined from the shaft encoder and the index pulse. In Figure 5.5 the position versus current curve used to determine the rotor position was presented for a supply voltage of 30 VDC. This same curve is used in the experimental drive system for any supply voltage by linearly scaling the sensed current with respect to the supply level, and then conducting a spline interpolation on a data table. Expressing this curve as before in Equation [5.34], the position estimate as function of the supply voltage and measured current is given by

$$\theta = f^{-1}(30I/V_b, (30V)(83\mu sec)), \tag{6.1}$$

where the function $f^{-1}()$ is the curve given in Figure 5.5. Since the $\lambda - i - \theta$ relationship is incorporated in the DSP software algorithms, the rotor position could have been calculated for an arbitrary sensing time and supply voltage but due to timing constraints this was not possible. The error decay rate shown in Figure 6.11 is indistinguishable from the corresponding simulation results of Figure 5.6 , thus the simulation and experiment observer poles are as originally predicted in Table 5.1. The simulation results are plotted again in Figures 6.11 and 6.12 for a direct comparison with the experimental results. Note that the initial increase in the position state error is not due to a minimum phase effect, but is simply a function of the rotor speed and rotor speed error at the same update time. For the simulation the actual position and speed are calculated by the simulated plant dynamics.

Table 6.1: Experimental test parameters for the 'on-line' observer response to initial errors shown in Figures 6.11 through 6.13.

Parameter	Units	Value
V_b	[Volts]	68
θ_{on}	[degrees]	29.0
θ_{cond}	[degrees]	15.5
$\tilde{\theta}_0$	[degrees]	5.0
$\tilde{\omega}_0$	[rpm]	200
ω	[rpm]	3500

The various sensing system update rates can also be observed in Figure 6.13. Since motor speed is only calculated each time an index pulse is received, its update is significantly slower than the observer speed which updates every DSP program cycle. In this experiment the DSP loop is 300 μsec and at 3500 rpm the motor speed is calculated every 17.1 msec. The observer innovation term is only non-zero, though, each time a phase commutates which is 12 times per revolution (3 phases times 4 electrical cycles per mechanical cycle). At 3500 rpm, the innovation occurs every 1.43 msec.

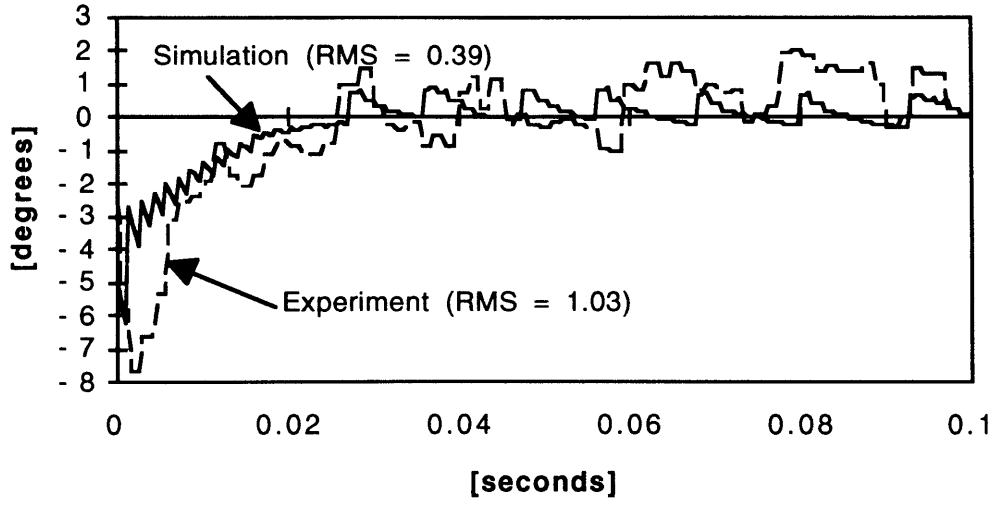

Figure 6.11: Position error (actual - observer) versus time for a simulation and an experiment given an imposed initial state error.

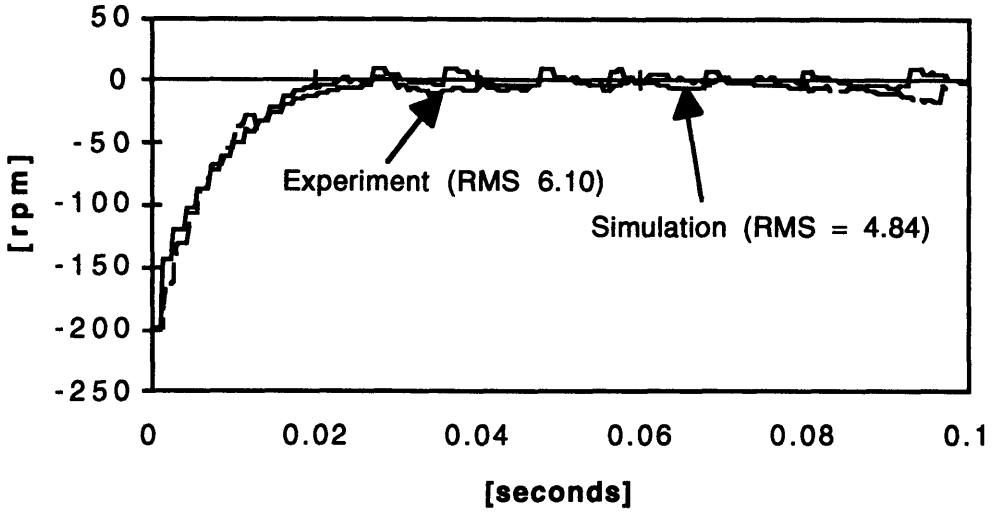

Figure 6.12: Speed error (actual - observer) versus time for a simulation and an experiment given an imposed initial state error.

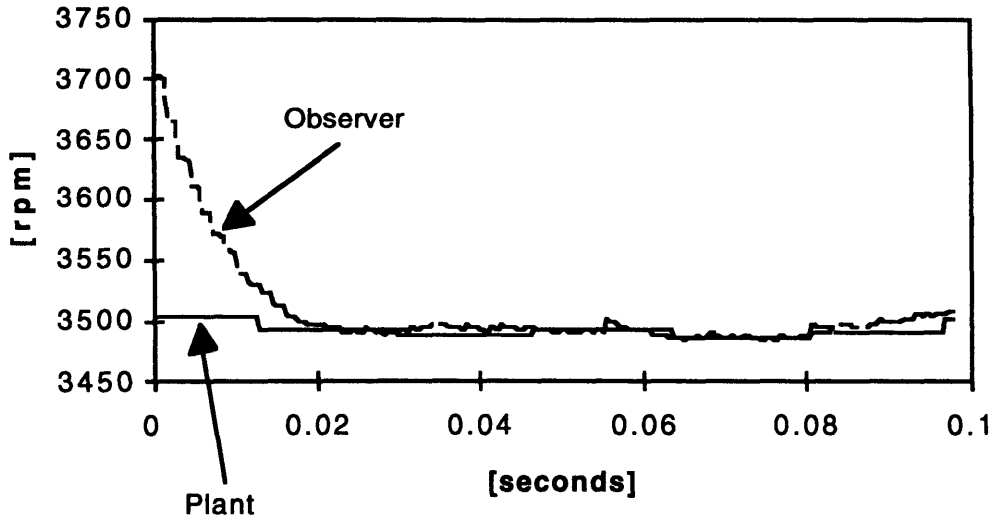

Figure 6.13: Observer and encoder calculated speed versus time for a simulation and an experiment given an imposed initial state error.

Figures 6.14 through 6.20 show the transient response of the drive to a large speed setpoint change with the observer on-line providing position and speed feedback to the controller. The experiment parameters for this test are given in Table 6.2 with the initial observer errors set to zero. Table 6.2 also shows a conduction angle limit, $\theta_{cond\,max}$, and a control angle slew rate limit, $\hat{\omega}_{rate\,lim}$, in terms of the speed error command. The slew rate limit was experimentally determined to insure stable operation for large setpoint changes and was employed in the VRM simulations in Chapter 4. The conduction angle limit was added due to switching noise which corrupted the current sensing as the conduction angle approached 30 degrees. This switching noise is examined later in this chapter for in higher speed transient to 6000 rpm.

This transients shows both the stability of the controller for a large setpoint change with the observer providing the controller feedback inputs. This previous figures spanned a shorter time scale to demonstrate the observer error rejection. In that time frame there is minimal controller action due to the update rate of the controller and the relatively slow mechanical dynamics of the motor. In Figure 6.14 the observer provides almost perfect tracking of the actual rotor speed as calculated from the index pulses. The steady-state error ranges shown in Figures 6.15 and 6.16 are 2 degrees and 10 rpm. The apparent offset in the steady-state observer position is not real, and is due to software timing constraints in observer and position encoder calculations. The offset is equal to the time between calculating the observer position and reading the shaft encoder times the motor speed. Since the DSP program loop time is approximately constant, this steady-state position error will be proportional the instantaneous motor speed.

The relative contributions of mechanical acceleration and the observer innovation term to the estimated speed are represented in Figure 6.17 for the same experiment as above. The mechanical acceleration term as calculated from the average motor torque is insignificant compared to the observer innovation term. This demonstrates that the stability of the observer is tolerant of significant prediction errors when the torque imbalance, or accelerating torque, is not extreme. This means that for certain applications a sophisticated torque model may not be necessary.

Table 6.2: Experimental test parameters for the speed setpoint change with on-line observer shown in Figures 6.14 through 6.20.

Parameter	Units	Value
V_b	[Volts]	68
$\hat{\omega}_{rate\,lim}$	[rad/sec/update]	4
$\theta_{cond\,max}$	[degrees]	20
$\tilde{\theta}(0)$	[degrees]	5.0
$\tilde{\omega}(0)$	[rpm]	200
$\omega(0)$	[rpm]	2000
Ω	[rpm]	3500

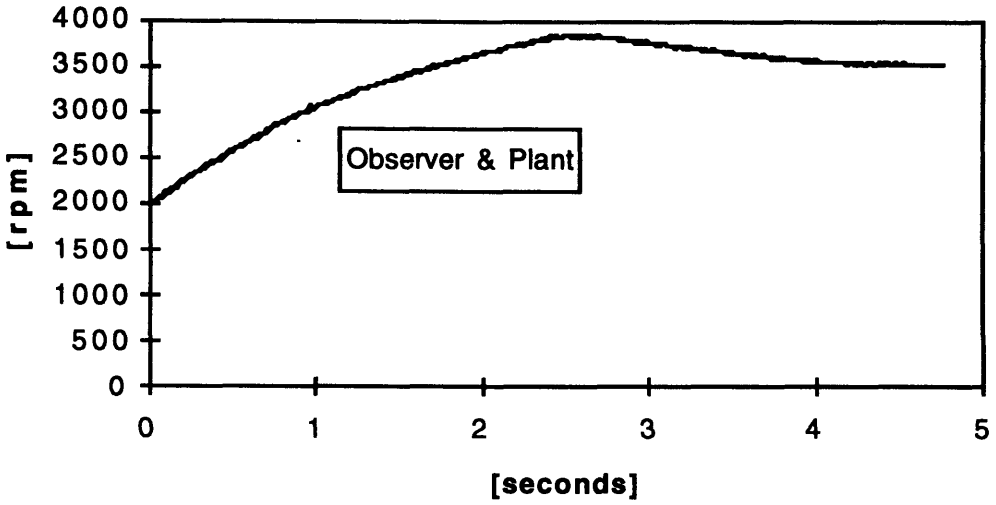

Figure 6.14: Observer and encoder calculated speed versus time given a speed setpoint change from 2000 rpm to 3500 rpm.

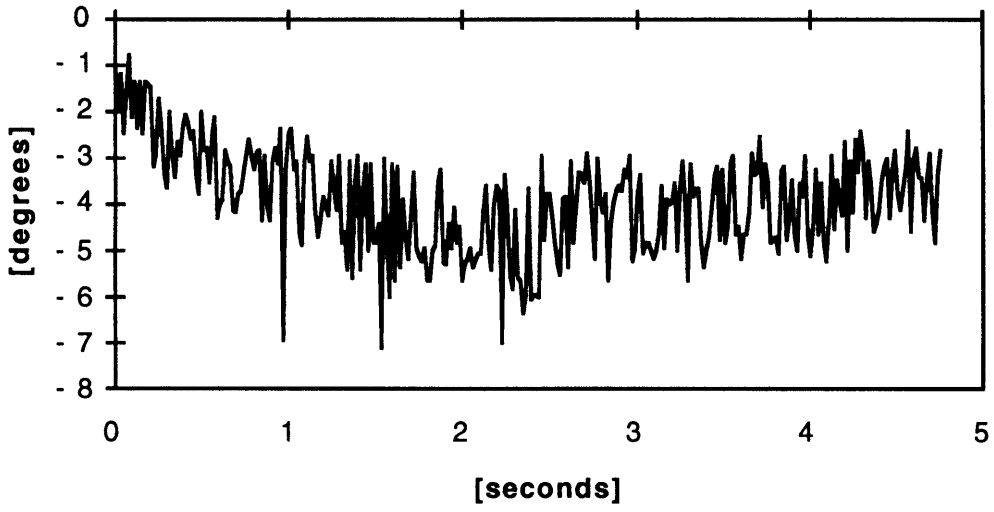

Figure 6.15: Position error (observer - encoder) versus time for a setpoint change from 2000 rpm to 3500 rpm.

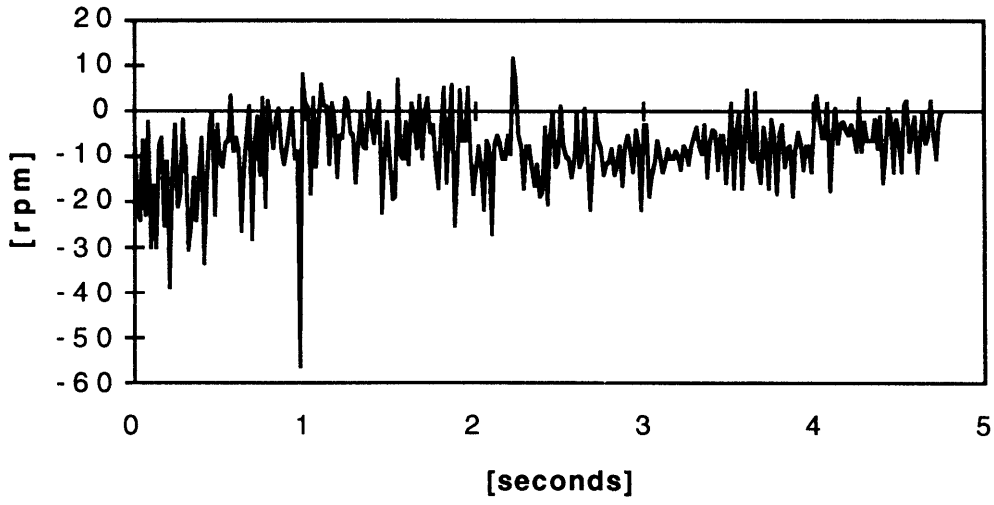

Figure 6.16: Speed error (observer - index pulse calculation) versus time for a setpoint change from 2000 rpm to 3500 rpm.

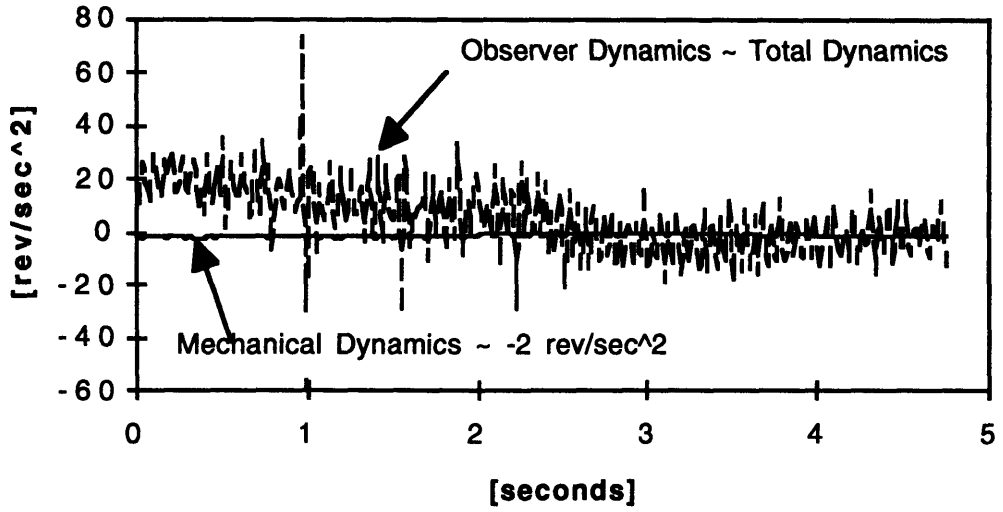

Figure 6.17: Contributions of observer and mechanical terms to observer estimate of the motor acceleration showing dominance of observer terms.

To check the accuracy of the VRM $\lambda - i - \theta$ calculations, Figure 6.18 shows the simulated and experimental phase current for an electrical cycle for the conditions given in Table 6.3. The multiple experimental samples are data points from a series of electrical cycles. The scatter is due primarily to small variations in the conduction angle. The figure confirms that the VRM electromagnetic representation is accurate within 10% except at the peak current. Since the torque is calculated directly from the phase currents, the predicted electrical torque must be fairly accurate. Since the torque imbalance is not accurate as shown above, this leads to the conclusion that the errors in the predicted mechanical dynamics are due to incorrect load estimates. Recall that the form used for the load function includes viscous and Coulombic friction terms (*B, C*) with the load function set to zero ($V_L = 0$). Either this form is inappropriate for the experimental motor load, or the test data used to estimate these parameters was not sufficient to determine a unique, but accurate, solution.

Table 6.3: Experiment and simulation test parameters for the measurement of current over one electrical cycle shown in Figure 6.18.

Parameter	Units	Value
V_b	[Volts]	68
θ_{on}	[degrees]	32
θ_{cond}	[degrees]	13.5
ω	[rpm]	2000 rpm

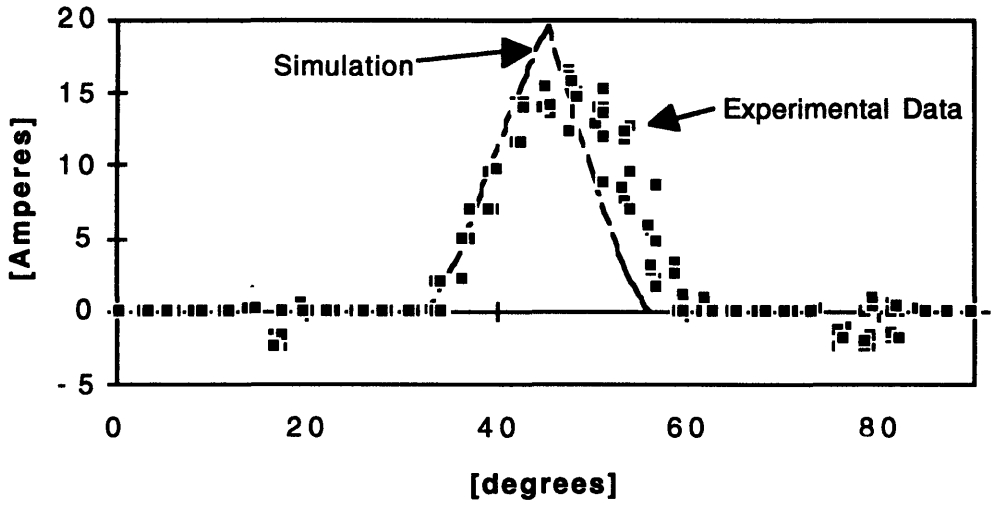

Figure 6.18: Simulated and experimental current profiles for the conditions given in Table 6.3.

The control angles for the same transient given by Table 6.2 are shown in Figures 6.19 and 6.20. The experimental system software records the control angles computed by the speed controller algorithm with the other measured and calculated data presented in Figures 6.14 through 6.17. Figures 4.6 and 4.7 showed the control angles for a similar transient control experiment. The transient evolution of the control angles in Figures 6.19 and 6.20 does not follow the same trajectory as the simulated transient in Figures 4.6 and 4.7. This is due to the incorporation of the lower maximum limits on θ_{cond}, and because θ_{on} must be driven to $\theta_{on,min}$ for the experimental drive.

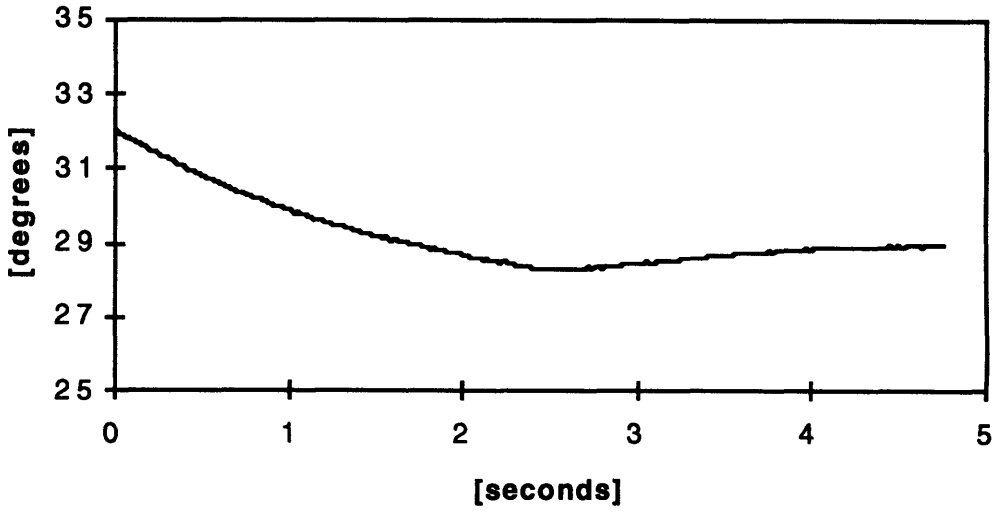

Figure 6.19: Turn on angle for experimental drive speed setpoint change from 2000 rpm to 3500 rpm. The turn on angle is computed by the speed controller.

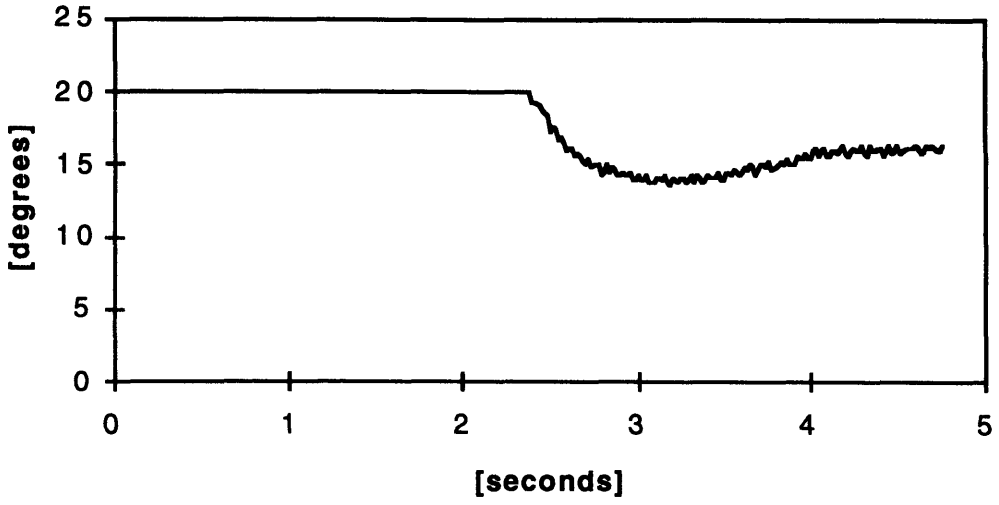

Figure 6.20: Conduction angle for experimental drive speed setpoint change from 2000 rpm to 3500 rpm. The conduction angle is computed by the speed controller.

If the same controller restrictions are incorporated into the transient simulation, the simulation develops insufficient drive torque to reach 3500 rpm while the experimental

VRM can actually reach 3500 rpm with these restrictions. In Figures 6.21 through 6.24 the transient simulation discussed in Chapter 3 is repeated with $\theta_{on} = \theta_{on,min}$ and with $\theta_{cond} \leq 20$ degrees. The simulation parameters, shown in Table 6.4, are the same as the experimental parameters given in Table 6.2. Figure 6.23 shows that even with both control angles operating at their respective limits, the maximum speed of the VRM simulation is approximately 2615 rpm. The average electrical torque is shown to fall below the viscous plus Coulombic load torque in Figure 6.24. The average output power calculated at this condition is 0.15 hp. This further confirms that the predicted load that was estimated from spin down test measurements is significantly higher than actually present in the experimental drive.

Table 6.4: Simulation test parameters for the speed setpoint change with on-line observer shown in Figures 6.21 through 6.24.

