NONLINEAR DEADBEAT CURRENT CONTROL OF A SWITCHED RELUCTANCE MOTOR

Benjamin Rudolph

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Masters of Science

In

Electrical Engineering

APPROVED:

Dr. Krishnan Ramu, Chair

Dr. Daniel Stilwell

Dr. William Baumann

December 4, 2009

Blacksburg Virginia

Keywords:

switched reluctance motor, SRM, current control, nonlinear control, deadbeat

NONLINEAR DEADBEAT CURRENT CONTROL OF A SWITCHED RELUCTANCE MOTOR

Benjamin Rudolph

ABSTRACT

High performance current control is critical to the success of the switched reluctance motor (SRM). Yet high motor phase nonlinearities in the SRM place extra burden on the current controller, rendering it the weakest link in SRM control. In contrast to linear motor control techniques that respond to current error, the deadbeat controller calculates the control voltage by the current command, phase current, rotor position and applied phase voltage. The deadbeat controller has demonstrated superior response in three-phase inverter current control, PM motor current control, and other relatively linear control applications. This study will investigate the viability and performance of a deadbeat controller for the highly nonlinear SRM.

The need for an accurate deadbeat control model first motivates the investigation of experimental inductance measurement techniques. A deadbeat control law is then proposed through multiple revisions to demonstrate the benefit of the numerical method chosen to derive the controller and a current predictor that accounts for processor latency and PWM delay. The practical problems of loop delay, feedback noise, feedback filtering, and deadbeat controller parameter sensitivity are investigated by linear analysis, simulation, experimental implementation and nonlinear model analysis. Simulation and implementation verify deadbeat performance and various measures of transient performance are presented. To address the problem of SRM model error the study ends with a brief discussion of adaptive deadbeat control modifications for possible future research.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my advisor, Dr Krishnan Ramu, for providing the equipment and resources that made this paper possible. His encouragement to pursue my interests and support of my independent research are greatly appreciated. I also want to express my warmest appreciation for my colleague Nimal Lobo who introduced me to the topic of deadbeat control and offered valuable time and support during the early stages of my research. I also want to kindly thank Larry Pierce for volunteering his editing expertise to help edit this thesis.

1 Introduction

1.1 Background

The switched reluctance, or variable reluctance, motor is one of the lesser known motor types in industry. Although originally proposed by S.A Nasar for variable speed applications in 1969[1], it received little attention for a couple of decades. During the early years of motors, mechanically commutated DC machines and AC driven induction and synchronous machines dominated because they could be operated smoothly by readily available DC or 60Hz AC power sources. The growing demand for variable speed motor applications and the advent of high power electronics in the last 20 years drove the development of high-frequency switching semiconductor motor current controllers. While computing power and power electronics have grown cheaper and more reliable, the cost of motor copper and steel and fabrication has decreased little, reviving interest in cheaper motors driven by more sophisticated current control methods.

The switched reluctance motor (SRM) has a laminated steel rotor with no rotor field windings and is similar in form and function to a stepper motor, but with fewer poles. Its construction is cheap and rugged yet it is one of the least forgiving motors to control due to highly nonlinear electromagnetic characteristics attributed to rotor and stator saliency (varying air gap), concentric stator windings (not sinusoidally distributed), and the need for electronic current commutation. Furthermore, the SRM is usually operated in deep magnetic saturation to maximize its power-to-mass ratio. Perhaps the greatest obstacle to the success of the SRM is high acoustic noise produced by large radial forces and torque ripple. Although vibration may be mitigated partly by mechanical motor design, it may be reduced mainly by well-designed torque distribution schemes. These schemes may involve overlapping or mutually exclusive phase conduction, but in either case require precise tracking of an irregular current trajectory. Thus, the burden of SRM success in industrial application is largely set on the performance of its current controller, particularly the current control algorithm.