Parameter	Units	Value
V_b	[Volts]	68
$\hat{\omega}_{rate\,lim}$	[rad/sec/update]	4
$\theta_{cond\,max}$	[degrees]	20
$\tilde{\theta}(0)$	[degrees]	5.0
$\tilde{\omega}(0)$	[rpm]	200
$\omega(0)$	[rpm]	2000
Ω	[rpm]	3500

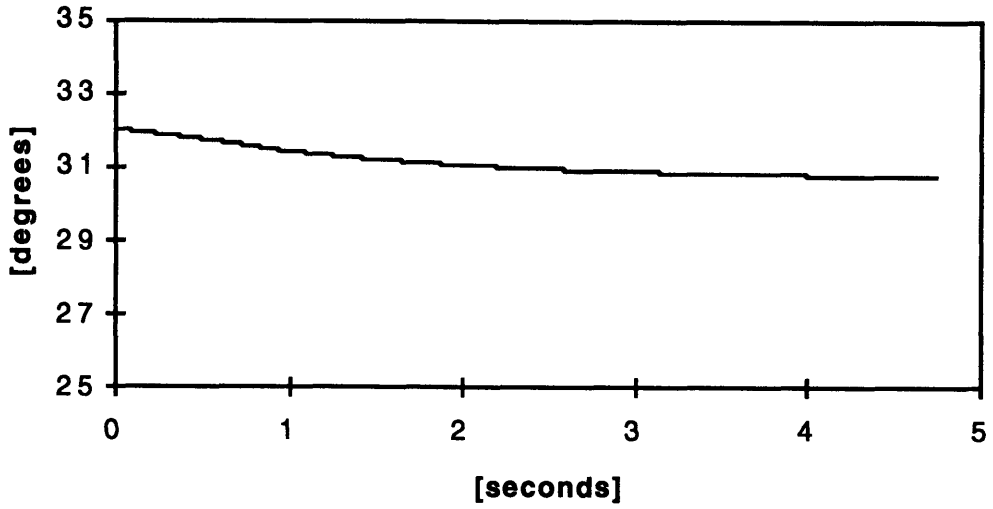

Figure 6.21: Simulation of the turn on angle for a transient from 2000 rpm to 3500 rpm using the controller restrictions developed for the experimental drive.

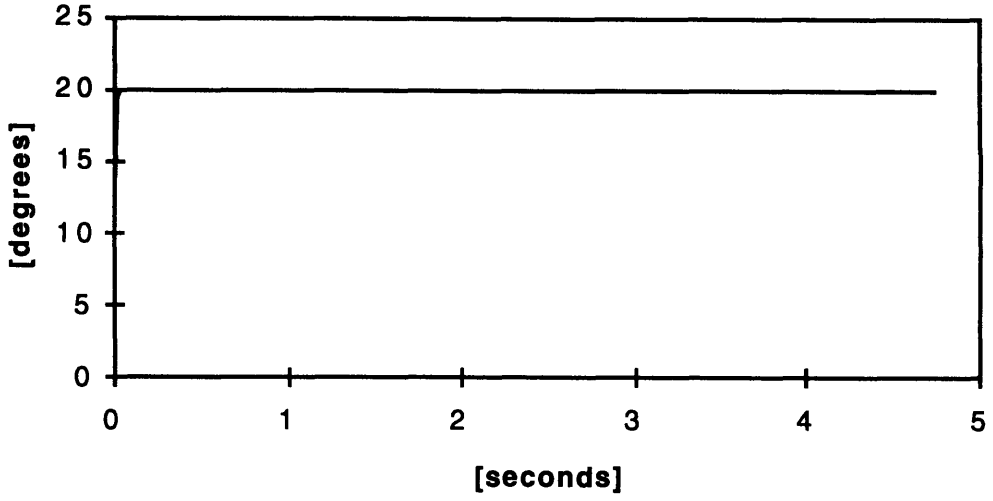

Figure 6.22: Simulation of the conduction angle for a transient from 2000 rpm to 3500 rpm using the controller restrictions developed for the experimental drive.

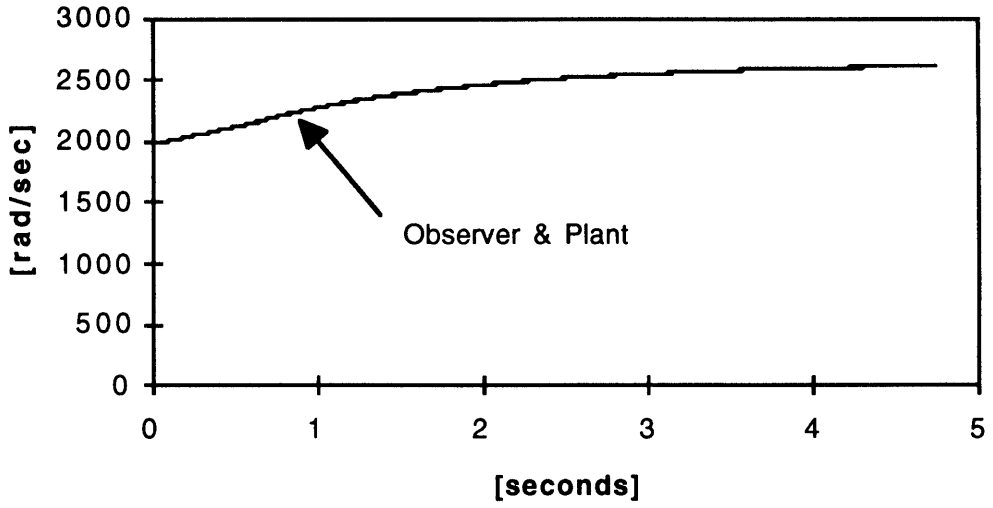

Figure 6.23: Simulation of the VRM plant and observer speeds for a transient from 2000 rpm to 3500 rpm using the controller restrictions developed for the experimental drive.

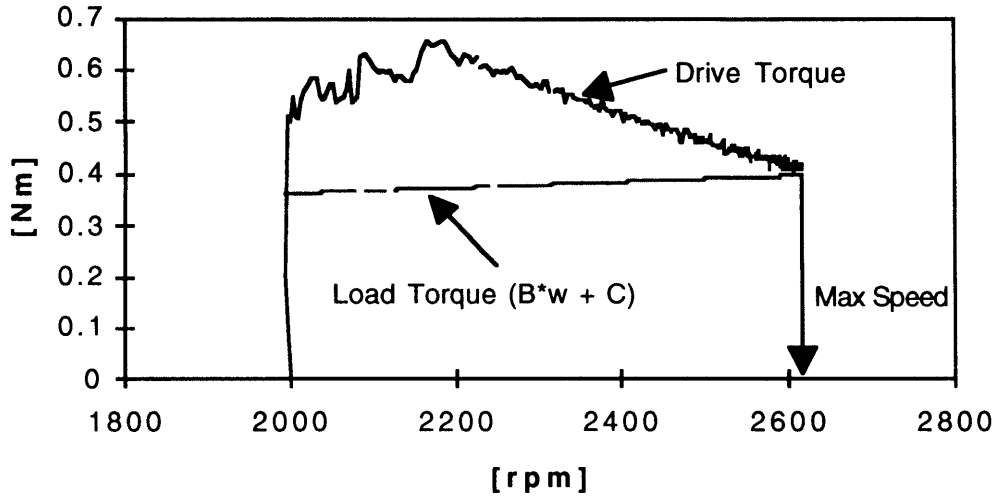

Figure 6.24: Simulation of the electrical drive and load torques versus VRM speed for a transient from 2000 rpm to 3500 rpm using the controller restrictions developed for the experimental drive.

Table 6.5 and Figures 6.25 through 6.28 show results for an experimental setpoint transient from 5500 rpm to 6000 rpm. Due to the switching noise which limited the conduction angles to less than 30 degrees, this became the practical speed limit for the experimental drive system. As can be seen in Figure 6.25, the speed tracking of the observer is more erratic at this condition. This is because the conduction angle limit was set higher than in previous experiments in order to develop enough torque to reach 6000 rpm. As the conduction angle gets closer to 30 degrees, though, the corruption to the sensed current for the observer gets worse. Additionally the maximum slew rate was lowered in order to maintain stable operation with this additional current measurement noise. The measurement noise is clearly observed in Figure 6.28 where a significant percentage of the current data is out of range. The controller software is designed to ignore the out of range data, but repeated data failures compromises the system stability and appropriateness of the observer gains.

Table 6.5: Experimental test parameters for setpoint change shown in Figures 6.25 through 6.28.

Parameter	Units	Value
V_b	[Volts]	68
$\hat{\omega}_{rate\,lim}$	[rad/sec/update]	1
$\theta_{cond\,max}$	[degrees]	25
ω	[rpm]	5500
Ω	[rpm]	6000

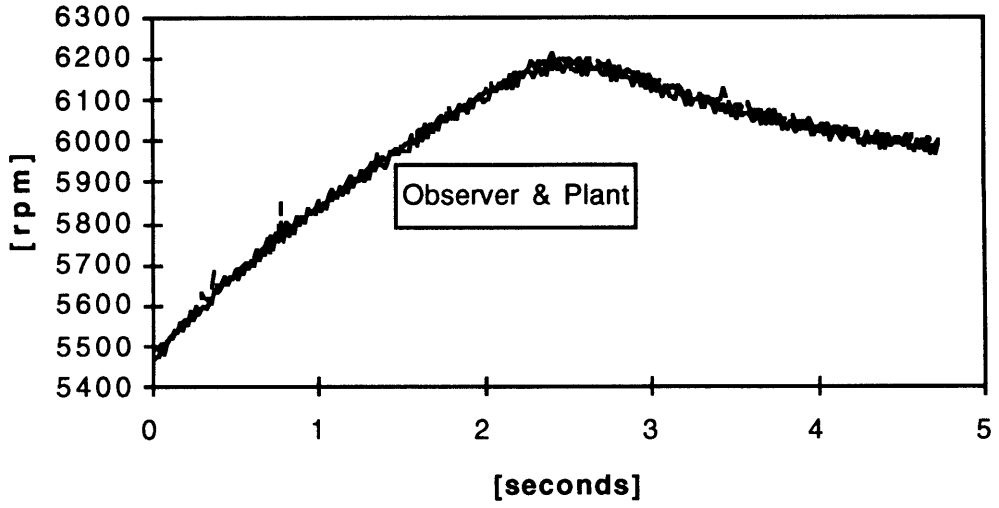

Figure 6.25: Observer and encoder calculated speed versus time given a speed setpoint change from 5500 rpm to 6000 rpm.

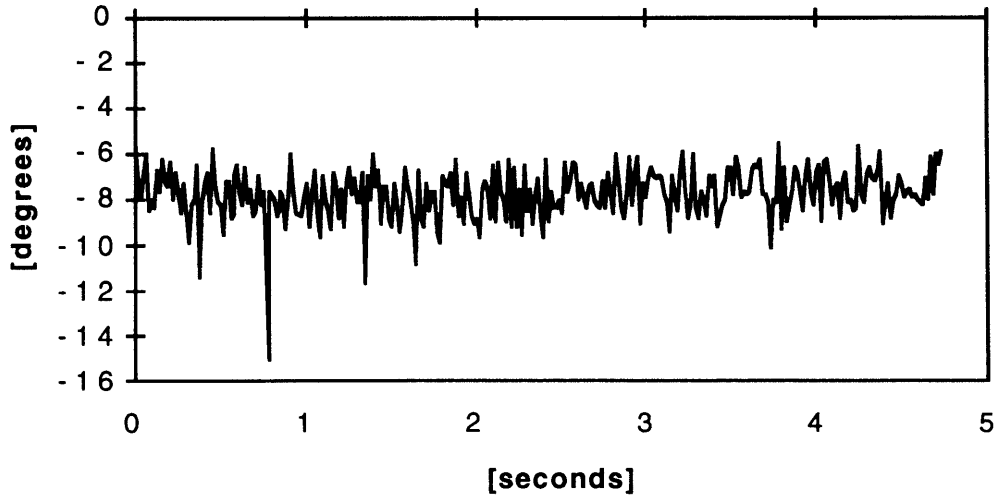

Figure 6.26: Position error (observer - plant) versus time for a setpoint change from 5500 rpm to 6000 rpm.

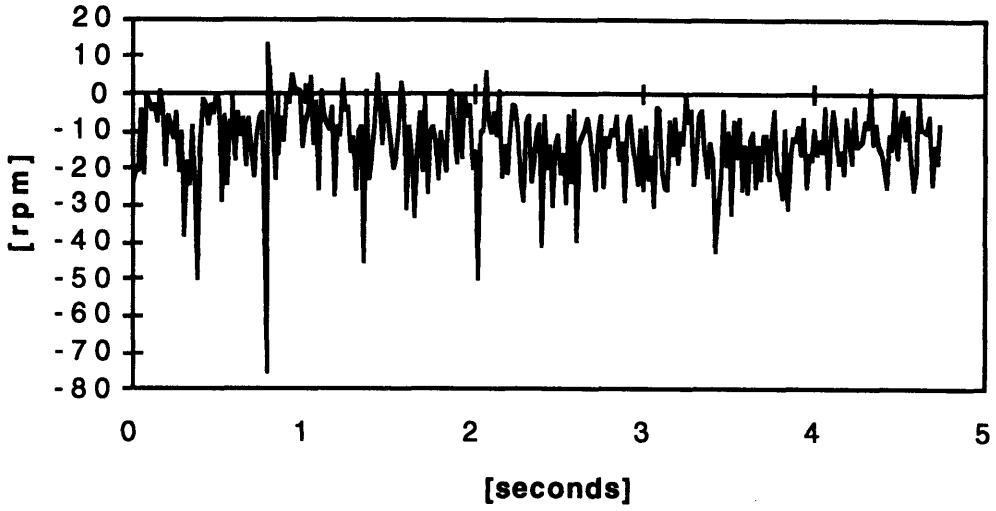

Figure 6.27: Speed error (observer - plant) versus time for a setpoint change from 5500 rpm to 6000 rpm. The noise coupled from the previous phase turning off causes spikes in the measured current which are translated to the position input to the observer.

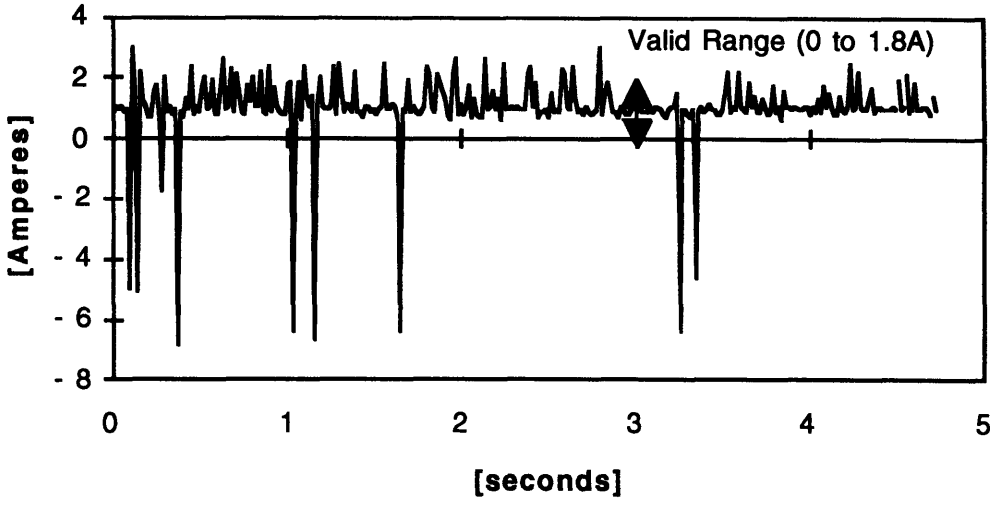

Figure 6.28: Current measurements showing a significant percentage of invalid measurements for a setpoint change from 5500 rpm to 6000 rpm. The noise coupled from the previous phase turning off causes spikes in the measured current which are translated to the position input to the observer.

The current measurements are corrupted by noise coupling from another phase which has recently been turned off. When a phase is turned off the current continuity through the motor winding inductance causes the voltage across it to be reversed almost instantaneously. This noise is capacitively coupled back to the analog interface board and from there to the other signals (commutation commands and current sensing) and the other phases. Corrupting the commutation command results in multiple turn on and turn off commands to the inverters which extends the duration of the noise rippling. The amplitude and duration also appears to increase with supply voltage. Figure 6.29 shows an example of the current sensing profile with a conduction angle of approximately 13.5 degrees. The observer operates well in this condition because there is no measurement noise when the current is sensed for the next phase. Since an electrical cycle is 90 degrees, one phase will be turning on at the same moment the previous phase is turning off when the conduction angle is 30 degrees. So as the conduction angle increases, the three noisy regions shown in Figure 6.29 spread apart and the third one approaches the point where the current measurement is to be made. In this condition the sensed current is unreliable. This results in limits on both the supply voltage and conduction angles which could be tested which effectively placed limits on the drive speed. Attempts to employ snubbers to reduce the rate of voltage change were unsuccessful at significantly reducing the coupled noise. The experiments conducted at the speeds at or below 6000 rpm, though, sufficiently demonstrate the stability of the observer for applications at higher speeds.

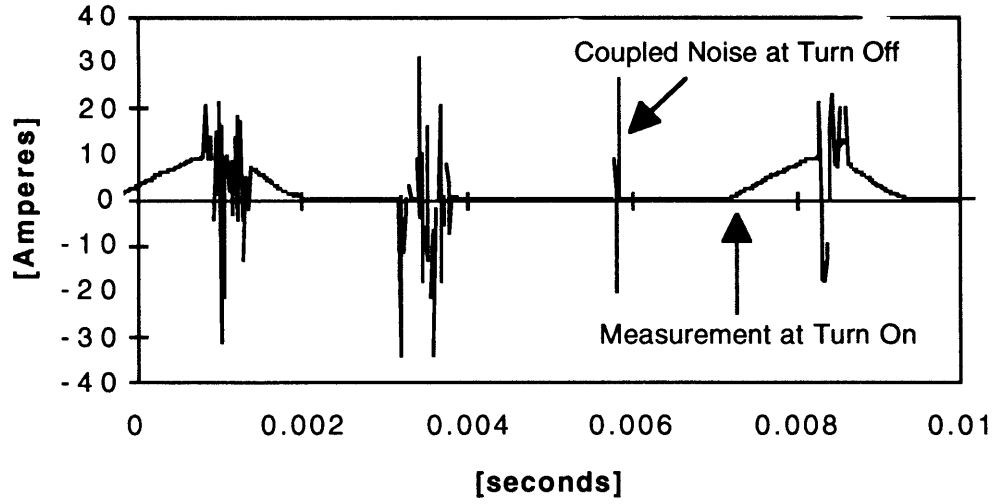

Figure 6.29: Oscilloscope trace of current sensor exhibiting coupled switching noise from each phase being turned off for $\theta_{cond} = 13.5$ degrees.

Though limitations were encountered with the controller that prevented operation at the design speed of 10 krpm, the lower speed results showed significant agreement between the simulated and experimental performance of the observer. Based on this agreement, the simulation can be used to predict the VRM observer operation at 10 krpm as shown in Figures 6.30 through 6.34. Table 6.6 shows the associated simulation parameters. These figures show stable controller regulation within a fraction of a percent of the desired speed using the observer for feedback. Furthermore the observer errors state errors are also fractional.

Table 6.6: Experimental test parameters for speed regulation with the on-line observer shown in Figures 6.30 through 6.34.

Parameter	Units	Value
V_b	[Volts]	160
C	[Nm]	0.0
$\hat{\omega}_{rate\,lim}$	[rad/sec/update]	4
$\theta_{cond\,max}$	[degrees]	45
ω	[rpm]	10000
Ω	[rpm]	10000

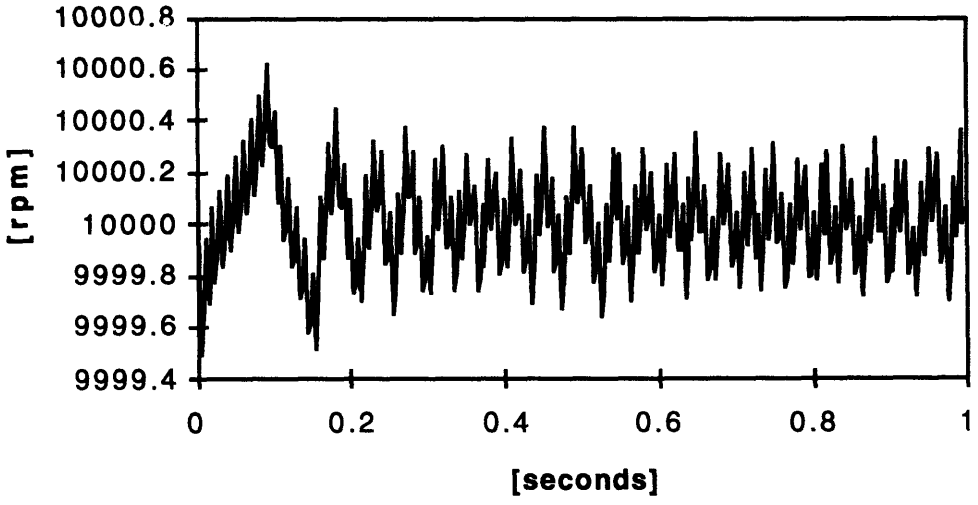

Figure 6.30: Simulation showing the speed regulation at 10 krpm using on-line observer for parameters given in Table 6.6.

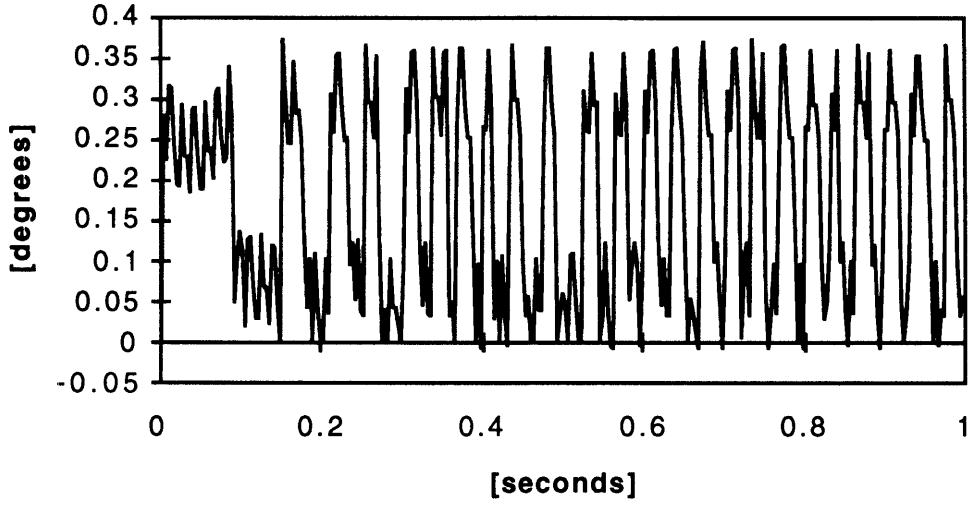

Figure 6.31: Simulation showing the position error at 10 krpm using on-line observer for parameters given in Table 6.6.

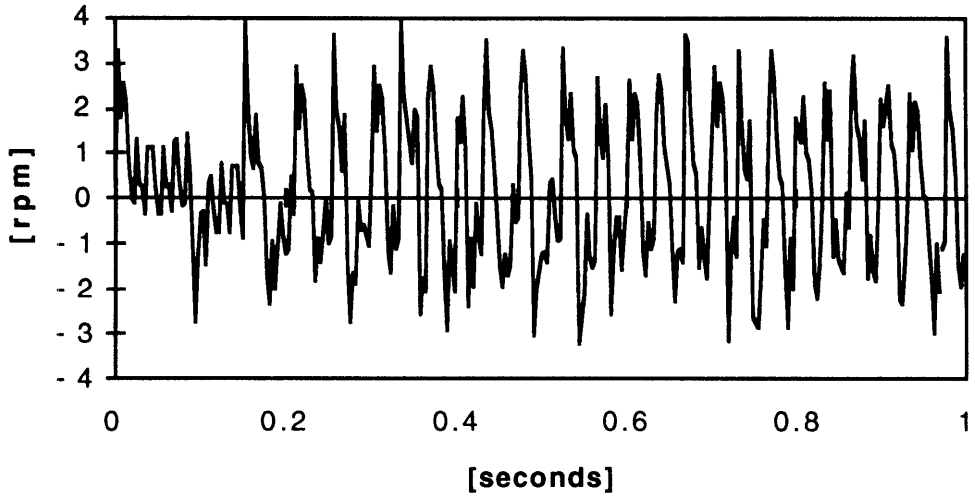

Figure 6.32: Simulation showing the speed error at 10 krpm using on-line observer for parameters given in Table 6.6.

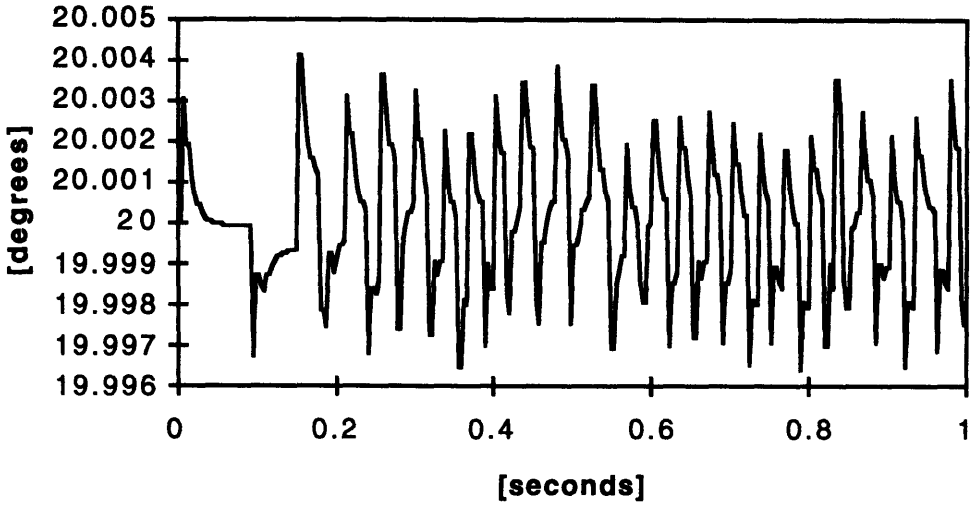

Figure 6.33: Simulation showing the turn on angle at 10 krpm using on-line observer for parameters given in Table 6.6. The turn on angle is computed by the speed controller.

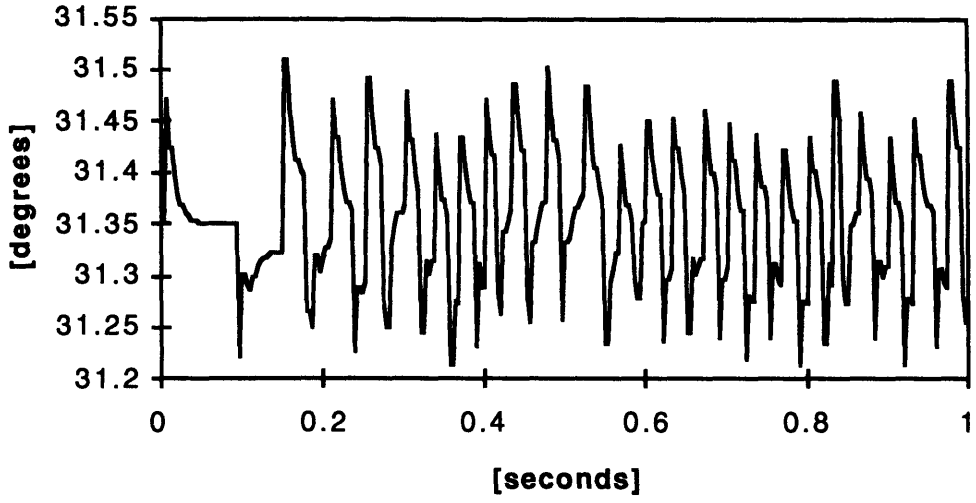

Figure 6.34: Simulation showing the conduction angle at 10 krpm using on-line observer for parameters given in Table 6.6. The conduction angle is computed by the speed controller.

6.5 Summary

In this chapter the predicted performance from the simulations were substantiated through experimental tests using a VRM drive system. The VRM simulation was designed to predict the experimental VRM electrical and mechanical behavior. Using a flexible, fast software system to control the VRM different operating conditions were investigated and data collected which verified the stable operation of the observer, the accuracy of the predicted motor current and torque, and the stable operation of the controller. Experiments at very high speeds and high power conditions were not possible due to switching noise in the electronic hardware.

The experiments verified that the observer rejects discrete disturbances using imposed initial state errors. The transient error response of the experiments closely matched the simulation, thus providing more confidence in using the simulation to predict behavior at other conditions. Furthermore, it was noted that in cases of relatively low accelerating torque the transient behavior of the observer speed state is dominated by the observer term. Thus the precision and update rate of the torque calculations for the observer is of lesser importance for a wide range of applications. The benefit is reduced system complexity either through computation time or reduced memory requirements if torque lookup tables are used. This may not be true, though, in conditions of significant torque imbalance.

The experimental tests also demonstrated that control for transient speed operation could be provided, but only with some restrictions. By restricting turn on angles from operation above 45 degrees, both the double-valued solution and the region of poor position sensing resolution were avoided. Furthermore slew rate limiting allowed for stable transition of the control angles for large setpoint changes. With these restrictions, the observer states accurately tracked the actual states for all the regions of experimental operation.

The drive system operating region was limited, though, due to electrical switching noise. This prevented experiments at the high design point of 10,000 rpm at 2 hp, but experiments at lower power up to 6000 rpm were sufficient to demonstrate that the observer provided stable VRM control with just one sample for each commutation. This stable performance was demonstrated to successfully reject discrete state errors and accurately track the actual states in the presence of torque prediction errors. To complete the results, a simulation was conducted at 10,000 rpm to demonstrate the predicted performance of the observer and controller. Given this experimental verification the simulation predictions are reviewed in the final chapter, and areas for further consideration are discussed.

Chapter 7

CONCLUSIONS AND RECOMMENDATIONS

7.1 Introduction

This thesis studied the simulation and experimental verification of a position observer for a variable-reluctance motor drive. Building on previous modeling, control and observer design research presented in Chapter 2, an observer-based VRM drive was successfully designed and demonstrated. Chapters 3 through 5 developed the VRM model, controller, and observer through theory and verified them by simulations. The VRM electrical model and associated simulations were based on the experimental drive presented in Chapter 6 thus providing a direct means of comparison. The performance accurately matched the simulation expectations thus increasing the confidence in utilizing the simulation for other applications and operating regions. The sections following this introduction summarize the major contributions of this thesis. Limitations that were encountered are also revisited concurrently, as well as related areas for further investigation.

The previous work introduced in Chapter 2 described the VRM model, controller, and observer bases for this thesis. The VRM model and speed controller developed by Torrey [3] provided the characteristic form upon which to build the observer. The nonlinear flux linkage algorithm created a more accurate representation of the VRM behavior both in linear and saturated operating regions. The observer designs from Harris [2] and Lumsdaine [1] showed a simple method for calculating rotor position from phase currents. The computational complexity and overhead for Lumsdaine's continuous observer was deemed prohibitive for high-speed VRM operation. Simplifications developed by Harris showed that acceptable speed control performance could be achieved without a continuous full-state observer model. This thesis extended Harris' theoretical design by eliminating restrictions imposed by probing unexcited

phases at specific rotor alignment positions, and by implementing this observer design in both simulations and hardware experiments.

Chapter 3 presented the electrical model of the VRM. The model represented the periodic nonlinear characteristics of the flux linkage relationship to instantaneous current using an analytic function with periodic coefficients. As shown by Torrey this model provided advantages over linear and piecewise linear models with respect to accurately predicting instantaneous current and therefore torque.

The transient speed controller in Chapter 4 was developed similarly to Torrey's. Since the chief focus of Torrey's work was on optimal excitation, the stability analysis was focused around these optimal operating points. A more generalized method was employed here to describe regions of operation where stable gains were developed based on sufficient conditions. The controller form chosen was a proportional plus integral compensator driven by the speed error between a reference and a feedback VRM speed signal. Following a linearized stability analysis, simulations were presented which confirmed the asymptotic decay of the speed error. Figures 4.4 through 4.7 showed a typical response which was overdamped with a settling time of approximately 3.5 seconds.

The observer, which is the chief contribution of this thesis, was developed in Chapter 5. The observer design extended Harris' research by demonstrating that rotor position could be determined by measuring currents of the active phase rather than probing unexcited phases. The commutating phase's current was sensed after turn on and then inverted to rotor position based on the electrical model. Intermediate rotor positions were interpolated using a mechanical dynamics model without innovation terms until the next phase commutation. Furthermore, unlike the Harris observer, the sensing angles were not restricted to a specific location such as maximum misalignment. Stability analysis was conducted based on a Kalman filter form incorporating the effects of interpolating without the innovation term between current measurements. The closed-loop poles calculated from the steady-state Kalman filter gains were corroborated by transient simulations shown in Figures 5.6 through 5.8. Furthermore, Figures 5.9 through 5.12 showed minimal interaction between the observer and controller dynamics when the observer was configured to provide the feedback states for the speed controller. The dynamic response of the error state vector was asymptotically stable and overdamped with a settling time of approximately 30 msec. The predicted steady-state accuracy from the 'on-line' simulations was 4.84 rpm and 0.39 degrees rms.

The model, controller, and observer aspects of the simulation were corroborated by the experimental drive tests presented in Chapter 6. The drive system consisted of the

6-4 VRM, a DC brake, inverters, interface electronics, and a DSP-based controller. The observer stability and disturbance rejection was compared and shown to have similar performance characteristics. The settling times of the simulation and experimental observer errors were approximately equal. The steady-state observer errors for the experimental drive were 6.03 rpm and 1.03 degrees rms. The experimental error was not significantly higher than the simulation predictions, and was attributed primarily to the accuracy of the position estimation from phase current sensors and the predicted $\lambda - i - \theta$ relationship. Tests were also conducted to verify the accuracy of the model by examining simulated and actual current profiles for given control angles and speed. These tests showed a typical accuracy within 10% of the measured instantaneous phase current. Finally the controller stability and response trajectory was corroborated by similar transient speed tests. Though the settling times were approximately equal, the experimental response trajectory was different due to inaccurate modeling of the drive load. Furthermore limitations of the experiment and their impact on the controller were discussed, and are presented in the following three sections in the context of suggestions for further study.

7.2 Estimator Performance

Estimator performance demonstrated the viability of sensorless control for high-speed VRM drives. Using a VRM designed for operation up to 10 krpm and 2 hp, simulations predicted stable behavior of the observer and observer-based speed control with discrete disturbances and during large transients. The experimental drive corroborated these results at low power conditions (less than 0.15 hp as calculated from Figure 6.24) up to 6 krpm as demonstrated in Figures 6.25 through 6.28. Switching noise in the hardware prevented operation at higher speeds and load conditions, but due to the close agreement of the simulations with the laboratory tests conducted it is reasonable to assume the simulation extensions are accurate.

The experimental drive was limited because switching noise coupled to the current sensing and drive signals. As described in Chapter 6, when a phase was turned off, the continuous flyback current resulted in rapid switching of the voltage across the phase. The voltage switching was coupled to the commutation command and current sensing signals of all the phases. Though this caused unintended chopping transients, the more significant effect was that the conduction angles were limited to less than 30 degrees. This limited the ultimate speed to 6 krpm with the experimental system. The torque output at this speed was limited by the low supply voltage, 68V, and the maximum

conduction angle which was 25 degrees which included enough time margin for the switching noise to decay before the next phase turned on.

The experimental system limitations may be overcome through more investigation of the noise coupling sources. Since the observer performance was investigated in the regions where noise did not affect measurements, the results presented are independent of the switching noise phenomena. Therefore the thesis results should be freely extendible to applications at higher speeds and operating conditions.

Additionally, there were other limitations observed with the experimental tests. Due to the shape of the position versus current relationship, the position sensing accuracy was not constant over the range of sensed currents and therefore turn on angles. The sensitivity was highest when sensing near maximum misalignment (45 degrees for the 6-4 VRM) which resulted in increased steady-state error when sensing current at 45 degrees. Furthermore, since there was a double-valued solution for positions symmetric around misalignment, the algorithm needed to determine which solution was correct. The simple method of discrimination based on the expected angle worked for steady-state operation, but was unreliable for transient operation where the turn on angle transitions from below to above 45 degrees or vice versa. The same problem occurs near maximum alignment (0 degrees), and in fact, the low sensing accuracy region spans a larger position range near alignment. Alignment, though, is not a likely turn angle for the majority of VRM drive applications.

Operation at the turn angle that produces maximum average torque, though, was in the region of highest sensing accuracy. Thus, restricting turn on angles while maintaining flexibility of the conduction angles would not limit drive output for either acceleration or deceleration. On the other hand, though, for the observer design to become widely applicable both an improvement in sensing accuracy near misalignment and a discrimination algorithm should be developed. There are methods for future study that appear promising, some of which have been investigated in other studies, and which are described below.

The threshold detection scheme discussed by Harris that was used to detect maximum misalignment could be modified for the purpose of variable turn on angle operation. For instance taking two successive measurements on the commutating phase would indicate whether the instantaneous inductance is increasing or decreasing thus providing discrimination. Another method is to sense the inductance on two phases simultaneously either by sensing a higher current on a commutating phase or pulsing an unexcited phase. Due to the fixed angular relationship between the phases, the pair of currents would completely specify the position within an electrical cycle. For generic application both the commutating and pulsed cases must be considered. The condition

with all three phases excited simultaneously on a VRM with the turn on angle near 45 degrees, though, could easily be avoided resulting in a simpler detection scheme. All three phases will be excited simultaneously if the conduction angle is greater than 30 degrees. This represents a higher power operating condition resulting in drive acceleration in many applications. Since the control angle condition resulting in the highest possible accelerating torque corresponds to a turn-on angle lower than 45 degrees, the turn on angle could be designed to always be below 45 degrees during drive acceleration without any loss of torque output flexibility.

To avoid the region of low sensing accuracy, measurement on another phase could be used when turn on angles are near 45 degrees. Alternately, the primary phase sensing could be delayed further also resulting in higher accuracy.

Finally, throughout this thesis the only mode considered was motoring operation. In theory, the observer characteristics should be identical in generation. As with motoring, though, certain limitations would surely be encountered. For instance, in order to maintain stable observer operation it may be necessary to maintain minimum conduction angles. On the other hand, for many generation applications the performance requirements will be different. Closed loop speed control will not be as much an issue as closed loop load control in generation.

7.3 VRM Modeling and Driving Torque

A relatively sophisticated electrical model was developed and used throughout this thesis. Using a nonlinear function that was periodic with respect to position, Equation 2.1, the instantaneous current was accurately modeled and utilized in simulations in Chapter 3. Comparison to experimental measurements of current in Chapter 6 showed close agreement. This model was used in the experimental system, as described in Chapter 6, to calculate average torque for the observer, and to produce a lookup table for the position calculation from measured current.

Results indicate that the relatively complex torque model may not be necessary for a wide variety of applications. Due to processing limitations in the experimental system, the average torque calculation was conducted off line from the observer and control software. It was noted that the evolution of the observer speed state was dominated by the innovation term, not the mechanical dynamics. Thus the conclusion was made that the model of average torque need not be highly accurate.

This conclusion, though, cannot be extrapolated to conditions of significant driving, or accelerating, torque. It is specifically the torque imbalance, and not simply the electrical torque, that is of interest since it is the imbalance that drives the mechanical dynamics of the motor. As the torque imbalance increases, the relative dominance of the observer and mechanical acceleration terms (see Figure 6.17) may switch. The data in Chapter 6 does not investigate the limits of stability with respect to driving torque error. This represents a potential area for further study.

7.4 Controller Dynamic Stability

The controller stability analysis developed in Chapter 4 was based on linearized design at constant speed. This type of solution assumes relatively slow mechanical dynamics. Implicit in this assumption is that the speed errors are small. For large speed setpoint changes this issue was handled by implementing slew rate limiting on the control angles. The appropriate slew rate limit that insured stable operation was determined experimentally. Simulations demonstrated the stable operation of the controller even over large speed setpoint changes. In Chapter 6, the controller was further demonstrated to operate similarly using the observer states to provide the feedback terms for control angle calculations and commutation.

If the effects of slew rate limiting are examined more closely, it can be shown that they alter the dynamic system behavior. By limiting the control angle transition each update for a given speed error another layer of control integration is added. Failure of the experimental system to operate successfully without the slew limit shows that the proportional gain is clearly inappropriate for large speed errors even with limits on the ultimate control angles. Instead of determining the appropriate limit experimentally, a dynamic stability analysis should be considered in further research.

7.5 Summary

This thesis demonstrates that with limited sampling a stable observer can be achieved for transient speed operation. It may not even be necessary for a sophisticated electrical model depending on the torque imbalance magnitudes and mechanical dynamics of the system. Furthermore the observer is capable of operating at variable control angles. Through further research specific limitations on the stable operating conditions may be

eliminated and operation in generation can be demonstrated. Specific areas for further research that were presented in the previous three sections are summarized as follows:

1. Experimentally demonstrate the observer under higher load, disturbance, and speed conditions.

2. Develop a more accurate sensing scheme when commutating near maximum misalignment.

3. Develop a simulation and experiment for the observer when operating as a generator.

4. Investigate the minimum electrical model requirements as a function of torque imbalance and disturbances.

5. Extend the controller analysis to consider the effects of limiting the controller action during large transients.

The VRM observer developed in this thesis can successfully be applied to high-speed drive and generation applications. The observer is particularly appropriate for situations where inclusion of a position sensing system is a high premium. The advantage of eliminating the sensor is potentially higher reliability and lower total cost. Of course, the additional complexity of the observer must be considered in any such application. With simple control situations the observer can be easily implemented in analog hardware. Further complications of variable speed, commutation angles, and loads may require NVM lookup tables for robust application or computer processing in the most general cases. When implementing this observer-based controller in hardware, the speed limiting factor becomes the eddy currents. The current measurements and position inversion will not be accurate until the eddy currents decay. The decay time for the experimental VRM, as calculated by Harris [2], is 6.9 μsec. Even if conduction is limited to a minimum of 15 μsec to provide a margin of safety, this results in a theoretical VRM speed limit of 500 krpm with the observer-based controller. Therefore, when the observer is implemented in hardware, it is no longer a potential speed limiting factor and speeds in excess of 100 krpm could be achieved. In the case of propulsion systems, though, the drive and generator systems are now more typically computer-controlled and therefore the limitation becomes the processing overhead. Thus the major contribution of this thesis is demonstrating a method to minimize the processing requirements for very high speed applications.

Appendix A

SIMULATION SOFTWARE

This appendix lists all the simulation software modules used in this thesis. It includes various modeling engines for capturing different types of data such as flux-maps, constant speed current profiles, transient speed simulations with an off-line observer, and with an on-line observer. All the corresponding header and data files are also provided. All the software was written using the C programming language [16].

TRANSIENT.C	Main simulation file containing nine simulation engines for different tasks.
NRUTIL.C	Software utility file modified from routines in [4].
GAINS.DAT	Data file for controller PI gains and observer Kalman filter gains.
MOTORDATA	Data file for $\lambda - i - \theta$ function coefficients.
MOTOR.H	Header file for simulation VRM drive constant names.
PARAM.SIM	Data file for simulation VRM drive constant values.