In most cases, SRM mutual phase coupling is very low, so the great majority of controllers in the literature use independent SISO control for each phase. Conventional hysteresis

current control has high transient performance but suffers from high current ripple and unpredictable frequency characteristics, among other shortcomings. Conventional PI control with integrator antiwindup reduces current ripple, but suffers in the area of transient performance and tracking. To combine high transient and tracking performance with low current ripple in a single controller, a number of more advanced current control techniques have been proposed for SRMs. Several model reference adaptive current controllers for SRMs have been proposed [2-4], as well as an iterative controller [5]. In general, most sophisticated current control schemes may be categorized in terms of model scope and model identification method. Adaptive control (model reference adaptive control) adjusts a parametric model online to minimize the error between the model reference states and observed states across a range of excitations spanning model dynamics. The iterative learning controller minimizes current error over a particular steady-state current trajectory by adjusting the parameters of a trajectory model with no assumption of model structure.

In contrast, deadbeat control relies on the accuracy of a predetermined system model to calculate the correct control input required to drive error to zero in a short time. It employs no adaptive techniques and is less computationally intensive than comparatively abstract adaptive counterparts, making it a good choice for high-speed control applications. Deadbeat control design is digital (discrete) by definition and has also been referenced rather generically as “high performance digital control” and “digital predictive control”. In the literature, deadbeat control has been proposed for linear multivariable control applications such as three phase VSI’s [6-9] and PM motors [10], but not for SRMs.

1.2 Thesis Proposal

This thesis will investigate the novel application of deadbeat current control to an SRM motor from a practical standpoint. Design, simulation and implementation will be presented for both deadbeat and PI current control to give perspective. Particular consideration will be given to the problems of loop delay and feedback noise. The deadbeat control law will be presented in three revisions -- control law A, control law B and control law C -- simulating each to demonstrate the necessity of the components in the final control law, C. Performance will be evaluated in terms of transient current response, current tracking, and computational burden.

Control law parameter sensitivity and noise sensitivity analysis will explore controller robustness and lend insights into controller design and weaknesses.

1.3 Organization of Materials Presented

The organization of this thesis will be as follows: Chapter 2 will briefly present the fundamentals of SRM physics and establish equations foundational to later analysis. Details concerning the motor, asymmetric phase converters, microprocessor and system control electronics are presented in Chapter 3. Chapter 4 will propose experimental SRM model automated measurement techniques and present implementation results in order to reinforce the credibility of FEA data and the underlying SRM model equations of Chapter 2. PWM configuration issues are discussed in Chapter 5 along with commutation control and speed control. Chapter 6 presents a linear analysis of the SRM current control problem including the effects of loop delay and current feedback filtering. The PI controller parameters are designed based on the linear model equations including nonlinear delay. A section analyzes noise rejection and closed loop current response. Chapter 7 presents the design of the deadbeat control law (control law C) with an in-depth discussion of the numerical methods used to discretize the SRM model. Special simplified cases of control law C are investigated, and one is used later for parameter sensitivity analysis. Chapter 8 presents simulation of PI and deadbeat current controllers including effects of noise and current filtering. Two filter compensation techniques with simulation are proposed to optimize current control performance. Various quantitative measures of transient performance including closed loop bandwidth are defined and summarized for all simulations. Challenges not encountered in simulation along with implementation results for PI and deadbeat control are investigated in Chapter 9. This chapter also addresses discrepancies between simulation and implementation, hysteresis results and large signal performance results, and benchmarking measurements to quantify the computation burden of the PI and deadbeat controllers implemented. Chapter 10 investigates deadbeat control law parameter sensitivity and noise sensitivity through simulation results, implementation results, and equations predicting control law and phase current error as a function of parameter and current feedback error.

1.4 Novel Contributions

The following developments originate with the author:

- Two methods for automating experimental inductance measurements (Chapter 4). The flux integration method is a variation on methods seen in literature[1], but adds built-in resistance estimation. The incremental inductance measurement technique has not been seen in literature and measures inductance by a technique that is highly immune to current signal noise and is independent of phase resistance.
- A nonlinear deadbeat current controller with predictive control to compensate delay (Chapter 7).
- Equations for analytically predicting the sensitivity of the deadbeat controller performance to inductance error and noise.
- A simple current feedback filter compensator implemented by adjusting sampling delay (section 8.2.2)