TRANSIENT.C

```
#include <stdio.h>
#include <math.h>
#define pi  3.141592654
double delta_theta;  /* angular increment of
simulation (degrees) */
double R_p;
double R_s;
double turns;
double tol;
double V_b;
double J;
double B;
double T_s;
double om_h_s;
double maxslew;
double coulombic;
double maxcond;
double nom_on,nom_cond;
main()
{
  char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
  double **dmatrix();
  int sim_type;
  A=dmatrix(1,4,1,31);
  inputs (gains,controls,speeds,times,name,A);
  printf("\n1 - transient torrey simulation\n");
  printf("2 - controller map at constant
speed\n");
printf("3 - partial derivatives at a given operating
point\n"); printf("4 - transient average torque
simulation\n"); printf("5 - transient
simulation\n");
```

```
  printf("6 - current/angle estimation table\n");
  printf("7 - static torque\n");
  printf("8 - off line transient estimator\n");
printf("9 - on line transient estimator\n");
printf("10 - one electrical cycle\n");
printf("\nDesired simulation: ");
scanf("%d",&sim_type);
  switch(sim_type)
  {
    case 1:
motor_dynamics1
(controls,gains,speeds,times,name,A);     break;
    case 2:
motor_dynamics2
(controls,gains,speeds,times,name,A);     break;
    case 3:
motor_dynamics3
(controls,gains,speeds,times,name,A);     break;
    case 4:
motor_dynamics4
(controls,gains,speeds,times,name,A);     break;
    case 5:
motor_dynamics5
(controls,gains,speeds,times,name,A); break;
    case 6:
motor_dynamics6
(controls,gains,speeds,times,name,A);    break;
    case 7:
motor_dynamics7
(controls,gains,speeds,times,name,A);     break;
    case 8:
motor_dynamics8
(controls,gains,speeds,times,name,A);     break;
    case 9:
motor_dynamics9
(controls,gains,speeds,times,name,A);     break;
    case 10:
motor_dynamics10
(controls,gains,speeds,times,name,A);     break;
  }
  free_dmatrix(A,1,4,1,31);
  printf ("Got this far\n");
}
/************************************/
motor_dynamics1
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count; double
theta_on,theta_cond;
  int C[3],j,phase,phases;
  FILE *fopen(),*fp;
  double temp;
  double imax;
  double command,intgrl;
  int cflag;
  double *a;
  double *dvector(),ilambda();
  double dtime;
  int ilimit,idelta,tdelta;
  double *deriv;
  printf ("What is the initial turn on angle (deg)?
");
  scanf ("%lf",&temp);
  controls[0] = temp;
  printf ("What is the initial turn off angle (deg)?
");
  scanf ("%lf",&temp);
  controls[1] = temp - controls[0];
  printf ("What is the initial rotor speed (rpm)?
");
  scanf ("%lf",&temp);
  speeds[0] = pi * temp/30.0;
  printf ("What is the final rotor speed (rpm)? ");
  scanf ("%lf",&temp);
  speeds[1] = pi * temp/30.0;
  printf ("What is the total time of interest (s)?
");
  scanf ("%lf",&temp);
  times[1] = temp;
  printf ("How many data points are desired (#)?
");
  scanf ("%lf",&temp);
  times[2] = times[1] / temp;
  printf ("Where do you want the simulation
results stored? ");
  scanf ("%s",name);
  T_avg = 0;
  T_motor = 0;
  T_count = 0;
  T = 0;
  T_cycle = 0;
speed = speeds[0];
times[3] = 0;
times[4] = 0;
times[5] = 0;
intgrl = 0.0;
command = 0.0;
cflag = 0;
angle = 0;     for(phase=0;phase<3;phase++)
{
  lambda[phase] = 0.0;
  C[phase] = 0.0;
}
phase = 0;
phases = 3;
theta_on = controls[0];
theta_cond = controls[1];
T_cycle = (delta_theta*pi/180)/speed;
do
```

```
{
if(fmod(angle,90.0) == 0.0) {
    if (times[5] >= times[3])
    {
store_data
(controls,speed,times,command,T_motor,name);
times[3] = times[3] + times[2];
printf("\nThe time is: %lf\n", times[5]);
printf("angle is: %lf\n", angle); printf("driving
torque: %lf\n",T_avg); printf("new speed:
%lf\n",(30.0*speed/pi));
    }
if (times[4] >= (times[0] - 0.5*(pi/(2.0*speed))))
{
      printf ("updating excitation at %lf
s.\n",times[5]);
controller1
(controls,gains,speed,speeds,&intgrl,times,&co
mmand);
      printf ("integral: %lf\n",intgrl);
      times[4] = 0.0;
    }
  }
  T_motor = 0;
  for(phase=0;phase<3;phase++)
  {
theta[phase] = fmod(angle + phase*60.0,90.0);
if((fmod(controls[0]+controls[1],90.0)<controls[
0]) && (theta[phase]<(controls[0] +
controls[1])))
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
else if((theta[phase]>controls[0]) &&
(theta[phase]<fmod(controls[0]+controls[1],90.0
)))
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
    else if(lambda[phase] > 0.0)
    {
      C[phase]=0;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
    }
    else
    {
      T = 0;
C[phase]=0; lambda[phase]=0; i=0;
    }
  T_motor = T_motor + T; }
T_count = T_count + T_motor;
if((fmod(angle,90.0) == 0.0) && times[5] != 0.0)
{
T_avg = T_count/(90.0/delta_theta); T_count = 0;
    }
    speed = speed + T_cycle * (T_motor -
B*speed)/J;
T_cycle = (delta_theta*pi/180)/speed;
times[5] = times[5] + T_cycle;
times[4] = times[4] + T_cycle; angle = angle +
delta_theta;
  }
while(times[5] <= times[1]);
  return;
}
/************************************/
motor_dynamics2
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count; double
theta_on,theta_cond;
  int C[3],j,phase,phases;
  FILE *fopen(),*fp;
  double temp;
  double imax;
  double command,intgrl;
  int cflag;
  double *a;
  double *dvector(),ilambda();
  double dtime;
  int ilimit,idelta,tdelta;
  double *deriv;
  printf ("What is the rotor speed (rpm)? ");
  scanf ("%lf",&temp);
  speeds[0] = pi * temp/30.0;
printf ("Where do you want the simulation results
stored? ");
  scanf ("%s",name);
  T_avg = 0;
  T_motor = 0;
  T_count = 0;
  T = 0;
  T_cycle = 0;
  speed = speeds[0];
  cflag = 0;
  for(phase=0;phase<3;phase++)
  {
    lambda[phase] = 0.0;
    C[phase] = 0.0;
  }
  phase = 0;
  phases = 3;
  fp = fopen(name,"w");
  fprintf(fp,"On_Angle\tCond_Angle\tAvg_Torq
ue\tMaxCurrent\n");

  for(theta_cond=5;theta_cond<=45;theta_cond
=theta_cond+5)
  {
```

```
  for(theta_on=0;theta_on<90;theta_on=theta_
on+5)
    {
imax = 0;
for(theta[phase]=0;theta[phase]<180;theta[phase
]=theta[phase]+delta_theta)
      {
if((theta[phase]>theta_on) &&
(theta[phase]<(theta_on + theta_cond)))
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
        else if(lambda[phase] > 0.0)
        {
C[phase] = 0;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
        }
        T_avg = T_avg + T;
        if(i>imax)
          imax=i;
      }
      T_avg = phases*T_avg*delta_theta/90.0;
fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",theta_on,theta_
cond,T_avg,imax);
printf("%lf\t%lf\t%lf\t%lf\n",theta_on,theta_con
d,T_avg,imax);
    }
  }
  fclose(fp);
  return;
}
/************************************/
motor_dynamics3
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
  double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count;
  double theta_on,theta_cond;
  int C[3],j,phase,phases;
  FILE *fopen(),*fp;
  double temp;
  double imax;
  double command,intgrl;
  int cflag;
  double *a;
  double *dvector(),ilambda();
  double dtime;
  int ilimit,idelta,tdelta;
  double *deriv;
  printf ("What is the nominal turn on angle
(deg)? ");
  scanf ("%lf",&temp);
  controls[0] = temp;
  printf ("What is the nominal turn off angle
(deg)? ");
  scanf ("%lf",&temp);
  controls[1] = temp - controls[0];
  printf ("What is the nominal rotor speed (rpm)?
");
  scanf ("%lf",&temp);
  speeds[0] = pi * temp/30.0;
  printf ("Where do you want the simulation
results stored? ");
  scanf ("%s",name);
T_avg = 0;
T_motor = 0;
T_count = 0;
T = 0;
T_cycle = 0;
speed = speeds[0];
times[3] = 0;
times[4] = 0;
times[5] = 0;
intgrl = 0.0;
command = 0.0;
cflag = 0;
angle = 0;     for(phase=0;phase<3;phase++)
{
  lambda[phase] = 0.0;
  C[phase] = 0.0;
}
phase = 0;
phases = 3;
theta_on = controls[0];
theta_cond = controls[1];
T_cycle = (delta_theta*pi/180)/speed;
fp = fopen(name,"w");
fprintf(fp,"theta_on\ttheta_cond\tspeed\tT_avg\
n");
for(j=0;j<=2;j++)
{
  theta_on = controls[0]*(1+.01*(-1+j));
  for(theta[phase]=0;theta[phase]<180;theta[pha
se]=theta[phase]+delta_theta)
  {
if((theta[phase]>theta_on) &&
(theta[phase]<(theta_on + theta_cond))) {
  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A); }
    else if(lambda[phase] > 0.0)
    {
C[phase] = 0;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
    }
    T_avg = T_avg + T;
  }
  T_avg = phases*T_avg*delta_theta/90.0;
```

```
fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",theta_on,theta_
cond,speed,T_avg);
  printf("%lf\t%lf\t%lf\t%lf\n",theta_on,theta_c
ond,speed,T_avg);
}
theta_on = controls[0];
for(j=0;j<=2;j++)
{
  theta_cond = controls[1]*(1+.01*(-1+j));
  for(theta[phase]=0;theta[phase]<180;theta[pha
se]=theta[phase]+delta_theta)
  {
if((theta[phase]>theta_on) &&
(theta[phase]<(theta_on + theta_cond))) {
  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A); }
    else if(lambda[phase] > 0.0)
    {
C[phase] = 0;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
    }
      T_avg = T_avg + T;
    }
    T_avg = phases*T_avg*delta_theta/90.0;
fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",theta_on,theta_
cond,speed,T_avg);
printf("%lf\t%lf\t%lf\t%lf\n",theta_on,theta_con
d,speed,T_avg);
  }
  theta_cond = controls[1];
  for(j=0;j<=2;j++)
  {
    speed = speeds[0]*(1+.01*(-1+j));

  for(theta[phase]=0;theta[phase]<180;theta[pha
se]=theta[phase]+delta_theta)
    {
if((theta[phase]>theta_on) &&
(theta[phase]<(theta_on + theta_cond))) {
  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A); }
      else if(lambda[phase] > 0.0)
      {
C[phase] = 0;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
      }
      T_avg = T_avg + T;
    }
    T_avg = phases*T_avg*delta_theta/90.0;
fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",theta_on,theta_
cond,speed,T_avg);
printf("%lf\t%lf\t%lf\t%lf\n",theta_on,theta_con
d,speed,T_avg);
  }
  fclose(fp);
  return;
}
/************************************/
motor_dynamics4
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
  double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count;
  double theta_on,theta_cond;
  int C[3],j,phase,phases;
  FILE *fopen(),*fp;
  double temp;
  double imax;
  double command,intgrl;
  int cflag;
  double *a;
  double *dvector(),ilambda();
  double dtime;
  int ilimit,idelta,tdelta;
  double *deriv;
  printf ("What is the initial turn on angle (deg)?
");
  scanf ("%lf",&temp);
  controls[0] = temp;
  printf ("What is the initial turn off angle (deg)?
");
  scanf ("%lf",&temp);
  controls[1] = temp - controls[0];
  printf ("What is the initial rotor speed (rpm)?
");
  scanf ("%lf",&temp);
speeds[0] = pi * temp/30.0;
printf ("What is the final rotor speed (rpm)? ");
scanf ("%lf",&temp);
speeds[1] = pi * temp/30.0;
printf ("What is the total time of interest (s)? ");
scanf ("%lf",&temp);
times[1] = temp;
printf ("How many data points are desired (#)? ");
scanf ("%lf",&temp);
times[2] = times[1] / temp;
printf ("Where do you want the simulation results
stored? "); scanf ("%s",name);
T_avg = 0;
T_motor = 0;
T_count = 0;
T = 0;
T_cycle = 0;
speed = speeds[0];
times[3] = 0;
times[4] = 0;
times[5] = 0;
intgrl = 0.0;
```

```
command = 0.0;
cflag = 0;
angle = 0;
for(phase=0;phase<3;phase++)
{
  lambda[phase] = 0.0;
  C[phase] = 0.0;
}
phase = 0;
phases = 3;
theta_on = controls[0];
theta_cond = controls[1];
T_cycle = (delta_theta*pi/180)/speed;
do
{
if(fmod(angle,90.0) == 0.0) {
     if (times[5] >= times[3])
     {
store_data
(controls,speed,times,command,T_motor,name);
times[3] = times[3] + times[2];
printf("\nThe time is: %lf\n", times[5]);
printf("angle is: %lf\n", angle); printf("driving
torque: %lf\n",T_avg); printf("new speed:
%lf\n",(30*speed/pi));
     }
if (times[4] >= (times[0] - 0.5*(pi/(2*speed)))) {

        printf ("updating excitation at %lf
s.\n",times[5]);
controller1
(controls,gains,speed,speeds,&intgrl,times,&co
mmand);
        printf ("integral: %lf\n",intgrl);
        times[4] = 0.0;
     }
  }
  T_motor = 0;
  for(phase=0;phase<3;phase++)
  {
theta[phase] = fmod(angle + phase*60.0,90.0);
if((fmod(controls[0]+controls[1],90.0)<controls[
0]) && (theta[phase]<(controls[0] +
controls[1])))

  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A);
else if((theta[phase]>controls[0]) &&
(theta[phase]<fmod(controls[0]+controls[1],90.0
)))
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
        else if(lambda[phase] > 0.0)
        {
          C[phase]=0;

  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A);
        }
        else
        {
          T = 0;
          C[phase]=0;
          lambda[phase]=0;
          i=0;
        }
        T_motor = T_motor + T;
     }
T_count = T_count + T_motor;
if((fmod(angle,90.0) == 0.0) && times[5] != 0.0)
{
T_avg = T_count/(90.0/delta_theta); T_count = 0;
     }
     if(fmod(angle,90.0) == 0.0)
       speed = speed + ((pi/2.0)/speed) * (T_avg -
B*speed)/J;
     T_cycle = (delta_theta*pi/180.0)/speed;
     times[5] = times[5] + T_cycle;
times[4] = times[4] + T_cycle; angle = angle +
delta_theta;
  }
while(times[5] <= times[1]);
  return;
}
/************************************/
motor_dynamics5
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count; double
theta_on,theta_cond;
  int C[3],j,phase,phases;
  FILE *fopen(),*fp;
  double temp;
  double imax;
  double command,intgrl;
  int cflag;
  double *a;
  double *dvector(),ilambda();
  double dtime;
  int ilimit,idelta,tdelta;
  double *deriv;
  printf ("What is the initial turn on angle (deg)?
");
  scanf ("%lf",&temp);
  controls[0] = temp;
  printf ("What is the initial turn off angle (deg)?
");
```

```
   scanf ("%lf",&temp);
   controls[1] = temp - controls[0];
printf ("What is the initial rotor speed (rpm)? ");
   scanf ("%lf",&temp);
   speeds[0] = pi * temp/30.0;
printf ("What is the final rotor speed (rpm)? ");
   scanf ("%lf",&temp);
   speeds[1] = pi * temp/30.0;
printf ("What is the total time of interest (s)? ");
   scanf ("%lf",&temp);
   times[1] = temp;
printf ("How many data points are desired (#)? ");
   scanf ("%lf",&temp);
   times[2] = times[1] / temp;
printf ("Where do you want the simulation results
stored? "); scanf ("%s",name);
   T_avg = 0;
   T_motor = 0;
   T_count = 0;
   T = 0;
   T_cycle = 0;
   speed = speeds[0];
   times[3] = 0;
   times[4] = 0;
   times[5] = 0;
   intgrl = 0.0;
   command = 0.0;
   cflag = 0;
   angle = 0;
   for(phase=0;phase<3;phase++)
   {
     lambda[phase] = 0.0;
     C[phase] = 0.0;
   }
   phase = 0;
   phases = 3;
   theta_on = controls[0];
   theta_cond = controls[1];
   T_cycle = (delta_theta*pi/180)/speed;
   do
   {
if(fmod(angle,90.0) == 0.0) {
        if (times[5] >= times[3])
        {
store_data
(controls,speed,times,command,T_motor,name);
times[3] = times[3] + times[2];
printf("\nThe time is: %lf\n", times[5]);
printf("angle is: %lf\n", angle); printf("driving
torque: %lf\n",T_avg); printf("new speed:
%lf\n",(30.0*speed/pi));
        }
/*        if (times[4] >= (times[0] -
0.5*(pi/(2.0*speed)))) */
        if (times[4] >= times[0])
        {
printf ("updating excitation at %lf
s.\n",times[5]); controller2
(controls,gains,speed,speeds,&intgrl,times,&co
mmand); printf ("integral: %lf\n",intgrl);
          times[4] = 0.0;
        }
     }
     T_motor = 0;

     for(phase=0;phase<3;phase++)
     {
theta[phase] = fmod(angle + phase*60.0,90.0);
if((fmod(controls[0]+controls[1],90.0)<controls[
0]) && (theta[phase]<(controls[0] +
controls[1])))
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
else if((theta[phase]>controls[0]) &&
(theta[phase]<fmod(controls[0]+controls[1],90.0
)))
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i,A);
        else if(lambda[phase] > 0.0)
        {
          C[phase]=0;

  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A);
        }
        else
        {
          T = 0;
          C[phase]=0;
          lambda[phase]=0;
          i=0;
        }
        T_motor = T_motor + T;
     }
T_count = T_count + T_motor;
if((fmod(angle,90.0) == 0.0) && times[5] != 0.0)
{
T_avg = T_count/(90.0/delta_theta); T_count = 0;
     }
     speed = speed + T_cycle * (T_motor -
B*speed - 0.7125)/J;
T_cycle = (delta_theta*pi/180)/speed;
times[5] = times[5] + T_cycle;
times[4] = times[4] + T_cycle; angle = angle +
delta_theta;
   }
while(times[5] <= times[1]);
   return;
}
/************************************/
motor_dynamics6
(controls,gains,speeds,times,name,A)
char name[25];
```

```
double
controls[2],gains[6],speeds[2],times[6],**A;
{
double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count; double
theta_on,theta_cond;
  int C[3],j,phase,phases;
  FILE *fopen(),*fp;
  double temp;
  double imax;
  double command,intgrl;
  int cflag;
  double *a;
  double *dvector(),ilambda();
  double dtime;
  int ilimit,idelta,tdelta;
  double *deriv;
printf ("Where do you want the simulation results
stored? ");
  scanf ("%s",name);
  T_avg = 0;
  T_motor = 0;
  T_count = 0;
  T = 0;
  T_cycle = 0;
speed = speeds[0];
  times[3] = 0;
  times[4] = 0;
  times[5] = 0;
  intgrl = 0.0;
  command = 0.0;
  cflag = 0;
angle = 0;      for(phase=0;phase<3;phase++)
  {
lambda[phase] = 0.0; C[phase] = 0.0;
  }
  phase = 0;
  phases = 3;
theta_on = controls[0]; theta_cond = controls[1];
T_cycle = (delta_theta*pi/180)/speed;
  a=dvector(1,3);
  fp = fopen(name,"w");
  tol = 0.00001;
printf("Input estimation time in degrees: ");
scanf("%lf",&dtime);
printf("Input speed in rpm: ");
scanf("%lf",&speed);
  dtime = dtime/(360*speed/60);
  printf("%lf seconds\n",dtime);
  printf("%lf Webers\n",dtime*V_b);
fprintf(fp,"Angle\tCurrent\n");
for(angle=0;angle<=45;angle=angle+delta_theta)
{
    coefficient (a,A,angle);
    i = ilambda (dtime*V_b,a);
    fprintf(fp,"%lf\t%lf\n",angle,i);
  }
  free_dvector(a,1,3);
  fclose(fp);
  return;
}
/***********************************/
motor_dynamics7
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count; double
theta_on,theta_cond;
  int C[3],j,phase,phases;
  FILE *fopen(),*fp;
  double temp;
  double imax;
  double command,intgrl;
int cflag;
double *a;
double *dvector(),ilambda();
double dtime;
int ilimit,idelta,tdelta;
double *deriv;
printf ("Where do you want the simulation results
stored? "); scanf ("%s",name);
T_avg = 0;
T_motor = 0;
T_count = 0;
T = 0;
T_cycle = 0;
speed = speeds[0];
times[3] = 0;
times[4] = 0;
times[5] = 0;
intgrl = 0.0;
command = 0.0;
cflag = 0;
angle = 0;
for(phase=0;phase<3;phase++)
{
  lambda[phase] = 0.0;
  C[phase] = 0.0;
}
phase = 0;
phases = 3;
theta_on = controls[0];
theta_cond = controls[1];
T_cycle = (delta_theta*pi/180)/speed;
a=dvector(1,3);
deriv=dvector(1,3);
printf("\nCurrent limit: "); scanf("%d",&ilimit);
printf("\nCurrent delta: "); scanf("%d",&idelta);
printf("\nAngle delta: "); scanf("%d",&tdelta);
```

```
fp = fopen(name,"w");
fprintf(fp,"Angle\tCurrent\tFlux\tTorque\n");
for(i=0;i<=ilimit;i=i+idelta)
{
for(angle=0;angle<=45;angle=angle+tdelta) {
coefficient (a,A,angle);      coeff_deriv
(deriv,A,angle);
lambda[0] = a[1]*(1-exp(a[2]*i))+a[3]*i;
torque (i,angle,&T,a,deriv);
fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",angle,i,lambda[
0],T);
  }
}
free_dvector(a,1,3);
  free_dvector(deriv,1,3);
  fclose(fp);
  return;
}
/************************************/
motor_dynamics8
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
  double
theta[3],speed,lambda[3],T,i[3],T_motor,angle,T
_cycle,T_avg,T_count;
  double theta_on,theta_cond;
  double command,intgrl;
double
eT_motor,etheta[3],eangle,espeed,elambda[3],eT
,ei[3],eT_count,eT_avg,dspeed,dangle; double
old_angle,old_eangle,obs_angle,observer(),obs
_error;
  double temp;
  int new_cycle,new_ecycle,etrigger,new_obs;
  int C[3],j,phase,phases;
  int eC[3],ecount[3];
  FILE *fopen(),*fp;
  printf ("What is the initial turn on angle (deg)?
");
  scanf ("%lf",&temp);
  controls[0] = temp;
  printf ("What is the initial turn off angle (deg)?
");
  scanf ("%lf",&temp);
  controls[1] = temp - controls[0];
  printf ("What is the initial rotor speed (rpm)?
");
  scanf ("%lf",&temp);
  speeds[0] = pi * temp/30.0;
printf("The initial electrical cycle time is %lf
milliseconds\n",60.0/(temp*4));
  printf ("What is the final rotor speed (rpm)? ");
  scanf ("%lf",&temp);
  speeds[1] = pi * temp/30.0;
printf("The final electrical cycle time is %lf
milliseconds\n",60.0/(temp*4));
  printf ("What is the total time of interest (s)?
");
  scanf ("%lf",&temp);
  times[1] = temp;
  printf ("How many data points are desired (#)?
");
  scanf ("%lf",&temp);
  times[2] = times[1] / temp;
  printf ("Where do you want the simulation
results stored? ");
  scanf ("%s",name);
  printf ("What is the initial speed error (rpm)?
");
  scanf ("%lf",&temp);
  dspeed = pi * temp/30.0;
  printf ("What is the initial position error (deg)?
");
  scanf ("%lf",&temp);
  dangle = temp;
  T_avg = 0;
  T_motor = 0;
  T_count = 0;
  T = 0;
  speed = speeds[0];
  times[3] = 0;
  times[4] = 0;
  times[5] = 0;
  intgrl = 0.0;
  command = 0.0;
  angle = 0;
  new_cycle = 0;
  new_ecycle = 0;
  new_obs = 0;
  obs_error = 0;
  etrigger = 4;
  eT_avg = 0;
  eT_motor = 0;
  eT_count = 0;
eT = 0; for(phase=0;phase<3;phase++)
  {
i[phase] = 0.0; lambda[phase] = 0.0; C[phase] =
0; ei[phase] = 0.0; elambda[phase] = 0.0;
eC[phase] = 0;
ecount[phase] = etrigger; theta[phase] = 0;
etheta[phase] = 0;
  }
  phase = 0;
  phases = 3;
theta_on = controls[0]; theta_cond = controls[1];
  T_cycle = (delta_theta*pi/180)/speed;
  eangle = angle + dangle;
  etheta[0] = theta[0] + dangle;
  espeed = speed + dspeed;
  do
  {
```

```
    if (times[5] >= times[3])
    {
store_data2
(controls,speed,times,command,T_avg,name,esp
eed,etheta[0],theta[0]); times[3] = times[3] +
times[2];
printf("\nThe time is: %lf\n", times[5]);
printf("angle is: %lf\n", angle); printf("estimated
angle is: %lf\n", eangle); printf("motor electrical
torque: %lf\n",T_avg); printf("new speed:
%lf\n",(30.0*speed/pi));
printf("estimated speed:
%lf\n",(30.0*espeed/pi));
    }
    if(new_cycle == 1)
    {
      if (times[4] >= times[0])
      {
printf ("updating excitation at %lf
s.\n",times[5]); controller2
(controls,gains,speed,speeds,&intgrl,times,&co
mmand); printf ("integral: %lf\n",intgrl);
            times[4] = 0.0;
      }
    }
/* Motor Simulation */
    T_motor = 0;
    for(phase=0;phase<3;phase++)
    {
theta[phase] = fmod(angle + phase*60.0,90.0);
if(((fmod(controls[0]+controls[1],90.0)<controls
[0]) &&
(theta[phase]<(controls[0] + controls[1]))) ||
        ((theta[phase]>controls[0]) &&
(theta[phase]<fmod(controls[0]+controls[1],90.0
))))
    {
ecount[phase] = ecount[phase] + 1;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i[phase],A);   if(ecount[phase] ==
etrigger)
          {
obs_angle =
observer(etrigger*T_cycle,eangle,phase,i[phase]
,A); new_obs = 1;
          }
    }
    else if(lambda[phase] > 0.0)
    {
C[phase]=0;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i[phase],A);
    }
    else
    {
          T = 0;
C[phase]=0; lambda[phase]=0; i[phase]=0;
          ecount[phase] = 0;
    }
    T_motor = T_motor + T;
  }
T_count = T_count + T_motor;
  if(new_cycle == 1)
  {
T_avg = T_count/(90.0/T_cycle*speed*180/pi);
T_count = 0;
  }
  angle = angle + T_cycle*speed*180/pi;
  speed = speed + T_cycle * (T_motor - B*speed -
coulombic)/J;
if(fmod(angle,90.0) < fmod(old_angle,90.0))
    new_cycle = 1;
  else
    new_cycle = 0;
  old_angle = angle;
/* Estimator/Observer */
  eT_motor = 0;
  for(phase=0;phase<3;phase++)
  {
etheta[phase] = fmod(eangle + phase*60.0,90.0);
if(((fmod(controls[0]+controls[1],90.0)<controls
[0]) &&
(etheta[phase]<(controls[0] + controls[1]))) ||
((etheta[phase]>controls[0]) &&
(etheta[phase]<fmod(controls[0]+controls[1],90.
0))))
simulate(etheta[phase],espeed,&eC[phase],&ela
mbda[phase],&eT,&ei[phase],A);
    else if(elambda[phase] > 0.0)
    {
eC[phase]=0;
simulate(etheta[phase],espeed,&eC[phase],&ela
mbda[phase],&eT,&ei[phase],A);
    }
    else
    {
          eT = 0;
eC[phase]=0; elambda[phase]=0; ei[phase]=0;
      }
      eT_motor = eT_motor + eT;
    }
eT_count = eT_count + eT_motor;
    if(new_ecycle == 1)
    {
eT_avg =
eT_count/(90.0/T_cycle*speed*180/pi);
eT_count = 0;
    }
    obs_error = obs_angle - eangle;
    eangle = eangle + T_cycle * espeed*180/pi +
new_obs*gains[4]*obs_error;
espeed = espeed + T_cycle * (eT_motor -
B*espeed - coulombic)/J +
```

```
new_obs*gains[5]*obs_error*pi/180; new_obs
= 0;
     if(fmod(eangle,90.0) <
fmod(old_eangle,90.0))
        new_ecycle = 1;
     else
        new_ecycle = 0;
     old_eangle = eangle;
/* Increment Time */
times[5] = times[5] + T_cycle; times[4] =
times[4] + T_cycle;
   }
while(times[5] <= times[1]);
   return;
}
/***********************************/
motor_dynamics9
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
   double
theta[3],speed,lambda[3],T,i[3],T_motor,angle,T
_cycle,T_avg,T_count;
   double theta_on,theta_cond;
   double command,intgrl;
double
eT_motor,etheta[3],eangle,espeed,elambda[3],eT
,ei[3],eT_count,eT_avg,dspeed,dangle; double
old_angle,old_eangle,obs_angle,observer(),obs
_error;
   double temp;
   int new_cycle,new_ecycle,etrigger,new_obs;
   int C[3],j,phase,phases;
   int eC[3],ecount[3];
   FILE *fopen(),*fp;
   printf ("What is the initial turn on angle (deg)?
");
   scanf ("%lf",&temp);
   controls[0] = temp;
   printf ("What is the initial turn off angle (deg)?
");
   scanf ("%lf",&temp);
   controls[1] = temp - controls[0];
   printf ("What is the initial rotor speed (rpm)?
");
   scanf ("%lf",&temp);
   speeds[0] = pi * temp/30.0;
printf("The initial electrical cycle time is %lf
milliseconds\n",60.0/(temp*4));
   printf ("What is the final rotor speed (rpm)? ");
   scanf ("%lf",&temp);
   speeds[1] = pi * temp/30.0;
printf("The final electrical cycle time is %lf
milliseconds\n",60.0/(temp*4));
   printf ("What is the total time of interest (s)?
");
scanf ("%lf",&temp);
times[1] = temp;
printf ("How many data points are desired (#)? ");
scanf ("%lf",&temp);
times[2] = times[1] / temp;
printf ("Where do you want the simulation results
stored? "); scanf ("%s",name);
printf ("What is the initial speed error (rpm)? ");
scanf ("%lf",&temp);
dspeed = pi * temp/30.0;
printf ("What is the initial position error (deg)?
"); scanf ("%lf",&temp);
dangle = temp;
T_avg = 0;
T_motor = 0;
T_count = 0;
T = 0;
speed = speeds[0];
times[3] = 0;
times[4] = 0;
times[5] = 0;
intgrl = 0.0;
command = 0.0;
angle = 0;
new_cycle = 0;
new_ecycle = 0;
new_obs = 0;
obs_error = 0;
etrigger = 4;
eT_avg = 0;
eT_motor = 0;
eT_count = 0;
eT = 0;
for(phase=0;phase<3;phase++)
{
   i[phase] = 0.0;
   lambda[phase] = 0.0;
   C[phase] = 0;
   ei[phase] = 0.0;
   elambda[phase] = 0.0;
   eC[phase] = 0;
ecount[phase] = etrigger; theta[phase] = 0;
etheta[phase] = 0;
}
phase = 0;
phases = 3;
theta_on = controls[0]; theta_cond = controls[1];
T_cycle = (delta_theta*pi/180)/speed;
eangle = angle + dangle;
etheta[0] = theta[0] + dangle;
espeed = speed + dspeed;
do
{
   if (times[5] >= times[3])
   {
```

```
store_data2
(controls,speed,times,command,T_avg,name,esp
eed,etheta[0],theta[0]); times[3] = times[3] +
times[2];
     printf("\nThe time is: %lf\n", times[5]);
printf("angle is: %lf\n", angle); printf("estimated
angle is: %lf\n", eangle); printf("electrical avg
torque: %lf\n",T_avg); printf("new speed:
%lf\n",(30.0*speed/pi));
printf("estimated speed:
%lf\n",(30.0*espeed/pi));
   }
   if(new_cycle == 1)
   {
     if (times[4] >= times[0])
     {
          printf ("updating excitation at %lf
s.\n",times[5]);
controller2
(controls,gains,espeed,speeds,&intgrl,times,&c
ommand); printf ("integral: %lf\n",intgrl);
          times[4] = 0.0;
     }
   }
/* Motor Simulation */
   T_motor = 0;
   for(phase=0;phase<3;phase++)
   {
     theta[phase] = fmod(angle +
phase*60.0,90.0);
etheta[phase] = fmod(eangle + phase*60.0,90.0);
if(((fmod(controls[0]+controls[1],90.0)<controls
[0]) &&
(etheta[phase]<(controls[0] + controls[1])))  ||
((etheta[phase]>controls[0]) &&
(etheta[phase]<fmod(controls[0]+controls[1],90.
0))))
     {
ecount[phase] = ecount[phase] + 1;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i[phase],A);   if(ecount[phase] ==
etrigger)
          {
obs_angle =
observer(etrigger*T_cycle,eangle,phase,i[phase]
,A); new_obs = 1;
          }
     }
     else if(lambda[phase] > 0.0)
     {
C[phase]=0;
simulate(theta[phase],speed,&C[phase],&lambda
[phase],&T,&i[phase],A);
     }
     else
     {
          T = 0;
C[phase]=0; lambda[phase]=0; i[phase]=0;
          ecount[phase] = 0;
     }
     T_motor = T_motor + T;
   }
T_count = T_count + T_motor;
   if(new_cycle == 1)
   {
T_avg = T_count/(90.0/T_cycle*speed*180/pi);
T_count = 0;
   }
   angle = angle + T_cycle*speed*180/pi;
   speed = speed + T_cycle * (T_motor - B*speed -
coulombic)/J;
if(fmod(angle,90.0) < fmod(old_angle,90.0))
   new_cycle = 1; else
   new_cycle = 0; old_angle = angle;
/* Estimator/Observer */
eT_motor = 0;
for(phase=0;phase<3;phase++) {
etheta[phase] = fmod(eangle + phase*60.0,90.0);
if(((fmod(controls[0]+controls[1],90.0)<controls
[0]) &&
(etheta[phase]<(controls[0] + controls[1])))  ||
((etheta[phase]>controls[0]) &&
(etheta[phase]<fmod(controls[0]+controls[1],90.
0))))
simulate(etheta[phase],espeed,&eC[phase],&ela
mbda[phase],&eT,&ei[phase],A);
        else if(elambda[phase] > 0.0)
        {
eC[phase]=0;
simulate(etheta[phase],espeed,&eC[phase],&ela
mbda[phase],&eT,&ei[phase],A);
        }
        else
        {
               eT = 0;
eC[phase]=0; elambda[phase]=0; ei[phase]=0;
        }
        eT_motor = eT_motor + eT;
     }
eT_count = eT_count + eT_motor;
     if(new_ecycle == 1)
     {
eT_avg =
eT_count/(90.0/T_cycle*speed*180/pi);
eT_count = 0;
     }
     obs_error = obs_angle - eangle;
     eangle = eangle + T_cycle * espeed*180/pi +
new_obs*gains[4]*obs_error;
espeed = espeed + T_cycle * (eT_motor -
B*espeed - coulombic)/J +
new_obs*gains[5]*obs_error*pi/180;  new_obs
= 0;
```

```
    if(fmod(eangle,90.0) <
fmod(old_eangle,90.0))
       new_ecycle = 1;
    else
       new_ecycle = 0;
    old_eangle = eangle;
/* Increment Time */
times[5] = times[5] + T_cycle; times[4] =
times[4] + T_cycle;
  }
while(times[5] <= times[1]);
  return;
}
/************************************/
motor_dynamics10
(controls,gains,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
double
theta[3],speed,lambda[3],T,i,T_motor,angle,T_c
ycle,T_avg,T_count; double
theta_on,theta_cond;
int C[3],j,phase,phases;
FILE *fopen(),*fp;
double temp;
double imax;
double command,intgrl;
int cflag;
double *a;
double *dvector(),ilambda();
double dtime;
int ilimit,idelta,tdelta;
double *deriv;
printf ("What is the nominal turn on angle (deg)?
");
scanf ("%lf",&temp);
controls[0] = temp;
printf ("What is the nominal turn off angle (deg)?
");
scanf ("%lf",&temp);
controls[1] = temp - controls[0];
printf ("What is the nominal rotor speed (rpm)?
");
scanf ("%lf",&temp);
speeds[0] = pi * temp/30.0;
printf ("Where do you want the simulation results
stored? ");
scanf ("%s",name);
T_avg = 0;
T_motor = 0;
T_count = 0;
T = 0;
T_cycle = 0;
speed = speeds[0];
times[3] = 0;
times[4] = 0;
times[5] = 0;
intgrl = 0.0;
command = 0.0;
cflag = 0;
angle = 0;
for(phase=0;phase<3;phase++)
{
  lambda[phase] = 0.0;
  C[phase] = 0.0;
}
phase = 0;
phases = 3;
theta_on = controls[0];
theta_cond = controls[1];
T_cycle = (delta_theta*pi/180)/speed;
fp = fopen(name,"w");
fprintf(fp,"theta_on\ttheta_cond\tspeed\tT_avg\
n");
for(theta[phase]=0;theta[phase]<180;theta[phase
]=theta[phase]+delta_theta)
{
if((theta[phase]>theta_on) &&
(theta[phase]<(theta_on + theta_cond))) {
  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A); }
  else if(lambda[phase] > 0.0)
  {
    C[phase] = 0;
  simulate(theta[phase],speed,&C[phase],&lamb
da[phase],&T,&i,A); }
T_avg = T_avg + T;
fprintf(fp,"%lf\t%lf\t%lf\t%lf\n",theta[phase],i,l
ambda[phase],T);
printf("%lf\t%lf\t%lf\t%lf\n",theta[phase],i,lamb
da[phase],T);
  }
  T_avg = phases*T_avg*delta_theta/90.0;
  fclose(fp);
  return;
}
/************************************/
inputs (gains,controls,speeds,times,name,A)
char name[25];
double
controls[2],gains[6],speeds[2],times[6],**A;
{
double temp;
FILE *fopen(),*fp;
int i;
/* Kp, Ki, Kon, Kcond */
fp = fopen ("gains.dat","r");
for (i = 0; i < 6; i++)
  {
  fscanf (fp,"%lf",&temp);
  gains[i] = temp;
  }
```

```
fclose (fp);
fp = fopen ("param.sim","r");
fscanf (fp,"%lf",&delta_theta);
fscanf (fp,"%lf",&R_p);
fscanf (fp,"%lf",&R_s);
fscanf (fp,"%lf",&turns);
turns = turns/160.0;
R_p = R_p*turns*turns;
R_s = R_s*turns*turns;
fscanf (fp,"%lf",&tol);
fscanf (fp,"%lf",&V_b);
fscanf (fp,"%lf",&J);
fscanf (fp,"%lf",&B);
fscanf (fp,"%lf",&T_s);
times[0] = T_s;
fscanf (fp,"%lf",&om_h_s);
fscanf (fp,"%lf",&maxslew);
fscanf (fp,"%lf",&coulombic);
fscanf (fp,"%lf",&maxcond);
fclose (fp);
splind(A);
return;
}
/************************************/
store_data
(controls,speed,times,command,torque,name)
char name[25];
double
command,controls[2],speed,times[6],torque; {
FILE *fopen(),*fp;
speed = speed * 30.0 / pi;
fp = fopen (name,"a");
if (times[5] == 0.0)
  fprintf (fp,"TIME\tON\tCOND\tSPEED\tC.
COM\tDR. T\n");
fprintf (fp,"%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
times[5],controls[0],controls[1],speed,command
,torque);
fclose (fp);
return;
}
/************************************/
store_data2
(controls,speed,times,command,torque,name,esp
eed,etheta,theta) char name[25];
double
command,controls[2],speed,times[6],torque,theta
,etheta,espeed;
{
FILE *fopen(),*fp;
speed = speed * 30.0 / pi;
espeed = espeed * 30.0 / pi;
fp = fopen (name,"a");
if (times[5] == 0.0)
  fprintf
(fp,"TIME\tON\tCOND\tTHETA\tETHETA\tSPEE
D\tESPEED\tC. COM\tDR. T\n");
fprintf
(fp,"%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\
n",
times[5],controls[0],controls[1],theta,etheta,spe
ed,espeed,command,torque);
fclose (fp);
return;
}
/************************************/
double observer(dtime,eangle,phase,ei,A)
double eangle,ei,**A,dtime;
int phase;
{
  double obs_angle;
  double angle,hangle,langle,delta_angle,i;
  double *a;
  double *dvector(),ilambda();
  obs_angle = eangle;
  a=dvector(1,3);
  hangle = 45;
  langle = 0;
  delta_angle = 5;
  angle = -5;
  do
  {
    do
    {
angle = angle + delta_angle; coefficient
(a,A,angle);      i = ilambda (dtime*V_b,a);
    }
while(i<ei);
hangle = angle;
langle = angle - delta_angle; delta_angle =
delta_angle/5; angle = langle;
  }
while((fabs(i-ei)/ei)>0.01);
angle = (hangle + langle)/2;
obs_angle = eangle + (angle - fmod(eangle +
phase*60.0,90.0));
  free_dvector(a,1,3);
  return(obs_angle);
}
/************************************/
controller1
(controls,gains,speed,speeds,pintgrl,times,pcom
mand)
double
controls[2],gains[6],*pcommand,*pintgrl,speed,
speeds[2],times[6];
{
double
i_chop_limit,old_intgrl,old_on,old_cond,old_c
hop,omega_hat,turn_on_limit;
omega_hat = speed - speeds[1];
old_intgrl = *pintgrl;
old_on = controls[0];
old_cond = controls[1];
```

```
*pintgrl = old_intgrl + omega_hat * times[4];
*pcommand = gains[0] * omega_hat + gains[1] *
old_intgrl; if (fabs (*pcommand) > om_h_s)
  {
printf ("CONTROLLER SATURATION
LIMITED\n");
  if (*pcommand < 0.0)
    {
    *pintgrl = old_intgrl;
*pcommand = - om_h_s;
    }
  else
    {
    *pintgrl = old_intgrl;
    *pcommand = om_h_s;
    }
  }
controls[0] = controls[0] + gains[2] *
(*pcommand);
if (controls[0] < 0.0)
  controls[0] = controls[0] + 90.0;
if (controls[0] > 90.0)
  controls[0] = controls[0] - 90.0;
if (controls[1] < 45.0 || (*pcommand) > 0.0)
  controls[1] = controls[1] + gains[3] *
(*pcommand); on_limit (speed,&turn_on_limit);
if (controls[0] < turn_on_limit)
  {
  printf ("TURN ON LIMITED\n");
  controls[0] = turn_on_limit;
/*  controls[1] = old_cond + (gains[3]/gains[2])
* (turn_on_limit - old_on); */
  if (controls[1] > 45.0)
    controls[1] = 45.0;
  }
else if (controls[1] > 45.0)
  {
  printf ("CONDUCTION LIMITED\n");
  controls[1] = 45.0;
controls[0] = old_on + (gains[2]/gains[3]) *
(45.0 - old_cond); if (controls[0] <
turn_on_limit)
    controls[0] = turn_on_limit;
  }
else if (controls[1] < delta_theta)
  {
  controls[1] = delta_theta;
  controls[0] = old_on + (gains[2]/gains[3]) *
(delta_theta - old_cond);
  }
printf ("CONTROLLER ACTION SUMMARY:\n");
printf ("              turn on    : from %lf to
%lf\n",old_on,controls[0]);
printf ("              conduction : from %lf to
%lf\n",old_cond,controls[1]);
return;
}

/************************************/
controller2
(controls,gains,speed,speeds,pintgrl,times,pcom
mand)
double
controls[2],gains[6],*pcommand,*pintgrl,speed,
speeds[2],times[6];
{
double
i_chop_limit,old_pcom,old_intgrl,old_on,old_c
ond,old_chop,omega_hat,turn_on_limit;
if(times[5] == times[4])
{
  nom_on = controls[0];
  nom_cond = controls[1];
}
omega_hat = speed - speeds[1];
old_pcom = *pcommand;
old_intgrl = *pintgrl;
old_on = controls[0];
old_cond = controls[1];
*pintgrl = old_intgrl + omega_hat * times[4];
*pcommand = gains[0] * omega_hat + gains[1] *
old_intgrl; if (fabs (*pcommand) > om_h_s)
{
printf ("CONTROLLER SATURATION
LIMITED\n");
  *pintgrl = old_intgrl;
  if (*pcommand < 0.0)
    *pcommand = - om_h_s;
  else
    *pcommand = om_h_s;
}
on_limit (speed,&turn_on_limit);
/*
if(controls[0] > turn_on_limit)
  controls[0] = nom_on + gains[2] * (-om_h_s);
*/
  controls[0] = nom_on + gains[2] *
*pcommand; if((controls[0] - old_on) >
gains[2]*maxslew)
controls[0] = old_on + gains[2]*maxslew;
if((controls[0] - old_on) < - gains[2]*maxslew)
controls[0] = old_on - gains[2]*maxslew;
if (controls[0] > 90.0)
  controls[0] = 90.0;
else if (controls[0] < turn_on_limit)
  controls[0] = turn_on_limit;
controls[1] = nom_cond + gains[3] *
(*pcommand); if((controls[1] - old_cond) > -
gains[3]*maxslew)
controls[1] = old_cond - gains[3] * maxslew;
if((controls[1] - old_cond) < gains[3]*maxslew)
controls[1] = old_cond + gains[3] * maxslew;
if (controls[1] > maxcond)
controls[1] = maxcond;
```

```
else if (controls[1] < delta_theta) controls[1] =
delta_theta;
printf ("CONTROLLER ACTION SUMMARY:\n");
printf ("              turn on    : from %lf to
%lf\n",old_on,controls[0]);
printf ("              conduction : from %lf to
%lf\n",old_cond,controls[1]);
return;
}
/************************************/
on_limit (speed,plimit)
double *plimit,speed;
{
double m,speed1,turn_on1;
if (speed <= 209.440)
  {
  m = -13.0 / 209.440;
  turn_on1 = 45.0;
  speed1 = 0.0;
  }
else if (speed >  209.440 && speed <= 418.879)
  {
  m = -4.0 / 209.440;
  turn_on1 = 32.0;
  speed1 =   209.440;
  }
else if (speed >  418.879 && speed <= 628.319)
  {
  m = -3.0 / 209.440;
  turn_on1 = 28.0;
  speed1 =   418.879;
  }
else if (speed >  628.319 && speed <= 837.758)
  {
  m = -3.0 / 209.440;
  turn_on1 = 25.0;
  speed1 =   628.319;
  }
else if (speed >  837.758 && speed <=
1047.198)
  {
  m = -2.0 / 209.440;
  turn_on1 = 22.0;
  speed1 =   837.758;
  }
else if (speed > 1047.198 && speed <=
1256.638) {
  m = -2.0 / 209.440;
  turn_on1 = 20.0;
  speed1 = 1047.198;
  }
else
  {
  m = 0.0;
  turn_on1 = 18.0;
  speed1 = 1256.638;
  }
*plimit = turn_on1 + m * (speed - speed1);
return;
}
/************************************/
simulate (double theta,double omega,int
*C,double *lambda,double *T,double *i,double
**A) {
  double lam_prev;
  double *a,*deriv;
  double *dvector();
  a=dvector(1,3);
  deriv=dvector(1,3);
  coefficient (a,A,theta);
  coeff_deriv (deriv,A,theta);
  lam_prev = *lambda;
  motor_current
(theta,omega,C,lam_prev,i,lambda,a);
  torque (*i,theta,T,a,deriv);
  free_dvector(a,1,3);
  free_dvector(deriv,1,3);
  return;
}
/************************************/
motor_current (double theta,double omega,int
*C,double lam_prev,double *i,double
*lambda,double a[]) {
  double delta_lambda,i_old,lambda_pr,R,temp;
  double V;
  int dir,flag,k;
  delta_lambda = V_b * pi * delta_theta / (180.0
* omega);
  i_old = 0.0;
/* Forward Conduction */
  if ((*lambda == 0) || (*C == 1))
  {
    *C = 1;
    V = V_b;
    R = R_p;
    temp = lam_prev + delta_lambda / 2.0;
    *i = ilambda (temp,a);
    do
    {
      i_old = *i;
*lambda = lam_prev + pi * delta_theta * (V - R *
(*i)) / (omega * 180.0);
      lambda_pr = *lambda;
      *i = ilambda (lambda_pr,a);
    }
while (fabs ((*i - i_old)) > tol);
    if(*i >= 20.654)
      *C = 0;
  }

/* Flyback Conduction */
  else
  {
    V = - V_b;
```

```
    R = R_s;
temp = lam_prev - delta_theta / 2.0; if (temp <
0.0)
      temp =  0.0;
    *i = ilambda (temp,a);
    do
    {
i_old = *i;
*lambda = lam_prev + pi * delta_theta * (V - R *
*i) / (omega * 180.0);
      if (*lambda < 0.0)
  *lambda = 0.0; lambda_pr = *lambda;
*i = ilambda (lambda_pr,a);
    }
while (fabs ((*i - i_old)) > tol); if (*i <= 0.0)
    {
      *i = 0.0;
      *lambda = 0.0;
    }
    if(*i < 20.0)
      *C = 1;
  }
  if(*i>40)
  return;
}
/************************************/
splind (A)
double **A;
{
  int i,j;
  FILE *fp,*fopen();
  fp = fopen("motordata","r");
  for(i=1;i<=31;i++)
  {
    for(j=1;j<=4;j++)
fscanf(fp,"%lf",&A[j][i]);
    A[2][i] = turns*turns*A[2][i];
A[3][i] = turns*A[3][i]; A[4][i] =
turns*turns*A[4][i];
  }
  fclose(fp);
  return;
}
/************************************/
coefficient (a,A,theta)
double a[],**A,theta;
{
  int j,k;
  double *y2;
void spline(double x[],double y[],int n,double
yp1,double ypn,double y2[]); void splint(double
xa[],double ya[],double y2a[],int n,double
x,double *y);

  y2=dvector(1,31);
  theta = (double)fmod(theta,90.0);
  for (j = 1; j <= 3; j++)
  {
spline(A[1],A[j+1],31,0.,0.,y2);
splint(A[1],A[j+1],y2,31,theta,&a[j]);
  }
  free_dvector(y2,1,31);
  return;
}
/************************************/
void spline(double x[],double y[],int n,double
yp1,double ypn,double y2[])
{
  int i,k;
  double p,qn,sig,un,*u;
  u=dvector(1,n-1);
  if (yp1 > 0.99e30)
    y2[1]=u[1]=0.0;
  else
  {
y2[1]=-0.5; u[1]=(3.0/(x[2]-x[1]))*((y[2]-
y[1])/(x[2]-x[1])-yp1);
  }
  for (i=2;i<=n-1;i++)
  {
    sig=(x[i]-x[i-1])/(x[i+1]-x[i-1]);
    p=sig*y2[i-1]+2.0;
    y2[i]=(sig-1.0)/p;
u[i]=(y[i+1]-y[i])/(x[i+1]-x[i]) - (y[i]-y[i-
1])/(x[i]-x[i-1]); u[i]=(6.0*u[i]/(x[i+1]-x[i-1])-
sig*u[i-1])/p;
  }
  if (ypn > 0.99e30)
    qn=un=0.0;
  else
  {
    qn=0.5;
un=(3.0/(x[n]-x[n-1]))*(ypn-(y[n]-y[n-1])/(x[n]-
x[n-1]));
  }
  y2[n]=(un-qn*u[n-1])/(qn*y2[n-1]+1.0);
  for (k=n-1;k>=1;k--)
    y2[k]=y2[k]*y2[k+1]+u[k];
  free_dvector(u,1,n-1);
  return;
}
/************************************/
void splint(double xa[],double ya[],double
y2a[],int n,double x,double *y)
{
  int klo,khi,k;
  double h,b,a;
  klo=1;
  khi=n;
  while (khi-klo > 1)
  {
    k=(khi+klo) >> 1;
    if (xa[k] > x)
      khi=k;
```

```
    else
      klo=k;
  }
  h=xa[khi]-xa[klo];
  if (h == 0.0)
    nrerror("Bad XA input to routine SPLINT");
  a=(xa[khi]-x)/h;
b=(x-xa[klo])/h;
*y=a*ya[klo]+b*ya[khi]+((a*a*a-
a)*y2a[klo]+(b*b*b-b)*y2a[khi])*(h*h)/6.0;
  return;
}
/*************************************/
void splindy (double xa[],double ya[],double
y2a[],int n,double x,double *dy)
{
  int klo,khi,k;
  double h,b,a;
  klo=1;
  khi=n;
  while (khi-klo > 1)
  {
    k=(khi+klo) >> 1;
    if (xa[k] > x)
      khi=k;
    else
      klo=k;
  }
  h=xa[khi]-xa[klo];
  if (h == 0.0)
    nrerror("Bad XA input to routine SPLINT");
  a=(xa[khi]-x)/h;
  b=(x-xa[klo])/h;
  *dy=(ya[khi]-ya[klo])/(xa[khi]-xa[klo]) -
(3*a*a-1)*(xa[khi]-xa[klo])*y2a[klo]/6.0 +
(3*b*b-1)*(xa[khi]-xa[klo])*y2a[khi]/6.0;
  return;
}
/*************************************/
double func (y,x,a)
double a[],x,y;
{
double result;
result = y - a[1] * (1.0 - exp(a[2] * x)) - a[3] * x;
return (result);
}
/*************************************/
double ilambda (lambda,a)
double a[],lambda;
{
double  root,x,x1,xr,y,y1,yr;
int flag1,flag2;
flag1 = 0;
flag2 = 0;
x1 = 0;
y1 = func(lambda,x1,a);
do
    {
    xr = x1 + 25.0;
    yr = func(lambda,xr,a);
    if ((y1*yr) ==   0.0)
        {
    if (y1 == 0.0)
          root = x1;
    else
          root = xr;
    flag1 = 1;
    }
    else if ((y1*yr) > 0.0)
        {
    x1 = xr;
y1 = yr;
        }
    else
        {
        do
          {
x = (xr + x1)/2.0; y = func(lambda,x,a); if ((y1*y)
== 0.0)
{
            root = x;
            flag1 = 1;
            flag2 = 1;
            }
            else
              {
if ((y1*y) > 0.0) {
x1 = x; y1 = y; }
            else
              {
xr = x; yr = y; }
            if ((xr - x1) < tol)
              {
root = (xr + x1) / 2.0; flag1 = 1;
flag2 = 1;
              }
            }
          }
        while (flag2 == 0);
        }
  }
while (flag1==0.0);
return (root);
}
/*************************************/
torque (double i,double theta,double *pT,double
a[],double deriv[])
{
    double term1,term2,term3;
    int j,k;
    term1 = i + (1.0 - exp (a[2] * i)) / a[2];
term2 = ((1.0 - exp (a[2] * i)) / a[2] + i * exp (a[2]
* i)) * a[1] / a[2]; term3 = 0.5 * i * i;
```

```
   *pT = term1 * deriv[1] - term2 * deriv[2] +
term3 * deriv[3];
   return;
}
/************************************/
coeff_deriv (deriv,A,theta)
double deriv[],**A,theta;
{
   int j,k;
   double *y2;
void spline(double x[],double y[],int n,double
yp1,double ypn,double y2[]); void
splindy(double xa[],double ya[],double y2a[],int
n,double x,double *dy);
   y2=dvector(1,31);
   theta = (double)fmod(theta,90.0);
   for (j = 1; j <= 3; j++)
   {
spline(A[1],A[j+1],31,0.,0.,y2);
splindy(A[1],A[j+1],y2,31,theta,&deriv[j]);
      deriv[j] = 180*deriv[j]/pi;
   }
   free_dvector(y2,1,31);
   return;
}
```

NRUTIL.C

```
/************************************
 *
 * nrutil.c
 *
 * Ed Lovelace
 *
 * 2/2/92
 *

************************************/

#include <stdio.h>
#include <stdlib.h>

void nrerror(error_text)
char error_text[];
{
   void exit();
   fprintf(stderr,"Numerical Recipes run-time error
...\n");
   fprintf(stderr,"%s\n",error_text);
   fprintf(stderr,"...now exiting to system...\n");
   exit(1);
}

/************************************/

float *vector(nl,nh)
int nl,nh;
{
   float *v;

   v=(float *)malloc((unsigned) (nh-
nl+1)*sizeof(float));
   if (!v) nrerror("allocation failure in vector()");
   return v-nl;
}

/************************************/

int *ivector(nl,nh)
int nl,nh;
{
   int *v;

   v=(int *)malloc((unsigned) (nh-
nl+1)*sizeof(int));
   if (!v) nrerror("allocation failure in ivector()");
   return v-nl;
}

/************************************/

double *dvector(nl,nh)
int nl,nh;
{
   double *v;

   v=(double *)malloc((unsigned) (nh-
nl+1)*sizeof(double));
   if (!v) nrerror("allocation failure in dvector()");
   return v-nl;
}

/************************************/

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
   int i;
   float **m;

   m=(float **) malloc((unsigned) (nrh-
nrl+1)*sizeof(float*));
   if (!m) nrerror("allocation failure 1 in
matrix()");
   m -= nrl;

   for(i=nrl;i<=nrh;i++) {
      m[i]=(float *) malloc((unsigned) (nch-
ncl+1)*sizeof(float));
      if (!m[i]) nrerror("allocation failure 2 in
matrix()");
      m[i] -= ncl;
```

```
  }

  return m;
}

/*********************************/

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
  int i;
  double **m;

  m=(double **) malloc((unsigned) (nrh-
nrl+1)*sizeof(double*));
  if (!m) nrerror("allocation failure 1 in
dmatrix()");
  m -= nrl;

  for(i=nrl;i<=nrh;i++) {
    m[i]=(double *) malloc((unsigned) (nch-
ncl+1)*sizeof(double));
    if (!m[i]) nrerror("allocation failure 2 in
dmatrix()");
    m[i] -= ncl;

  }

  return m;
}

/*********************************/

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
  int i;
  int **m;

  m=(int **) malloc((unsigned) (nrh-
nrl+1)*sizeof(int*));
  if (!m) nrerror("allocation failure 1 in
imatrix()");
  m -= nrl;

  for(i=nrl;i<=nrh;i++) {
    m[i]=(int *) malloc((unsigned) (nch-
ncl+1)*sizeof(int));
    if (!m[i]) nrerror("allocation failure 2 in
imatrix()");
    m[i] -= ncl;

  }

  return m;
}

/*********************************/

float
**submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,new
cl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
  int i,j;
  float **m;

  m=(float **) malloc((unsigned) (oldrh-
oldrl+1)*sizeof(float*));
  if (!m) nrerror("allocation failure in
submatrix()");
  m -= newrl;

  for(i=oldrl,j=newrl;i<=oldrh;i++,j++)
m[j]=a[i]+oldcl-newcl;

  return m;
}

/*********************************/

void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
  free((char*) (v+nl));
}

/*********************************/

void free_ivector(v,nl,nh)
int *v;
int nl,nh;
{
  free((char*) (v+nl));
}

/*********************************/

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
  free((char*) (v+nl));
}

/*********************************/

void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
  int i;
```

```
  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

/*************************************/

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int  nrl,nrh,ncl,nch;
{
  int i;

  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

/*************************************/

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int  nrl,nrh,ncl,nch;
{
  int i;

  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

/*************************************/

void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int  nrl,nrh,ncl,nch;
{
  free((char*) (b+nrl));
}

/*************************************/

float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int  nrl,nrh,ncl,nch;
{
  int i,j,nrow,ncol;
  float **m;

  nrow=nrh-nrl+1;
  ncol=nch-ncl+1;

  m = (float **) malloc((unsigned)
(nrow)*sizeof(float*));
  if (!m) nrerror("allocation failure in
convert_matrix()");
  m -= nrl;
  for(i=0,j=nrl;i<=nrow-1;i++,j++)
m[j]=a+ncol*i-ncl;
  return m;
}

/*************************************/

void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int  nrl,nrh,ncl,nch;
{
  free((char*) (b+nrl));
}
```