2 SRM Fundamentals

Most motor electromagnetic models can be represented simply by applying Faraday's law:

$$\frac{d\lambda(\theta, i, i_1 \ldots i_k, \lambda_f)}{dt} = v - Ri \tag{2.1}$$

The flux, λ, linking the phase coil is a function of the rotor position, θ, the phase current, i, the flux produced by the rotor, λ_f, and currents produced by other phases, i_1-i_n. Without rotor coils, a switched reluctance motor has no rotor field excitation. Phase coupling is usually low in SRMs and occurs only when currents conduct simultaneously in more than one phase. In this thesis, mutual inductance will be neglected, so the phase flux is a function of rotor angle and phase current:

$$\frac{d\lambda(\theta, i)}{dt} = v - Ri \tag{2.2}$$

Hereafter, it will be assumed that inductance and flux are functions of rotor position and phase current unless otherwise specified. Since flux is the product of current and inductance,

$$\lambda = iL$$

the product rule may be applied to the derivative on the left side of (2.2):

$$i\frac{dL}{dt} + L\frac{di}{dt} = v - Ri \tag{2.3}$$

Applying the chain rule for derivatives of multivariable equations to (2.2) and (2.3) respectively gives:

$$\frac{\partial\lambda}{\partial\theta}\omega + \frac{\partial\lambda}{\partial i}\frac{\partial i}{\partial t} = v - Ri \tag{2.4}$$

$$i\left(\frac{\partial L}{\partial\theta}\omega + \frac{\partial L}{\partial i}\frac{\partial i}{\partial t}\right) + L\frac{di}{dt} = v - Ri \tag{2.5}$$

where $\omega = \frac{\partial\theta}{\partial t}$, the rotor angular velocity in rad/sec. The first term in (2.5) is referred to as the back emf and is the product of the partial inductance derivative with respect to rotor position and

rotor speed. The second term in (2.5) contributes the effects of inductance saturation and is zero in the absence of saturation. The third term is the inductive voltage drop. Although (2.3) will be used for all later control derivations, (2.4) will be referenced to develop locked rotor inductance measurements. Equation (2.5) will be employed again in Chapter 6 for plant model linearization.

Mechanical dynamics for motors are the well known expressions relating air gap torque, T_e, to angular motion:

$$T_e(\theta, i) - T_l = J\frac{d\omega}{dt} + B\omega \tag{2.6}$$

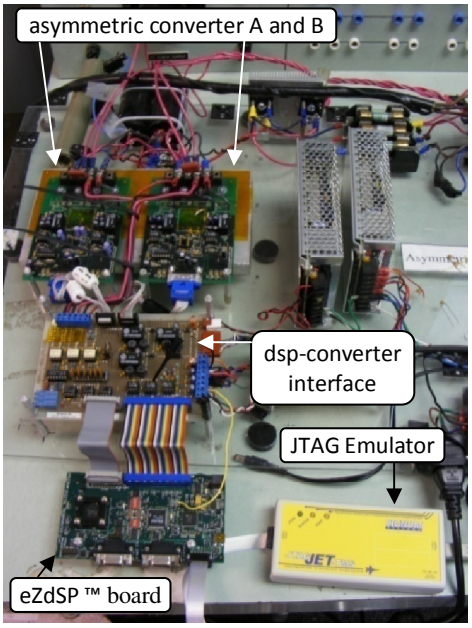
Here, J indicates the rotor angular inertia and B is the viscous friction constant. The notion of electrical angular velocity will not be used, so ω always refers to the mechanical velocity of the rotor. The air gap torque is the sum of the torque generated by each phase and is a function of current and rotor angle. In the absence of magnetic phase saturation, the phase torque is equal to the product of the inductance derivative with respect to angle and the phase current squared[1]:

$$T_e(\theta, i) = \frac{1}{2}i^2\frac{\partial L(\theta, i)}{\partial \theta} \tag{2.7}$$