GAINS.DAT

```
0.5
0.5
0.5
-0.5
0.4
18.9
Kp
Ki
Kon
Kod
Lt
Lo
```

MOTOR.H

```
#include <stdio.h>
#include <math.h>
#define pi  3.141592654

double t_lock;     /* lock out time built into
controller */
double delta_theta;  /* angular increment of
simulation (degrees) */
double R_p;
double R_s;
double tol;
double turns;
double V_b;
double J;
double B;
double T_s;
double om_h_s;
double nom_on,nom_cond;
```

MOTORDATA

```
0    0.151906  -0.305662    0.002493
3    0.151619  -0.298114    0.002508
```

```
7   0.148072  -0.284335   0.002522
11  0.139373  -0.279143   0.00256
14  0.129272  -0.277409   0.00266
16  0.12022   -0.276613   0.002741
18  0.109505  -0.275656   0.002824
22  0.086498  -0.271217   0.002989
24  0.075357  -0.267324   0.003071
26  0.064814  -0.262365   0.003156
28  0.054813  -0.254499   0.003241
30  0.045    -0.24  0.00332
32  0.03521   -0.216162   0.003389
38  0.012649  -0.109607   0.003485
42  0.007395  -0.033026   0.003495
45  0.007056  -0.005278   0.0035
48  0.007395  -0.033026   0.003495
52  0.012649  -0.109607   0.003485
58  0.03521   -0.216162   0.003389
60  0.045    -0.24  0.00332
62  0.054813  -0.254499   0.003241
64  0.064814  -0.262365   0.003156
66  0.075357  -0.267324   0.003071
68  0.086498  -0.271217   0.002989
72  0.109505  -0.275656   0.002824
74  0.12022   -0.276613   0.002741
76  0.129272  -0.277409   0.00266
79  0.139373  -0.279143   0.00256
83  0.148072  -0.284335   0.002522
87  0.151619  -0.298114   0.002508
90  0.151906  -0.305662   0.002493
```

PARAM.SIM

```
0.25
0.800
0.800
160.0
0.001
68.0
0.00708
0.000531
0.004
50.0
4.0
0.252
44.5
Nominal Values
delta_theta .25
R_p   0.800
R_s   0.800
turns  160.0
tol  0.001
V_b  160.0,70
J 0.00015
B 0.00136
T_s   0.004
om_h_s  50.0
maxslew   4.0
coulombic  0.7125,0.252
maxcond 44.5
```

Appendix B

EXPERIMENT DRIVE SOFTWARE

This appendix lists all the software modules used for the experimental VRM drive system including PC-resident programs, and real-time DSP-resident programs. The header and data files containing motor model constants, position versus current lookup tables, and control parameters are also included. The programs are all written in the C programming language [16].

PCOBON.C	PC-resident program which controls the interface between the user and the DSP-based VRM controller software.
OBON.C	DSP-resident program which controls the VRM drive and performs the observer functions.
DSP.H	Header file which contains the global DSP board register and I/O address definitions.
PCDSP.H	Header file which defines the dual port memory locations used to communicate between the PC and the DSP.
MOTOR.H	Header file for simulation VRM drive constant names.
MOTOR.DAT	Data file for $\lambda - i - \theta$ function coefficients.
PARAM.DAT	Data file for simulation VRM drive constant values.
GAINS.DAT	Data file for controller PI gains and observer Kalman filter gains.
DATAES03.TXT	Data file containing the $\lambda - \theta$ relationship for the inversion of the observer current measurements.

PCOBON.C

```
/*************************************

  FILE:       pcobon.c
  DATE:       January 15, 1996
  USE:        Transient VRM speed control
varying turn on and
              conduction angles using encoder
feedback with off-
              line observer based on current
feedback.
  DSP FILE:   oboff.c

************************************/

#include "c:\c30tools\tms30.h"
#include "pcdsp.h"
#include "motor.h"

#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <GRAPH.H>

#define BOARDADR      0x290
#define COMMO         0x30000
#define STATES        COMMO + 0
#define CURRENT       COMMO + 1
#define TIMES         COMMO + 2
#define GAINS         COMMO + 3
#define PARAMS        COMMO + 4
#define TRANSIENT     COMMO + 5
#define DSPPROCEEDFLAG COMMO + 6
#define PCPROCEEDFLAG  COMMO + 7
#define STORET        COMMO + 8
#define CONTROLS      COMMO + 9
#define THETA         COMMO + 10
#define OLD_THETA     COMMO + 11
#define FLOAT_DUMMY   COMMO + 12
#define ESTATES       COMMO + 13
#define CONTROL_MODE  COMMO + 14
#define LONG_DUMMY    COMMO + 15
#define OBSDATA       COMMO + 16
#define AVG_TORQUE    COMMO + 17
#define INIT_ERROR    COMMO + 18
#define EVENTINT      COMMO + 19

#define DSP_PROGRAM   "obon.out"
#define SHUTDOWN      "shutdown.txt"

void init_dsp();
void input_commands(char data_name[],long
*points);
void splind ();
void spline(float x[],float y[],int n,float
yp1,float ypn,float y2[]);
void display_labels();
void run_loop();
void shutdown();
void file_output(char data_name[],long *points);
void simulate (float stheta,float omega,int
*C,float *lambda,float *T,float *i);
void coefficient (float a[],float stheta);
void splint(float xa[],float ya[],float y2a[],int
n,float x,float *y);
void coeff_deriv (float deriv[],float stheta);
void splindy (float xa[],float ya[],float y2a[],int
n,float x,float *dy);
void motor_current (float omega,int *C,float
lam_prev,float *i,float *lambda,float a[]);
void torque (float i,float stheta,float *pT,float
a[],float deriv[]);
void display_data(float avg_torque);
float **matrix(int nrl,int nrh,int ncl,int nch);
float motor_torque();
float *vector(int nl,int nh);
float ilambda(float lambda,float a[]);
float func(float y,float x,float a[]);

float **A,**y2,load_power;

void main()
{
  char data_name[25];
  long points;

  A=matrix(1,MR,1,MC);
  y2=matrix(1,MR-1,1,MC);

  init_dsp();
  input_commands(data_name,&points);
  display_labels();
  run_loop();
  shutdown();
  file_output(data_name,&points);

  free_matrix(y2,1,MR-1,1,MC);
  free_matrix(A,1,MR,1,MC);

  return;
}

/************************************

init_dsp

Initializes dsp board and loads dsp program

************************************/

void init_dsp()
```

```
{
  unsigned short  loadstat;
  char name[] = DSP_PROGRAM;

/* Initialize Board */

  SelectBoard(BOARDADR);
  loadstat = coffLoad(name);

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

/* Reset & Hold DSP */

  Put32Bit(DSPPROCEEDFLAG,DUAL,0x0L);
  Reset();

  return;
}

/************************************

input_commands

Requests user input of commands for simulation.

************************************/

void input_commands(char data_name[],long
*points)
{
  float
temp,controls[4],times[6],gains[6],params[14],o
bsdata[2][181],estates[3],avg_torque[1],*a,event
_int[1];
  FILE *fopen(),*fp;
  int i,j;
  char *line;

  a=vector(1,3);

  _clearscreen(_GCLEARSCREEN);
  _displaycursor(_GCURSOROFF);

  /* Flux linkage - current model */

  splind();
  for(j=1;j<MR;j++)
    spline(A[1],A[j+1],MC,0.,0.,y2[j]);

  /* Control Gains Kp, Ki, Kon, Kcond, Lt, Lo */

  fp = fopen ("gains.dat","r");
  for (i = 0; i < 6; i++)
  {
    fscanf (fp,"%f",&temp);
    gains[i] = temp;
  }
  fclose (fp);

  /* Drive System Parameters */

  fp = fopen ("param.dat","r");
  for (i = 0; i < 14; i++)
  {
    fscanf (fp,"%f",&temp);
    params[i] = temp;
  }
  fclose (fp);

  delta_theta = params[0]*PI/180; /* angular
increment of simulation */
  R_p = params[1];          /* primary winding
resistance */
  G2R = params[2];          /* load motor
constant G^2/Ra */
  temp = params[3];         /* ratio of winding
turns to 160 turns */
  turns = temp/160.0;
  tol = params[4];          /* root-finding
tolerance */
  V_b = params[5];          /* battery voltage */
  J = params[6];            /* load inertia */
  B = params[7];            /* load viscous
damping */
  T_s = params[8];          /* controller update
time */
  om_h_s = params[9];       /* controller
saturation limit */
  coulombic = params[10];   /* coulombic
load torque */
  i_chop = params[11];      /* current
chopping limit */
  i_hyst = params[12];      /* current chopping
hysteresis */
  maxslew = params[13];

  /* Input User Commands */

  printf("Input turn on angle:  ");
  scanf("%f",&controls[0]);
  controls[0] = controls[0]*PI/180;
  printf("\nInput conduction angle:  ");
  scanf("%f",&controls[1]);
  controls[1] = controls[1]*PI/180;
  printf("\nEvent Interval [updates]:  ");
  scanf("%f",&event_int[0]);
  printf("\nInput speed limit:  ");
```

```
  scanf("%f",&controls[2]);
  controls[2] = controls[2]*PI/30;
  controls[3] = 0.0;
  printf("\nInput load field amps: ");
  scanf("%f",&temp);
  load_power = G2R*temp*temp;
  printf("\nInput end time: ");
  scanf("%f",&times[2]);
  printf("\nInput number of points to store: ");
  scanf("%ld",points);
  times[0] = times[2]/(float)(*points);
  printf("\nCalculated storage time interval is:
%f",times[0]);
  printf("\n\nInput data storage file name: ");
  scanf("%s",data_name);

  /* Calculate Observer Data */
/*
  fp = fopen ("dataes01.txt","r");
  fgets(line,180,fp);
*/
  fp = fopen ("dataes03.txt","r");
  for (i = 0; i <= 180; i++)
  {
    fscanf (fp,"%f",&temp);
    obsdata[0][i] = temp*PI/180;
    fscanf (fp,"%f\n",&temp);
    obsdata[1][i] = temp;
  }
  fclose (fp);

  tol = 0.00001;
/*
  fp = fopen ("dataes02.txt","w");
  for (i = 0; i <= 180; i++)
  {
    obsdata[0][i] = ((float)(i))*PI/720.0;
    coefficient(a,obsdata[0][i]);
    obsdata[1][i] = ilambda(est_time[0]*V_b,a);

  fprintf(fp,"%f\t%f\n",obsdata[0][i]*180/PI,ob
sdata[1][i]);
  }
  fclose (fp);
*/
  avg_torque[0] = 0.111;
  tol = params[4];

  WrBlkFlt(Get32Bit(CONTROLS,DUAL),DUAL,
4,controls);
  WrBlkFlt(Get32Bit(TIMES,DUAL),DUAL,6,ti
mes);
  WrBlkFlt(Get32Bit(GAINS,DUAL),DUAL,6,ga
ins);
  WrBlkFlt(Get32Bit(PARAMS,DUAL),DUAL,1
4,params);
  WrBlkFlt(Get32Bit(OBSDATA,DUAL),DUAL,2
*181,obsdata[0]);
  WrBlkFlt(Get32Bit(ESTATES,DUAL),DUAL,3,
estates);
  WrBlkFlt(Get32Bit(AVG_TORQUE,DUAL),DU
AL,1,avg_torque);
  WrBlkFlt(Get32Bit(EVENTINT,DUAL),DUAL,
1,event_int);
  Put32Bit(CONTROL_MODE,DUAL,0x0L);
  Put32Bit(PCPROCEEDFLAG,DUAL,0x0L);
  Put32Bit(DSPPROCEEDFLAG,DUAL,0x1L);

  free_vector(a,1,3);

  return;
}

/************************************/

void splind()
{
  int i,j;
  FILE *fp,*fopen();

  fp = fopen("motor.dat","r");
  for(i=1;i<=31;i++)
  {
    for(j=1;j<=4;j++)
      fscanf(fp,"%f",&A[j][i]);
    A[1][i] = A[1][i]*PI/180.0;
  }
  fclose(fp);

  return;
}

/************************************

display_labels

Display screen labels

************************************/

void display_labels()
{
/* Setup Screen Display */

  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,10);
  printf("   Motor Angle:        ");
  _settextposition(2,10);
  printf(" Phase A Current:      ");
  _settextposition(3,10);
  printf(" Phase B Current:      ");
  _settextposition(4,10);
  printf(" Phase C Current:      ");
```

```
  _settextposition(5,10);
  printf("    Motor Speed:        ");
  _settextposition(6,10);
  printf("  Elapsed Time:         ");
  _settextposition(7,10);
  printf("  Turn on Angle:        ");
  _settextposition(8,10);
  printf("Conduction Angle:       ");
  _settextposition(9,10);
  printf(" Estimated Speed:       ");
  _settextposition(10,10);
  printf("Est. Motor Angle:       ");
  _settextposition(11,10);
  printf("  Battery Volts:        ");
  _settextposition(12,10);
  printf("  Dummy Float 1:        ");
  _settextposition(13,10);
  printf("  Dummy Float 2:        ");
  _settextposition(14,10);
  printf("  Dummy Float 3:        ");
  _settextposition(15,10);
  printf("  Dummy Float 4:        ");
  _settextposition(16,10);
  printf("    Long Dummy:         ");
  _settextposition(17,10);
  printf("  Average Torque:       ");

  return;
}

/************************************

run_loop

Running program loop waiting for user
commands

************************************/

void run_loop()
{
  int go,c,tempc;
  long control_mode;
  float
avg_torque,controls[4],init_error[3],temp,event
_int[1];

  avg_torque = 0.0;

/* Wait for DSP to complete VRM startup */
  while(Get32Bit(PCPROCEEDFLAG,DUAL) ==
0x0L)
    ;

/* Enter Running Loop */
  go = 1;
  while(go)
  {
    if(kbhit()!=0)
    {
      tempc = getche();
      if(tempc == '\r')
        switch(c)
        {
          /* Change control angles */
          case 'C':
            _clearscreen(_GCLEARSCREEN);
            printf("Input turn on angle: ");
            scanf("%f",&controls[0]);
            controls[0] = controls[0]*PI/180;
            printf("\nInput conduction angle:
");
            scanf("%f",&controls[1]);
            controls[1] = controls[1]*PI/180;
            printf("\nInput speed limit: ");
            scanf("%f",&controls[2]);
            controls[2] = controls[2]*PI/30;
            printf("\nInput target speed: ");
            scanf("%f",&controls[3]);
            controls[3] = controls[3]*PI/30;
            printf("\nInput load field amps: ");
            scanf("%f",&temp);
            load_power = G2R*temp*temp;
            printf("\nEvent Interval [updates]:
");
            scanf("%f",&event_int[0]);

  WrBlkFlt(Get32Bit(EVENTINT,DUAL),DUAL,
1,event_int);

  WrBlkFlt(Get32Bit(CONTROLS,DUAL),DUAL,
4,controls);
            display_labels();
            break;

          /* Change control mode */
          case 'm':
            _clearscreen(_GCLEARSCREEN);
            printf("\n#\tControl Mode");
            printf("\n0\tFixed Control");
            printf("\n1\tTransient Control");
            printf("\n2\tEstimation");
            printf("\n3\tEstimation &
Commutation Control");
            printf("\n4\tEstimation &
Speed/Comm Control");
            printf("\n\nSelect mode: ");
            scanf("%ld",&control_mode);
            if(control_mode >= 2)
            {
              printf("\n\n\tInput initial
position error (degrees): ");
              scanf("%f",&init_error[1]);
```

```
                init_error[1] =
init_error[1]*PI/180;
                printf("\n\tInput initial speed
error (rpm): ");
                scanf("%f",&init_error[2]);
                init_error[2] =
init_error[2]*PI/30;

  WrBlkFlt(Get32Bit(INIT_ERROR,DUAL),DUA
L,3,init_error);
                printf("\n\tEvent Interval
[updates]: ");
                scanf("%f",&event_int[0]);

  WrBlkFlt(Get32Bit(EVENTINT,DUAL),DUAL,
1,event_int);
              }

  Put32Bit(CONTROL_MODE,DUAL,control_m
ode);
              display_labels();
              break;

            /* Stop motor */
            default:
              go = 0;
              break;
          }
        else
          c = tempc;
    }

    /* Calculate Average Torque */
    avg_torque = motor_torque();

    /* Refresh Screen */
    display_data(avg_torque);
  }

/* Halt DSP and Record Datafile */

  Put32Bit(DSPPROCEEDFLAG,DUAL,0x0L);
  _clearscreen(_GCLEARSCREEN);
  _settextposition(1,1);
  _displaycursor(_GCURSORON);

  return;
}

/************************************/

float motor_torque()
{
  float
controls[4],states[3],T,avg_torque[1],angle,lamb
da,i;
  int C;

  T = 0.0;
  avg_torque[0] = 0.0;
  lambda = 0.0;
  i = 0.0;
  C = 0;

  RdBlkFlt(Get32Bit(CONTROLS,DUAL),DUAL,
4,controls);
  RdBlkFlt(Get32Bit(STATES,DUAL),DUAL,3,st
ates);
  if((states[2] > 1000.0*PI/30.0) && (states[2] <
11000.0*PI/30.0))
  {

  for(angle=0.0;angle<PI;angle=angle+delta_the
ta)
    {
      if((angle>controls[0]) &&
(angle<(controls[0]+controls[1])))

  simulate(angle,states[2],&C,&lambda,&T,&i);
      else if(lambda > 0.0)
      {
        C=0;

  simulate(angle,states[2],&C,&lambda,&T,&i);
      }
      avg_torque[0]=avg_torque[0] + T;
    }
    avg_torque[0] =
3*avg_torque[0]*delta_theta/(PI/2) -
load_power/states[2] - coulombic;
  }
  else
    avg_torque[0] = 0.0;

  WrBlkFlt(Get32Bit(AVG_TORQUE,DUAL),DU
AL,1,avg_torque);
  return(avg_torque[0]);
}

/************************************/

void simulate (float stheta,float omega,int
*C,float *lambda,float *T,float *i)
{
  float lam_prev;
  float *a,*deriv;

  a=vector(1,3);
  deriv=vector(1,3);

  coefficient (a,stheta);
  coeff_deriv (deriv,stheta);

  lam_prev = *lambda;
```

```
  motor_current
(omega,C,lam_prev,i,lambda,a); /*
  motor_current
(2000.0*PI/30.0,C,lam_prev,i,lambda,a); */
  torque (*i,stheta,T,a,deriv);

  free_vector(a,1,3);
  free_vector(deriv,1,3);

  return;
}

/*************************************/

void coefficient (float a[],float stheta)
{
  int j,k;

  stheta = (float)fmod(stheta,PI/2.0);
  for (j = 1; j <= 3; j++)
    splint(A[1],A[j+1],y2[j],31,stheta,&a[j]);

  return;
}

/*************************************/

void splint(float xa[],float ya[],float y2a[],int
n,float x,float *y)
{
  int klo,khi,k;
  float h,b,a;

  klo=1;
  khi=n;
  while (khi-klo > 1)
  {
    k=(khi+klo) >> 1;
    if (xa[k] > x)
      khi=k;
    else
      klo=k;
  }
  h=xa[khi]-xa[klo];
  if (h == 0.0)
    nrerror("Bad XA input to routine SPLINT");
  a=(xa[khi]-x)/h;
  b=(x-xa[klo])/h;
  *y=a*ya[klo]+b*ya[khi]+((a*a*a-
a)*y2a[klo]+(b*b*b-b)*y2a[khi])*(h*h)/6.0;

  return;
}

/*************************************/

void coeff_deriv (float deriv[],float stheta)
{
  int j,k;

  stheta = (float)fmod(stheta,PI/2.0);
  for (j = 1; j <= 3; j++)
  {

  splindy(A[1],A[j+1],y2[j],31,stheta,&deriv[j])
;
    deriv[j] = deriv[j];
  }

  return;
}

/*************************************/

void splindy (float xa[],float ya[],float y2a[],int
n,float x,float *dy)
{
  int klo,khi,k;
  float h,b,a;

  klo=1;
  khi=n;
  while (khi-klo > 1)
  {
    k=(khi+klo) >> 1;
    if (xa[k] > x)
      khi=k;
    else
      klo=k;
  }
  h=xa[khi]-xa[klo];
  if (h == 0.0)
    nrerror("Bad XA input to routine SPLINT");
  a=(xa[khi]-x)/h;
  b=(x-xa[klo])/h;
  *dy=(ya[khi]-ya[klo])/(xa[khi]-xa[klo]) -
(3*a*a-1)*(xa[khi]-xa[klo])*y2a[klo]/6.0 +
(3*b*b-1)*(xa[khi]-xa[klo])*y2a[khi]/6.0;

  return;
}

/*************************************/

void motor_current (float omega,int *C,float
lam_prev,float *i,float *lambda,float a[])
{
  float delta_lambda,i_old,lambda_pr,R,temp;
  float V;
  int dir,flag,k;

  delta_lambda = V_b * delta_theta/omega;
  i_old = 0.0;
```

```
/* Forward Conduction */

  if ((*lambda == 0) || (*C == 1))
  {
    *C = 1;
    V = V_b;
    R = R_p;
    temp = lam_prev + delta_lambda / 2.0;
    *i = ilambda (temp,a);
    do
    {
      i_old = *i;
      *lambda = lam_prev + (V - R * (*i)) *
delta_theta/omega;
      lambda_pr = *lambda;
      *i = ilambda (lambda_pr,a);
    }
    while (fabs ((*i - i_old)) > tol);
    if(*i > i_chop)
      *C = 0;
  }

/* Flyback Conduction */

  else
  {
    V = - V_b;
    R = R_p;
    temp = lam_prev - delta_lambda / 2.0;
    if (temp < 0.0)
      temp = 0.0;
    *i = ilambda (temp,a);
    do
    {
      i_old = *i;
      *lambda = lam_prev + (V - R * *i) *
delta_theta/omega;
      if (*lambda < 0.0)
        *lambda = 0.0;
      lambda_pr = *lambda;
      *i = ilambda (lambda_pr,a);
    }
    while (fabs ((*i - i_old)) > tol);
    if (*i <= 0.0)
    {
      *i = 0.0;
      *lambda = 0.0;
    }
    if(*i <= (i_chop - i_hyst))
      *C = 1;
  }
  return;
}

/*************************************

ilambda()

Calculates the current based on flux estimates
(previous flux + voltage*time)

*************************************/

float ilambda(float lambda,float a[])
{
  float  root,x,x1,xr,y,y1,yr;
  int  flag1,flag2;

  flag1 = 0;
  flag2 = 0;
  x1 = 0;

  /* Flux error for current = 0 */
  y1 = func(lambda,x1,a);
  do
  {
    /* Increment for second current estimate -
nominal delta = 25 Amps */
    xr = x1 + 5.0;
    yr = func(lambda,xr,a);

    /* One current estimate is exact */
    if ((y1*yr) == 0.0)
    {
      if (y1 == 0.0)
        root = x1;
      else
        root = xr;
      flag1 = 1;
    }

    /* Current estimates are too low - start over
and increment current */
    else if ((y1*yr) > 0.0)
    {
      x1 = xr;
      y1 = yr;
    }

    /* Current is between the two estimates -
iterate */
    else
    {
      do
      {
        x = (xr + x1)/2.0;
        y = func(lambda,x,a);
        if ((y1*y) == 0.0)
        {
          root = x;
          flag1 = 1;
          flag2 = 1;
        }
        else
```

```
        {
          if ((y1*y) > 0.0)
          {
            x1 = x;
            y1 = y;
          }
          else
          {
            xr = x;
            yr = y;
          }
          if ((xr - x1) < tol)
          {
            root = (xr + x1) / 2.0;
            flag1 = 1;
            flag2 = 1;
          }
        }
      }
      while (flag2 == 0);
    }
  }
  while (flag1==0);
  return (root);
}

/************************************

func()

Calculates flux from current using an exponential
family of functions, and
returns the difference between the estimated and
calculated flux.

************************************/

float func(float y,float x,float a[])
{
  float result;

  result = y - a[1] * (1.0 - exp(a[2] * x)) - a[3] * x;
  return (result);
}

/************************************/

void torque (float i,float stheta,float *pT,float
a[],float deriv[])
{
  float term1,term2,term3;
  int j,k;

  term1 = i + (1.0 - exp (a[2] * i)) / a[2];
  term2 = ((1.0 - exp (a[2] * i)) / a[2] + i * exp
(a[2] * i)) * a[1] / a[2];
  term3 = 0.5 * i * i;

  *pT = term1 * deriv[1] - term2 * deriv[2] +
term3 * deriv[3];

  return;
}

/************************************/

void spline(float x[],float y[],int n,float
yp1,float ypn,float y2[])
{
  int i,k;
  float p,qn,sig,un,*u;
  u=vector(1,n-1);
  if (yp1 > 0.99e30)
    y2[1]=u[1]=0.0;
  else
  {
    y2[1]=-0.5;
    u[1]=(3.0/(x[2]-x[1]))*((y[2]-y[1])/(x[2]-
x[1])-yp1);
  }
  for (i=2;i<=n-1;i++)
  {
    sig=(x[i]-x[i-1])/(x[i+1]-x[i-1]);
    p=sig*y2[i-1]+2.0;
    y2[i]=(sig-1.0)/p;
    u[i]=(y[i+1]-y[i])/(x[i+1]-x[i]) - (y[i]-y[i-
1])/(x[i]-x[i-1]);
    u[i]=(6.0*u[i]/(x[i+1]-x[i-1])-sig*u[i-1])/p;
  }
  if (ypn > 0.99e30)
    qn=un=0.0;
  else
  {
    qn=0.5;
    un=(3.0/(x[n]-x[n-1]))*(ypn-(y[n]-y[n-
1])/(x[n]-x[n-1]));
  }
  y2[n]=(un-qn*u[n-1])/(qn*y2[n-1]+1.0);
  for (k=n-1;k>=1;k--)
    y2[k]=y2[k]*y2[k+1]+u[k];
  free_vector(u,1,n-1);

  return;
}

/************************************/

float *vector(nl,nh)
int nl,nh;
{
  float *v;

  v=(float *)malloc((unsigned) (nh-
nl+1)*sizeof(float));
  if (!v) nrerror("allocation failure in vector()");
```

```
  return v-nl;
}

/*************************************/

free_vector(v,nl,nh)
float *v;
int nl,nh;
{
  free((char*) (v+nl));
}

/*************************************

display_data

Update screen display of VRM parameters

*************************************/

void display_data(float avg_torque)
{
  float
states[3],current[3],theta[3],old_theta[3],control
s[4],times[6];
  long    long_dummy;
  float   estates[3],float_dummy[BUGS];

  RdBlkFlt(Get32Bit(STATES,DUAL),DUAL,3,st
ates);
  RdBlkFlt(Get32Bit(TIMES,DUAL),DUAL,6,ti
mes);
  RdBlkFlt(Get32Bit(CURRENT,DUAL),DUAL,3
,current);
  RdBlkFlt(Get32Bit(CONTROLS,DUAL),DUAL,
4,controls);
  RdBlkFlt(Get32Bit(THETA,DUAL),DUAL,3,th
eta);
  RdBlkFlt(Get32Bit(OLD_THETA,DUAL),DUAL
,3,old_theta);
  RdBlkFlt(Get32Bit(ESTATES,DUAL),DUAL,3,
estates);
  long_dummy =
Get32Bit(LONG_DUMMY,DUAL);
  RdBlkFlt(Get32Bit(FLOAT_DUMMY,DUAL),D
UAL,BUGS,float_dummy);
  _settextposition(1,28);
  printf("%-4.3f    ",states[1]*180/PI);
  _settextposition(2,28);
  printf("%-4.3f    ",current[0]);
  _settextposition(3,28);
  printf("%-4.3f    ",current[1]);
  _settextposition(4,28);
  printf("%-4.3f    ",current[2]);
  _settextposition(5,28);
  printf("%-4.3f     ",states[2]*30/PI);
  _settextposition(6,28);
  printf("%-4.3f     ",states[0]);
  _settextposition(7,28);
  printf("%-4.3f     ",controls[0]*180/PI);
  _settextposition(8,28);
  printf("%-4.3f     ",controls[1]*180/PI);
  _settextposition(9,28);
  printf("%-4.3f     ",estates[2]*30/PI);
  _settextposition(10,28);
  printf("%-4.3f     ",estates[1]*180/PI);
  _settextposition(11,28);
  printf("%-4.3g     ",float_dummy[0]);
  V_b = float_dummy[0];
  _settextposition(12,28);
  printf("%-4.3g     ",float_dummy[1]);
  _settextposition(13,28);
  printf("%-4.3g     ",float_dummy[2]);
  _settextposition(14,28);
  printf("%-4.3g     ",float_dummy[3]);
  _settextposition(15,28);
  printf("%-4.3g     ",float_dummy[4]);
  _settextposition(16,28);
  printf("%-2.2f     ",(float)(long_dummy));
  _settextposition(17,28);
  printf("%-4.3g     ",avg_torque);
  _settextposition(18,5);
  printf("          C - Change control setpoint");
  _settextposition(19,5);
  printf("          m - Change control mode");
  _settextposition(20,5);
  printf("          S - Stop motor and quit");
  _settextposition(21,5);
  printf("Enter choice:  ");

  return;
}

/*************************************

shutdown

Copies ring buffer data from DSP RAM to a PC
file

*************************************/

void shutdown()
{
  FILE        *fopen(),*fp;
  unsigned long   storetloc;
  int         i,mini;
  float       storet[M][N],mins;

  mins = 9999.9;
  fp = fopen(SHUTDOWN,"w");
  fprintf(fp,"Time  \tAngle  \tSpeed
\tCommand\tFlt_Dummy2\tFlt_Dummy3\tFlt_Du
```

```
mmy4\tEst_Angle\tEst_Speed\tTurn_On\tTurn_O
ff\n");
  storetloc = Get32Bit(STORET,DUAL);
  RdBlkFlt(storetloc,DUAL,M*N,storet[0]);
  for(i=0;i<M;i++)
    if(storet[i][0]<mins)
    {
      mins = storet[i][0];
      mini = i;
    }
  for(i=mini;i<M;i++)
  {
    fprintf(fp,"%f\t",storet[i][0]-
storet[mini][0]);
    fprintf(fp,"%f\t",storet[i][1]);
    fprintf(fp,"%f\t",storet[i][2]);
    fprintf(fp,"%-1.1f\t",storet[i][3]);
    fprintf(fp,"%f\t",storet[i][4]);
    fprintf(fp,"%f\t",storet[i][5]);
    fprintf(fp,"%f\t",storet[i][6]);
    fprintf(fp,"%f\t",storet[i][7]);
    fprintf(fp,"%f\t",storet[i][8]);
    fprintf(fp,"%f\t",storet[i][9]);
    fprintf(fp,"%f\t",storet[i][10]);
    fprintf(fp,"\n");
  }
  for(i=0;i<mini;i++)
  {
    fprintf(fp,"%f\t",storet[i][0]-
storet[mini][0]);
    fprintf(fp,"%f\t",storet[i][1]);
    fprintf(fp,"%f\t",storet[i][2]);
    fprintf(fp,"%-1.1f\t",storet[i][3]);
    fprintf(fp,"%f\t",storet[i][4]);
    fprintf(fp,"%f\t",storet[i][5]);
    fprintf(fp,"%f\t",storet[i][6]);
    fprintf(fp,"%f\t",storet[i][7]);
    fprintf(fp,"%f\t",storet[i][8]);
    fprintf(fp,"%f\t",storet[i][9]);
    fprintf(fp,"%f\t",storet[i][10]);
    fprintf(fp,"\n");
  }
  fclose(fp);

  return;
}

/************************************

file_output

Stores transient data from DSP RAM to a PC file

************************************/

void file_output(char name[],long *points)
{
  FILE        *fopen(),*fp;
  unsigned long   storetloc;
  int         i,j;
  float       transient[CM][CN];

  fp = fopen(name,"w");
  fprintf(fp,"Time    \tDummy_2 \tSpeed
\tDummy_3\tDummy_4\tErr_Angle\tErr_Speed\n
");
  storetloc = Get32Bit(TRANSIENT,DUAL);
  RdBlkFlt(storetloc,DUAL,(short)(*points)*CN
,transient[0]);
  for(i=0;i<(*points);i++)
  {
    fprintf(fp,"%f\t",transient[i][0]);
    fprintf(fp,"%f\t",transient[i][1]);
    fprintf(fp,"%f\t",transient[i][2]);
    fprintf(fp,"%f\t",transient[i][3]);
    fprintf(fp,"%f\t",transient[i][4]);
    fprintf(fp,"%f\t",transient[i][5]);
    fprintf(fp,"%f\t",transient[i][6]);
    fprintf(fp,"\n");
  }
  fclose(fp);

  return;
}

/************************************/

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
  int i;
  float **m;

  m=(float **) malloc((unsigned) (nrh-
nrl+1)*sizeof(float*));
  if (!m) nrerror("allocation failure 1 in
matrix()");
  m -= nrl;

  for(i=nrl;i<=nrh;i++) {
    m[i]=(float *) malloc((unsigned) (nch-
ncl+1)*sizeof(float));
    if (!m[i]) nrerror("allocation failure 2 in
matrix()");
    m[i] -= ncl;

  }

  return m;
}

/************************************/

free_matrix(m,nrl,nrh,ncl,nch)
```

```
float **m;
int nrl,nrh,ncl,nch;
{
  int i;

  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

/*************************************/

nrerror(error_text)
char error_text[];
{
  void exit();
  fprintf(stderr,"Numerical Recipes run-time error
...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
}
```