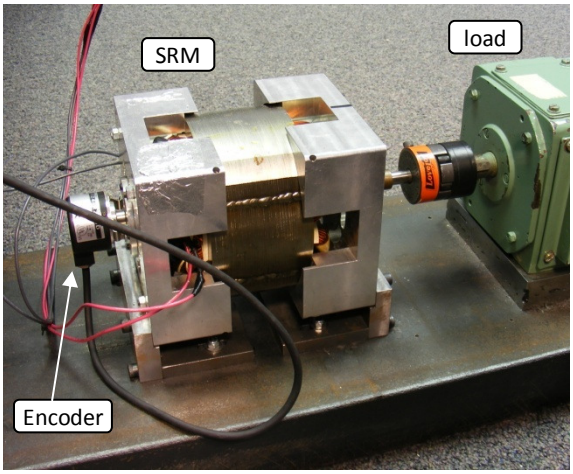
Phase torque may be derived from phase flux. It is important to note that all motor dynamics may be characterized by two electromagnetic variables, λ and R, and two mechanical parameters, J and B. Of course, the flux is a function of current and rotor and so contains a large amount of information. This information may also be represented in terms of inductance, L= λ/i. Aside from simulation, this thesis is not concerned with the mechanical dynamics of the SRM motor. Although current control dynamics are indeed tied to mechanical dynamics, the electromagnetic time constants of the motor in question are faster than its mechanical time constants. In this case, the electrical time constant, L/R, ranges from .02 to .1 sec, while the mechanical time constant, J/B, is about 2.3 sec (see constants in Appendix A). This is true for most motors. Thus, a high performance current controller may be designed independently of mechanical dynamics provided the rotor position is available to the controller.

3 Hardware Specifications and Operation

This thesis is based on a pre-existing setup from past projects. Figure 1 below displays the physical setup of the system electronics and machinery.



System control and power electronics.



SRM connected to load.

Figure 3.1. System hardware

The block diagram in Figure 3.1 shows the SRM motor drive system components relevant to this project. The dsp-signal interface board provides signal isolation necessitated by separate converter and dsp power supplies. It also scales feedback phase current and DC link voltage signals to useful dsp voltage levels and transmits PWM signals. The analog current feedback filter shown below in Figure 3.2 is a simple RC op-amp filter included on the dsp-converter signal interface board. It is a very significant component in control design, as will be seen later.

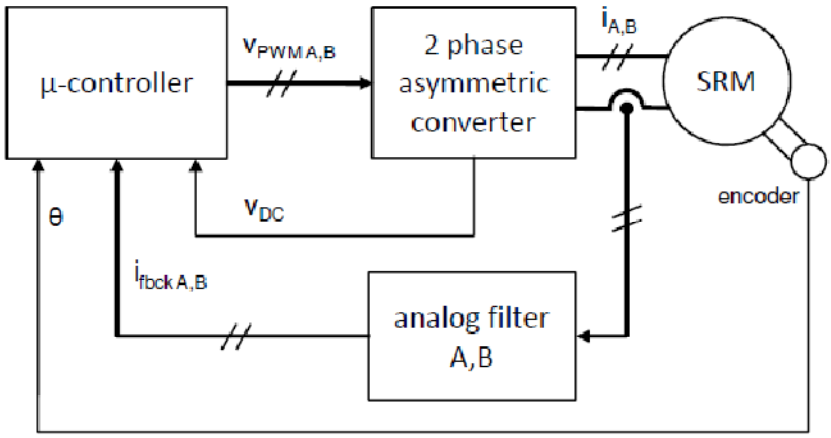


Figure 3.2. System block diagram

3.1 SRM Motor

A 6/3 machine prototype described in [11] was used. This is a 1500 watt two-phase switched reluctance machine with asymmetric inductance characteristics. The machine is rated at 3600rpm at 3.3 N-m. The motor is a 6/3 switched reluctance motor, meaning it has six salient stator poles and three salient rotor poles. A table of important motor specifications is given in Appendix A, and more details can be found in [11].

This motor was chosen because its magnetic and mechanical characteristics have been well documented, both by FEA and experiment. It is important for deadbeat current control that the electromagnetic motor model be modeled as accurately as possible. In addition, initial estimates of deadbeat current control algorithm complexity suggested a maximum algorithm frequency of 10 or 20 kHz. Of the SRMs immediately available, this SRM was judged a better choice for testing a deadbeat control algorithm because slower speeds would allow better tracking of the inductance trajectory, which is essential to deadbeat control. See section (controls section) for information on phase commutation of this motor. It is also convenient that mutual flux linkage is very low in this motor, such that it may be neglected in control design.