OBON.C

```
/*************************************

  File:     obon.c
  Date:     January 16, 1995
  Use:      Transient closed Loop VRM Control
using on-line observer
          based on current feedback.  All
calculations in std units.
  PC File:  pcobon.c

*************************************/

asm("   .length 58");
asm("   .width  120");

#include "pcdsp.h"
#include "dsp.h"
#include "motor.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float observer(float eangle,float ei,long phase);
float func(float y,float x,float a[]);
float ilambda(float lambda,float a[]);
float *vector(int nl,int nh);
float **matrix(int nrl,int nrh,int ncl,int nch);
float **convert_matrix(float *a,int nrl,int
nrh,int ncl,int nch);

extern float
*Comm0,*Comm1,*Comm2,*Comm3,*Comm4
,*Comm5;
extern long   Comm6,Comm7;
extern float
*Comm8,*Comm9,*Comm10,*Comm11;
extern float  *Comm12;
extern float  *Comm13;
extern long   Comm14,Comm15;
extern float
*Comm16,*Comm17,*Comm18,*Comm19;

float
states[3],current[3],controls[4],times[6],gains[6]
,params[14];
long    *dspproceedflag,*pcproceedflag;
float
storet[M][N],transient[CM][CN],theta[3],old_the
ta[3];
float  float_dummy[BUGS];
float  estates[3];
long    *control_mode,*long_dummy;
float
obsdata[2][181],avg_torque[1],init_error[3],even
t_int[1];

long
index1,init_counts0,revolutions,flag_storet,new
_obs,intmask;
float   dtime,coffset[3],**odata,*odata2,battery;

main()
{
  float command,intgrl,old_angle,nom_speed;
  long
inv_command,phase,data_point,shutdown_point
;
  float etime[3];
  long i,j,shutdown_update;

  init_pointers();

  odata2 = vector(1,181);
  for(i=0;i<CM;i++)
  {
    for(j=0;j<CN;j++)
      transient[i][j] = 0.0;
  }
  for(i=0;i<BUGS;i++)
    float_dummy[i] = 0.0;
  for(i=0;i<3;i++)
    etime[i] = ESTTIME;
  *long_dummy = 0;
  dtime = 0.0;
  nom_speed = 0.0;
  data_point = 0;
  shutdown_point = 0;
```

```
  shutdown_update = 0;
  revolutions = 0;
  flag_storet = 0;
  new_obs = 0;
  event_int[0] = 0;
  intmask = 1;

/* Turn VRM off and wait for PC control inputs */

  *probrd1 = PHOFF;
  while(*dspproceedflag == 0)
    ;

/* Initialize and start the motor */

  init_motor();
  spin_motor(&inv_command,&old_angle);

/* Motor control loop
  while((*dspproceedflag == 1) && (times[5] <
times[2]) && (revolutions < 4)) */
  while((*dspproceedflag == 1) && (revolutions
< 4))
  {
    /* Store motor data */
    if((times[3] >= times[0]) && (times[5] <
times[2]))
    {

  store_data(&command,&intgrl,&data_point);
      times[3] = 0.0;
    }

    /* Store ring buffer data for failure analysis
*/
    if(shutdown_update <= 0)
    {

  shutdown_data(inv_command,command,intgrl,
&shutdown_point);
      shutdown_update = event_int[0];
    }
    else
      shutdown_update--;

    /* Update control angles */
    if(times[4] >= times[1])
    {
      if(controls[3] > 0.1)
      {
        if(nom_speed != controls[3])
        {
          nom_speed = controls[3];
          nom_on = controls[0];
          nom_cond = controls[1];
          command = 0;
          intgrl = 0;
          shutdown_point = 0;
        }
        controller(&command,&intgrl);
      }
      times[4] = 0.0;
    }

    /* Send inverter commands */

  vrm_commands(&inv_command,&old_angle,e
time);

    /* Update estimator */
    if(*control_mode >= 2)
    {
      if(flag_storet == 0)
      {
        estates[1] = states[1] + init_error[1];
        estates[2] = states[2] + init_error[2];
        flag_storet = 1;
        shutdown_point = 0;
        /* Battery supply voltage */
        *anabrd0 = RDBATT;
        *anabrd1 = ANATIME;
        while(0 == (*anabrd0 & EOC))
          ;
        battery = ((float) (*anabrd2 >>
20))*201*2.5/2048;
        float_dummy[0] = battery;
      }
      estimator(etime);
    }
    else
    {
      estates[1] = states[1];
      estates[2] = states[2];
      for(i=0;i<3;i++)
        etime[i] = ESTTIME;
    }

    /* Increment time */
    states[0] = states[0] + dtime;
    times[3] = times[3] + dtime;
    times[4] = times[4] + dtime;
    times[5] = states[0];
    *pcproceedflag = 1;
  } /* End of control loop */

/* Shut off the motor */
  *pcproceedflag = 1;
  *timctl1 = RSTINTI;
  *probrd1 = PHOFF;
  free_matrix(odata2,1,181);
}

/************************************
```

```
init_pointers

Initialize pointers for communication to DSP
board and PC program.

************************************/

init_pointers()
{
/* Initialize DSP pointers */

   timctl1 = (long *) TIMCTL1;
   period1 = (long *) PERIOD1;
   counts1 = (long *) COUNT1;
   timctl0 = (long *) TIMCTL0;
   period0 = (long *) PERIOD0;
   counts0 = (long *) COUNT0;
   anabrd0 = (long *) ANABRD0;
   anabrd1 = (long *) ANABRD1;
   anabrd2 = (long *) ANABRD2;
   probrd0 = (long *) PROBRD0;
   probrd1 = (long *) PROBRD1;
   probrd2 = (long *) PROBRD2;

/* Initialize PC pointers */

   Comm0 = states;
   Comm1 = current;
   Comm2 = times;
   Comm3 = gains;
   Comm4 = params;
   Comm5 = transient[0];
   dspproceedflag = &Comm6;
   pcproceedflag = &Comm7;
   Comm8 = storet[0];
   Comm9 = controls;
   Comm10 = theta;
   Comm11 = old_theta;
   Comm12 = float_dummy;
   Comm13 = estates;
   control_mode = &Comm14;
   long_dummy = &Comm15;
   Comm16 = obsdata[0];
   Comm17 = avg_torque;
   Comm18 = init_error;
   Comm19 = event_int;

   return;
}

/************************************

init_motor

Spins the motor to trigger the index pulse and
calculate a home position.
(100 rpm with 60 degrees conduction)

Calibrates and starts the analog board to read the
current sensors.

************************************/

init_motor()
{
   long j,phase;

/* Initialize variables */
   delta_theta = params[0]*PI/180; /* angular
increment of simulation */
   R_p = params[1];            /* primary winding
resistance */
   G2R = params[2];            /* load motor
constant G^2/Ra */
   turns = params[3];          /* ratio of winding
turns to 160 turns */
   turns = turns/160.0;
   tol = params[4];            /* root-finding
tolerance */
   V_b = params[5];            /* battery voltage */
   J = params[6];              /* load inertia */
   B = params[7];              /* load viscous
damping */
   T_s = params[8];            /* controller update
time */
   om_h_s = params[9];         /* controller
saturation limit */
   coulombic = params[10];     /* coulombic
load torque */
   i_chop = params[11];        /* current
chopping limit */
   i_hyst = params[12];        /* current chopping
hysteresis */
   maxslew = params[13];       /* control angle
max slew rate */

   odata = convert_matrix(obsdata[0],1,2,1,181);
   spline(odata[2],odata[1],181,0.,0.,odata2);

   states[0] = 0.0;
   states[2] = 0.0;
   estates[1] = states[1];
   estates[2] = states[2];
   times[1] = T_s;
   times[3] = 0.0;
   times[4] = 0.0;
   times[5] = 0.0;
   nom_on = 0.0;
   nom_cond = 0.0;
   avg_torque[0] = 0.0;
   init_error[1] = 0.0;
   init_error[2] = 0.0;

/* Set up time counter */
   *timctl0 = RSTINTI;
```

```
  *period0 = 0x0ffffffff;
  *timctl0 = SETINTI;

/* Set up command interval timer */

  *timctl1 = RSTINTI;
  *counts1 = 0;
  *period1 = (long)(0.047/0.12e-6);
  index1 = 1;
  asm("  OR  200h,IE");
  asm("  OR  2000h,ST");

/* Calibrate analog board, then track inputs */

  *anabrd0 = SAMSTRT;
  *anabrd0 = CALSTRT;
  *anabrd0 = SAMSTRT;
  while(0 == (*anabrd0 & CALDONE))
    ;
  *anabrd0 = rdcur[2];
  *anabrd1 = ANATIME;
  while(0 == (*anabrd0 & EOC))
    ;
  current[2] = (*anabrd2 >> 20);

  for(phase=0;phase<3;phase++)
  {
    *anabrd0 = rdcur[phase];
    *anabrd1 = ANATIME;
    while(0 == (*anabrd0 & EOC))
      ;
    coffset[phase] = ((float) (*anabrd2 >>
20))*13.3*2.5/2048;
    current[phase] = 0.0;
  }
  /* Battery supply voltage */
  *anabrd0 = RDBATT;
  *anabrd1 = ANATIME;
  while(0 == (*anabrd0 & EOC))
    ;
  battery = ((float) (*anabrd2 >>
20))*201*2.5/2048;
  float_dummy[0] = battery;
}

/************************************

spin_motor()

Spins the motor until the index has been passed.
Starts the speed calculation using the index pulse
and a DSP timer as
a counter.

************************************/
spin_motor(pinv_command,pold_angle)
long  *pinv_command;
float *pold_angle;
{
/* Spin motor */
/* Enable *INT0 from index pulse and data record
counter */

  init_counts0 = 1;
  asm("  OR    1h,IE");
  *counts0 = 0;
  *probrd1 = PHONA;
  while(*counts0 < (0.07/0.12e-6))
    ;
  *timctl1 = SETINTI;
  while(init_counts0 == 1)
    ;
  *probrd1 = PHOFF;
  asm("  ANDN   200h,IE");
  *period1 = 0x0ffffffff;
  *counts1 = 0;

  *pold_angle = fmod(((float)((*probrd0 >> 16)
&  0x00000fff))*2*PI/2048.0
        + 2*PI - HOME_ANGLE,PI/2);
  *pinv_command = 0;
}

/************************************

store_data

************************************/

store_data(pcommand,pintgrl,pt)
float *pcommand,*pintgrl;
long *pt;
{
  transient[*pt][0] = states[0];
  transient[*pt][1] = float_dummy[2];
  transient[*pt][2] = states[2]*30/PI;
  transient[*pt][3] = float_dummy[3];
  transient[*pt][4] = float_dummy[4];
  transient[*pt][5] = (estates[1]-
states[1])*180/PI;
  transient[*pt][6] = (estates[2]-
states[2])*30/PI;
  (*pt)++;

  return;
}

/************************************

controller

Calculates new turn on and conduction angles
using P-I control gains.
```

```
**********************************/

controller(pcommand,pintgrl)
float *pcommand,*pintgrl;
{
  float
i_chop_limit,old_pcom,old_intgrl,old_on,old_c
ond,old_chop,omega_hat,turn_on_limit;

  if(*control_mode >= 4)
  {
    omega_hat = estates[2] - controls[3];
    delta_theta = 2*estates[2]*ESTTIME;
  }
  else
    omega_hat = states[2] - controls[3];

  old_on = controls[0];
  old_cond = controls[1];
  old_pcom = *pcommand;
  old_intgrl = *pintgrl;
  *pintgrl = old_intgrl + omega_hat * times[4];
  *pcommand = gains[0] * omega_hat + gains[1]
* old_intgrl;

/* Limits command error and updates nominal
control angle */
  if (fabs (*pcommand) > om_h_s)
  {
    *pintgrl = old_intgrl;
    if(*pcommand < 0)
      *pcommand = -om_h_s;
    else
      *pcommand = om_h_s;
  }

  on_limit (&turn_on_limit);
  /*
  controls[0] = nom_on + gains[2] *
(*pcommand);
  Force turn on angle to max torque turn on angle
*/
  if(controls[0] > turn_on_limit)
    controls[0] = nom_on + gains[2] * (-
om_h_s);
  if((controls[0] - old_on) > gains[2]*maxslew)
    controls[0] = old_on + gains[2]*maxslew;
  if((controls[0] - old_on) < - gains[2]*maxslew)
    controls[0] = old_on - gains[2]*maxslew;

  if (controls[0] > PI/2.0)
    controls[0] = PI/2.0;
  else if (controls[0] < turn_on_limit)
    controls[0] = turn_on_limit;

  controls[1] = nom_cond + gains[3] *
(*pcommand);
  if((controls[1] - old_cond) > -
gains[3]*maxslew)
    controls[1] = old_cond - gains[3] * maxslew;
  if((controls[1] - old_cond) <
gains[3]*maxslew)
    controls[1] = old_cond + gains[3] *
maxslew;
  if (controls[1] > MAXCOND*PI/180)
    controls[1] = MAXCOND*PI/180;
  else if (controls[1] < delta_theta)
    controls[1] = delta_theta;
  return;
}

/************************************

on_limit()

Calculates the lower limit for the turn_on angle.

**********************************/

on_limit(plimit)
float *plimit;
{

float m,speed1,turn_on1,speedl;

if(*control_mode >= 4)
  speedl = estates[2];
else
  speedl = states[2];

if (speedl <= 209.440)
  {
  m = -13.0 / 209.440;
  turn_on1 = 45.0;
  speed1 = 0.0;
  }
else if (speedl > 209.440 && speedl <= 418.879)
  {
  m = -4.0 / 209.440;
  turn_on1 = 32.0;
  speed1 = 209.440;
  }
else if (speedl > 418.879 && speedl <= 628.319)
  {
  m = -3.0 / 209.440;
  turn_on1 = 28.0;
  speed1 = 418.879;
  }
else if (speedl > 628.319 && speedl <= 837.750)
  {
  m = -3.0 / 209.440;
  turn_on1 = 25.0;
  speed1 = 628.319;
  }
```

```
else if (speed1 > 837.750 && speed1 <=
1047.200)
  {
  m = -2.0 / 209.440;
  turn_on1 = 22.0;
  speed1 = 837.750;
  }
else if (speed1 > 1047.200 && speed1 <=
1256.640)
  {
  m = -2.0 / 209.440;
  turn_on1 = 20.0;
  speed1 = 1047.200;
  }
else
  {
  m = 0.0;
  turn_on1 = 18.0;
  speed1 = 1256.640;
  }

*plimit = (turn_on1 + m * (speed1 -
speed1))*PI/180.0;

return;
}

/************************************

estimator()

************************************/

estimator(etime)
float etime[];
{
  long phase,temp;
  float obs_error,obs_angle,ei[3],real_angle;

  new_obs = 0;
  estates[1] = fmod(estates[1] + dtime *
estates[2],PI/2);

  for(phase=0;phase < 3;phase++)
  {
    *anabrd0 = rdcur[phase];

/* Update observer */
    if(etime[phase] < ESTTIME)
    {
      while((0.12e-6*((float)(*counts0))) <
ESTTIME)
        ;
      *anabrd1 = ANATIME;
      while(0 == (*anabrd0 & EOC))
        ;
      current[phase] = ((float) (*anabrd2 >>
20))*13.3*2.5/2048 - coffset[phase];
      real_angle = fmod(((float)((*probrd0 >>
16) & 0x00000fff))*2*PI/2048.0 + 2*PI -
HOME_ANGLE,PI/2);
      etime[phase] = 0.12e-
6*((float)(*counts0));
      obs_angle =
observer(estates[1],current[phase],phase);
      new_obs = 1;

      float_dummy[2] = obs_angle*180/PI;
      float_dummy[3] = real_angle*180/PI;
      float_dummy[4] = estates[1]*180/PI;
    }
  }

  obs_error = obs_angle - estates[1];
  if(obs_error > PI/4)
    obs_error = obs_error - PI/2;
  else if(obs_error < -PI/4)
    obs_error = PI/2 + obs_error;
  estates[1] = fmod(estates[1] +
new_obs*gains[4]*obs_error + PI/2,PI/2);
  estates[2] = estates[2] + dtime * (avg_torque[0]
- B*estates[2])/J +
new_obs*gains[5]*obs_error;
  /*
  float_dummy[2] = avg_torque[0];
  float_dummy[3] = dtime*(avg_torque[0]-
B*estates[2])/J;
  float_dummy[4] =
new_obs*gains[5]*obs_error;
  */
  return;
}

/************************************/

float observer(float eangle,float ei,long phase)
{
  float obs_angle;
  float angle;
  ei = ei*30.0/battery;
  if(ei > odata[2][181])
/*
    angle = odata[1][181];
  else

  splint(odata[2],odata[1],odata2,181,ei,&angle
);
  if((controls[0] + ESTTIME*estates[2]) > PI/4)
    angle = PI/2 - angle;
  obs_angle = fmod(angle + 2*PI -
phase*PI/3,PI/2);
*/
    obs_angle = estates[1];
```

```
  else if (ei < odata[2][1])
    obs_angle = estates[1];
  else
  {

  splint(odata[2],odata[1],odata2,181,ei,&angle
);
    if((controls[0] + ESTTIME*estates[2]) >
PI/4)
      angle = PI/2 - angle;
    obs_angle = fmod(angle + 2*PI -
phase*PI/3,PI/2);
  }
  float_dummy[1] = angle;
  return(obs_angle);
}

/************************************/

splint(float xa[],float ya[],float y2a[],int n,float
x,float *y)
{
  int klo,khi,k;
  float h,b,a;

  klo=1;
  khi=n;
  while (khi-klo > 1)
  {
    k=(khi+klo) >> 1;
    if (xa[k] > x)
      khi=k;
    else
      klo=k;
  }
  h=xa[khi]-xa[klo];
  if (h == 0.0)
    *y=ya[klo];
  else
  {
    a=(xa[khi]-x)/h;
    b=(x-xa[klo])/h;
    *y=a*ya[klo]+b*ya[khi]+((a*a*a-
a)*y2a[klo]+(b*b*b-b)*y2a[khi])*(h*h)/6.0;
  }
  return;
}

/************************************/

spline(float x[],float y[],int n,float yp1,float
ypn,float y2[])
{
  int i,k;
  float p,qn,sig,un,*u;

  u=vector(1,n-1);
  if (yp1 > 0.99e30)
    y2[1]=u[1]=0.0;
  else
  {
    y2[1]=-0.5;
    u[1]=(3.0/(x[2]-x[1]))*((y[2]-y[1])/(x[2]-
x[1])-yp1);
  }
  for (i=2;i<=n-1;i++)
  {
    sig=(x[i]-x[i-1])/(x[i+1]-x[i-1]);
    p=sig*y2[i-1]+2.0;
    y2[i]=(sig-1.0)/p;
    u[i]=(y[i+1]-y[i])/(x[i+1]-x[i]) - (y[i]-y[i-
1])/(x[i]-x[i-1]);
    u[i]=(6.0*u[i]/(x[i+1]-x[i-1])-sig*u[i-1])/p;
  }
  if (ypn > 0.99e30)
    qn=un=0.0;
  else
  {
    qn=0.5;
    un=(3.0/(x[n]-x[n-1]))*(ypn-(y[n]-y[n-
1])/(x[n]-x[n-1]));
  }
  y2[n]=(un-qn*u[n-1])/(qn*y2[n-1]+1.0);
  for (k=n-1;k>=1;k--)
    y2[k]=y2[k]*y2[k+1]+u[k];
  free_vector(u,1,n-1);

  return;
}

/************************************/

nrerror(error_text)
char error_text[];
{
  /*
  void exit();
  fprintf(stderr,"Numerical Recipes run-time error
...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
  */
  return;
}

/************************************/

float *vector(nl,nh)
int nl,nh;
{
  float *v;
```

```
  v=(float *)malloc((unsigned) (nh-
nl+1)*sizeof(float));
  if (!v) nrerror("allocation failure in vector()");
  return v-nl;
}

/************************************/

free_vector(v,nl,nh)
float *v;
int nl,nh;
{
  free((char*) (v+nl));
}

/************************************/

float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
  int i;
  float **m;

  m=(float **) malloc((unsigned) (nrh-
nrl+1)*sizeof(float*));
  if (!m) nrerror("allocation failure 1 in
matrix()");
  m -= nrl;

  for(i=nrl;i<=nrh;i++) {
    m[i]=(float *) malloc((unsigned) (nch-
ncl+1)*sizeof(float));
    if (!m[i]) nrerror("allocation failure 2 in
matrix()");
    m[i] -= ncl;

  }

  return m;
}

/************************************/

free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
  int i;

  for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
  free((char*) (m+nrl));
}

/************************************/

float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
  int i,j,nrow,ncol;
  float **m;

  nrow=nrh-nrl+1;
  ncol=nch-ncl+1;

  m = (float **) malloc((unsigned)
(nrow)*sizeof(float*));
  if (!m) nrerror("allocation failure in
convert_matrix()");
  m -= nrl;
  for(i=0,j=nrl;i<=nrow-1;i++,j++)
m[j]=a+ncol*i-ncl;
  return m;
}

/************************************

vrm_commands()

Updates command to inverters on transitions
through turn_on and turn_off
angles.  All angles calculated modulo PI/2.

************************************/

vrm_commands(pinv_command,pold_angle,etim
e)
long  *pinv_command;
float *pold_angle,etime[3];
{
  float theta_off, new_angle;
  long phase;

/* Read angle */

  new_angle = ((float)((*probrd0 >> 16) &
0x00000fff))*2*PI/2048.0;
  if((new_angle > PI/2) && (new_angle <
3*PI/2))
    intmask = 1;
  new_angle = fmod(new_angle + 2*PI -
HOME_ANGLE,PI/2);
  if((states[2] > 150) && (fabs(new_angle -
*pold_angle) <= (PI/3)) && (fabs(new_angle -
*pold_angle) > PI/6))
    new_angle = fmod(*pold_angle +
dtime*states[2],PI/2);
  states[1] = new_angle;
  if(*control_mode >= 3)
    new_angle = estates[1];
  theta_off =
fmod(controls[0]+controls[1],PI/2);

  for(phase=0;phase < 3;phase++)
```

```
  {
    theta[phase] = fmod(new_angle +
phase*PI/3,PI/2);
    old_theta[phase] = fmod(*pold_angle +
phase*PI/3,PI/2);

    /* Determine new commands */
    if(! new_obs)
    {
      if((((theta_off > controls[0]) &&
(theta[phase] >= controls[0]) && (theta[phase] <
theta_off))
        || ((theta_off < controls[0]) &&
((theta[phase] >= controls[0]) || (theta[phase] <
theta_off))))
        && (!(*pinv_command &
turn_on[phase])))
      {
        *pinv_command = *pinv_command |
turn_on[phase];
        etime[phase] = 0.0;
      }

      if(((theta_off > controls[0]) &&
((theta[phase] < controls[0]) || (theta[phase] >=
theta_off)))
        || ((theta_off < controls[0]) &&
((theta[phase] < controls[0]) && (theta[phase] >=
theta_off))))
        *pinv_command = *pinv_command &
turn_off[phase];
    }
  }

  *pold_angle = new_angle;
  *probrd1 = *pinv_command;

  /* Save loop time stamp */
  dtime = 0.12e-6*((float)(*counts0));
  *counts0 = 0;
  return;
}

/************************************/

shutdown_data(inv_command,command,intgrl,pt
)
float command,intgrl;
long inv_command,*pt;
{
  long phase;

  if(*pt<M)
  {
  storet[*pt][0] = states[0] + 0.12e-
6*((float)(*counts0));
  storet[*pt][1] = states[1]*180/PI;
  storet[*pt][2] = states[2]*30/PI;
  storet[*pt][3] = (float)(inv_command >> 28);
  storet[*pt][4] = float_dummy[2];
  storet[*pt][5] = float_dummy[3];
  storet[*pt][6] = float_dummy[4];
  storet[*pt][7] = estates[1]*180/PI;
  storet[*pt][8] = estates[2]*30/PI;
  storet[*pt][9] = controls[0]*180/PI;
  storet[*pt][10] = controls[1]*180/PI;
  (*pt)++;
  }
/*
  if((fabs(estates[2] - states[2])/states[2] < 0.05)
&& (*pt >= M))
  if((*pt >= M) && (revolutions < 4))
  if(*pt >= M)
    *pt = 0;
*/

}


/* Place address of ISR in vector at "bottom" of
memory */

asm("   .sect   \".int01\"");
asm("   .word   _c_int01 ");
asm("   .text         ");

/************************************
  INTERRUPT SERVICE ROUTINE - EXTERNAL
*INT0 INDEX PULSE
************************************/
c_int01()
{
  float zero_pos;

  zero_pos = ((float)((*probrd0 >> 16) &
0x00000fff))*2*PI/2048.0;

  if(init_counts0 == 1)
  {
    *probrd2 = 0;
    states[2] = 0;
    states[0] = 0;
    init_counts0 = 0;
    *counts1 = 0;
  }
  else if((intmask) && ((zero_pos > 11*PI/6) ||
(zero_pos < PI/6)))
  {
    intmask = 0;
    *probrd2 = 0;
    states[2] = 2*PI/(0.12e-
6*((float)(*counts1)));
    *counts1 = 0;
  }
```

```
  if(states[2] > controls[2])
    revolutions++;
  else
    revolutions = 0;
}

asm("   .sect   \".int10\"");
asm("   .word   _c_int10 ");
asm("   .text          ");

/*************************************
  INTERRUPT SERVICE ROUTINE - INTERNAL
TIMER 1 COMMAND INTERVALS
*************************************/
c_int10()
{

  /* Send command */

  *probrd1 = turn_on[(long)(fmod(index1,3))];

  /* Index to new command */
  ++index1;
}    /* Return */
```

DSP.H

```
#define PRIMCTL 0x00808064    /* Primary bus
control register */
#define EXPCTL  0x00808060    /* Expansion
bus control register */

#define PRIMWD  0x00000800    /* Zero wait
state, 64K bank switching */
#define EXPWD   0x00000000    /* Zero wait
state */

#define ADCDACA 0x00804000    /* ADC/DAC
channel A */
#define ADCDACB 0x00804001    /* ADC/DAC
channel B */
#define ADCDACT 0x00804008    /* ADC/DAC
conversion trigger */

#define TIMCTL0 0x00808020    /* Timer0
control register */
#define PERIOD0 0x00808028    /* Timer0
period register */
#define COUNT0  0x00808024    /* Timer0
counter register */
#define TIMCTL1 0x00808030    /* Timer1
control register */
#define PERIOD1 0x00808038    /* Timer1
period register */
#define COUNT1  0x00808034    /* Timer1
counter register */

#define RSTEXTI 0x00000601    /* Hold timer:
int clk, ext INT */
#define SETEXTI 0x000006C1    /* Reset, start
timer: int clk, ext INT */
#define RSTINTI 0x00000600    /* Hold timer:
int clk, no INT */
#define SRTINTI 0x00000680    /* Start timer:
int clk, no INT */
#define SETINTI 0x000006C0    /* Reset, start
timer: int clk, no INT */

#define PROBRD0 0x00800004    /* Prototype
board Position (R) */
#define PROBRD1 0x00800005    /* Prototype
board Commands (W) */
#define PROBRD2 0x00800006    /* Prototype
board Reset Clock (W) */
#define PROBRD3 0x00800007    /* Prototype
board R/W 3 */

#define ANABRD0 0x00800008    /* Analog
board Control(W)/Status(R) Register */
#define ANABRD1 0x00800009    /* Analog
board Timer Register (W) */
#define ANABRD2 0x0080000a    /* Analog
board Current (R) */
#define ANABRD3 0x0080000b    /* Analog
board port 3 */

#define CALSTRT 0x200000      /* Start analog
board calibration */
#define SAMSTRT 0x00000       /* Start current
sampling */
#define RDCURA  0x400000      /* Initiate phase
A current conversion */
#define RDCURB  0x410000      /* Initiate phase
B current conversion */
#define RDCURC  0x420000      /* Initiate phase
C current conversion */
#define RDBATT  0x430000      /* Initiate
battery voltage conversion */
#define CALDONE 0x200000      /* Analog board
calibration done */
#define EOC     0x800000      /* Analog board
end-of-conversion */
#define ANATIME 0x000000000   /* S/W
triggered analog sampling */

#define PHONA   0x10000000
#define PHONB   0x20000000
#define PHONC   0x40000000
#define PHOFFA  0x60000000
#define PHOFFB  0x50000000
#define PHOFFC  0x30000000
#define PHOFF   0x00000000
#define HOME_ANGLE (31.5*PI/180.0)
```

```
long    *counts0,*timctl0,*period0;
long    *counts1,*timctl1,*period1;
long    *anabrd0,*anabrd1,*anabrd2;
long    *probrd0,*probrd1,*probrd2;
long  rdcur[] = {RDCURA,RDCURC,RDCURB};
long  turn_on[] = {PHONA,PHONC,PHONB};
long  turn_off[] = {PHOFFA,PHOFFC,PHOFFB};
```

PCDSP.H

```
#define PI  3.141592654
#define M  300
#define N  11
#define CM  300
#define CN  7
#define MR  4
#define MC  31
#define BUGS 5
#define ESTTIME 69.0e-6
#define MAXCOND 25
```

MOTOR.H

```
float t_lock;     /* lock out time built into
controller */
float delta_theta; /* angular increment of
simulation (degrees) */
float R_p;
float G2R;
float tol;
float turns;
float V_b;
float J;
float B;
float T_s;
float om_h_s;
float coulombic;
float i_chop;
float i_hyst;
float maxslew;
float  nom_on,nom_cond;
```

MOTOR.DAT

```
0.151906  -0.305662   0.0024933
0.151619  -0.298114   0.0025087
0.148072  -0.284335   0.00252211
0.139373  -0.279143   0.0025614
0.129272  -0.277409   0.0026616
0.12022   -0.276613   0.00274118
0.109505  -0.275656   0.00282422
0.086498  -0.271217   0.00298924
0.075357  -0.267324   0.00307126
0.064814  -0.262365   0.00315628
0.054813  -0.254499   0.00324130
0.0450    -0.240      0.0033232
0.03521   -0.216162   0.00338938
0.012649  -0.109607   0.00348542
0.007395  -0.033026   0.00349545
0.007056  -0.005278   0.003548
0.007395  -0.033026   0.00349552
0.012649  -0.109607   0.00348558
0.03521   -0.216162   0.00338960
0.0450    -0.240      0.0033262
0.054813  -0.254499   0.00324164
0.064814  -0.262365   0.00315666
0.075357  -0.267324   0.00307168
0.086498  -0.271217   0.00298972
0.109505  -0.275656   0.00282474
0.12022   -0.276613   0.00274176
0.129272  -0.277409   0.0026679
0.139373  -0.279143   0.0025683
0.148072  -0.284335   0.00252287
0.151619  -0.298114   0.00250890
0.151906  -0.305662   0.002493
```

PARAM.DAT

```
1.0
0.800
0.00317
160.0
0.001
30.0
0.00708
0.000531
0.004
50.0
0.7125
20.0
0.654
1.0
delta_theta .25    [degrees]
Rp        0.800  [ohms]
G2R       .00317  [Nm/A^2.rps]
turns     160.0  [turns]
tol       0.001  [amps]
V_b       160.0  [volts]
J         0.00015 [kg.m^2/rad]
B         0.00136 [kg.m^2/rad.s]
T_s       0.004  [sec]
om_h_s    50.0   [rad]
coulombic  0.7125  [Nm]
i_chop    20.0   [A]
i_hyst    0.654  [A]
maxslew   8.0    [rad/sec]
```

GAINS.DAT

```
0.5
0.5
0.00873
-0.0131
0.37
32.0
Kp 0.5   [sec]
Ki 0.5   []
Kon 0.00873 []
Kcd -.00873 []
Lt 0.4   []
Lo 18.9   [1/sec]
```

DATAES03.TXT

```
0    0.081177
0.25  0.081177
0.5 0.081177
0.75  0.081177
1    0.081177
1.25  0.081707
1.5 0.082386
1.75  0.083065
2    0.083745
2.25  0.084424
2.5 0.085103
2.75  0.085782
3    0.086461
3.25  0.08714
3.5 0.087819
3.75  0.088499
4    0.089178
4.25  0.089295
4.5 0.089295
4.75  0.089295
5    0.089295
5.25  0.089295
5.5 0.089295
5.75  0.089295
6    0.089295
6.25  0.089295
6.5 0.089295
6.75  0.089295
7    0.089295
7.25  0.089295
7.5 0.089295
7.75  0.089295
8    0.089295
8.25  0.089295
8.5 0.089295
8.75  0.089351
9    0.090072
9.25  0.090794
9.5 0.091516
9.75  0.092237
10  0.092959
10.25   0.09368
10.5  0.094402
10.75   0.095123
11  0.095845
11.25   0.096566
11.5  0.097288
11.75   0.097412
12  0.097412
12.25   0.097412
12.5  0.097468
12.75   0.09819
13  0.098911
13.25   0.099633
13.5  0.100355
13.75   0.101076
14  0.101798
14.25   0.102722
14.5  0.104165
14.75   0.105609
15  0.107052
15.25   0.108495
15.5  0.109938
15.75   0.111381
16  0.112824
16.25   0.114267
16.5  0.115711
16.75   0.117154
17  0.118597
17.25   0.12004
17.5  0.121483
17.75   0.122926
18  0.124369
18.25   0.12584
18.5  0.127332
18.75   0.128823
19  0.130314
19.25   0.131805
19.5  0.133297
19.75   0.134788
20  0.136279
20.25   0.13777
20.5  0.139262
20.75   0.140774
21  0.142939
21.25   0.145103
21.5  0.147268
21.75   0.149433
22  0.151598
22.25   0.153762
22.5  0.155927
22.75   0.15818
23  0.160511
23.25   0.162843
```

23.5 0.165174
23.75 0.167439
24 0.169685
24.25 0.171932
24.5 0.174178
24.75 0.176424
25 0.179526
25.25 0.183963
25.5 0.1884
25.75 0.192837
26 0.197274
26.25 0.201711
26.5 0.206148
26.75 0.213284
27 0.220981
27.25 0.228678
27.5 0.233536
27.75 0.242588
28 0.25192
28.25 0.261252
28.5 0.270585
28.75 0.279917
29 0.289249
29.25 0.298607
29.5 0.30803
29.75 0.317453
30 0.326876
30.25 0.336299
30.5 0.346704
30.75 0.357314
31 0.367923
31.25 0.377295
31.5 0.386319
31.75 0.395344
32 0.404369
32.25 0.412691
32.5 0.419572
32.75 0.426454
33 0.432962
33.25 0.438516
33.5 0.44407
33.75 0.449623
34 0.455177
34.25 0.459281
34.5 0.463084
34.75 0.466887
35 0.471304
35.25 0.475892
35.5 0.480479
35.75 0.485067
36 0.489228
36.25 0.492514
36.5 0.495801
36.75 0.499197
37 0.502876
37.25 0.506555
37.5 0.510233
37.75 0.513912
38 0.516622
38.25 0.519131
38.5 0.52164
38.75 0.524111
39 0.526139
39.25 0.528167
39.5 0.530195
39.75 0.532084
40 0.533437
40.25 0.534399
40.5 0.535361
40.75 0.536323
41 0.537285
41.25 0.538247
41.5 0.539209
41.75 0.540198
42 0.541192
42.25 0.542186
42.5 0.54318
42.75 0.544231
43 0.54563
43.25 0.54703
43.5 0.548429
43.75 0.549828
44 0.551227
44.25 0.552347
44.5 0.552752
44.75 0.553157
45 0.553562

Bibliography

[1] A. Lumsdaine, *Control of a Variable Reluctance Motor Based on State Observation*, S.B. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, November 1985.

[2] W. D. Harris, *Practical Indirect Position Sensing for a Variable Reluctance Motor*, S.B. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1987.

[3] D. A. Torrey, *Optimal-efficiency Constant-speed Control of Nonlinear Variable Reluctance Motor Drives*, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1988.

[4] W. H. Press et al., *Numerical Recipes in C*, Cambridge: Cambridge University Press, pp. 94-98, 540-547, 1991.

[5] F. J. Vallese, *Design and Operation of High-power Variable Reluctance Motor Drive Systems*, Sc.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, March 1985.

[6] H. H. Woodson and J. R. Melcher, *Electromechanical Dynamics, Part I: Discrete Systems*, New York: John Wiley and Sons, Chapters 3 and 5, 1968.

[7] A. E. Fitzgerald, C. Kingsley, and S. Umans, *Electric Machinery*, New York: McGraw-Hill, Inc., Chapters 1 through 4, 1983.

[8] M. Ilic-Spong, R. Marino, S. Peresada and D. G. Taylor, "Nonlinear control of switched reluctance motors in robotics applications," Conference on Applied Motion Control, U. of Minnesota, pp. 129-136, 1986.

[9] D. G. Taylor, M. Ilic-Spong and S. Peresada, "Nonlinear composite control of switched reluctance motors," IEEE Industrial Electronics Conference, pp. 739-749, 1986.

[10] R. Thorton, "Mechanical Specifications for the Zanussi VRM," Laboratory for Electromagnetic and Electronic Systems, Massachusetts Institute of Technology, 1985.

[11] D. G. Luenberger, *Introduction to Dynamic Systems*, New York: John Wiley and Sons, Chapters 8, 9, and 11, 1979.

[12] G. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Reading, MA: Addison-Wesley Publishing Co., Chapters 6 and 8, 1986.

[13] G. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*, Reading, MA: Addison-Wesley Publishing Co., 1990.

[14] D. Cameron, *Origin and Reduction of Acoustic Noise in Variable-Reluctance Motors*, S.M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1991.

[15] J. G. Kassakian, M. E. Schlecht, G. C. Verghese, *Principles of Power Electronics*, Reading, MA: Addison-Wesley Publishing Co., Chapters 1 and 2, 1991.

[16] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall, 1978.

[17] T. J. E. Miller, *Switched Reluctance Motors and Their Control*, New York City, NY: Oxford University Press, 1993.

[18] T. J. E. Miller, *Switched Reluctance Motor Drives*, Ventura, CA: Intertec Communications Inc., 1988.

[19] T. J. E. Miller, *Brushless Permanent Magnet and Reluctance Motor Drives*, New York City, NY: Oxford University Press, 1989.