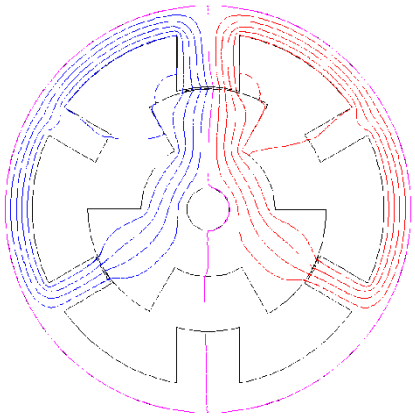


Figure 3.3. Finite element flux pattern of the two-phase 6/3 SRM ([11])

3.2 Asymmetric Converter

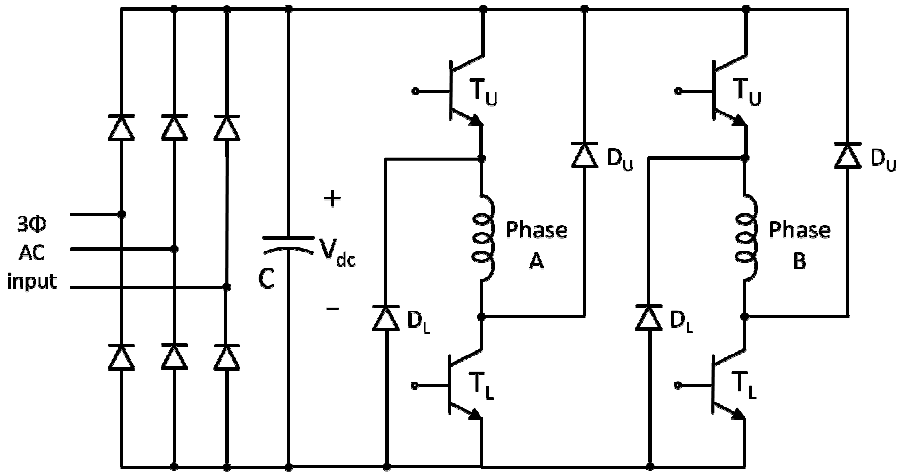


Figure 3.4. Two phase asymmetric converter with three phase rectifier voltage supply

Since SRM motors do not require bi-directional phase current, an asymmetric converter is used instead of a voltage source inverter. The asymmetric converter is designed particularly for SRM motors because it delivers unidirectional current, but bidirectional and zero voltage (freewheeling) when current is flowing. The ability to apply zero and negative dc voltage is enabled by the bypass diodes, which give the circuit its asymmetric design. The asymmetric converter is generally used for high performance SRM control because it incurs the high cost of

two switches per phase, but offers independent phase control. It also does not require a bifilar phase winding or split power supply, or even number of motor phases, as some simpler SRM converters do[12]. In order to fully evaluate the potential of a high performance control algorithm such as the deadbeat controller, it was important to use a higher performance converter capable of soft switching and independent phase control. The input to the converter comes from a large three-phase variac which allows the DC bus voltage to be set at any voltage desired up to 300V.

3.3 Microcontroller:

Device control is accomplished with an ez-Dsp development board with a TI C2000 F2808 microcontroller, as seen in Figure 3.5. The MCU is a 32-bit fixed point controller with 100Mhz clock and dsp type MAC operations. Although the F28x series are general purpose chips, they are especially designed for use in high performance motor controllers. Built-in peripherals include a highly configurable set of 16 PWMs and 6 high resolution PWMS, 16 12-bit ADCs, 3 32-bit CPU timers, 6 32-bit timers and 2 quadrature encoder interfaces. Code development was composed in C++ with the Code Composer Studio™ IDE and processor programming was enabled by the JTAG-Jet Tms emulator. Precompiled header files and libraries from the TI site were used to configure and interface with peripherals, particularly the ADC, PWM and quadrature encoder.

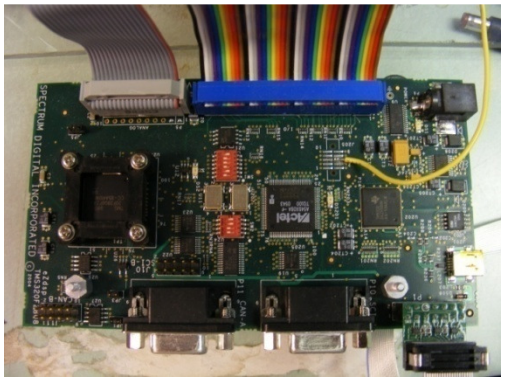
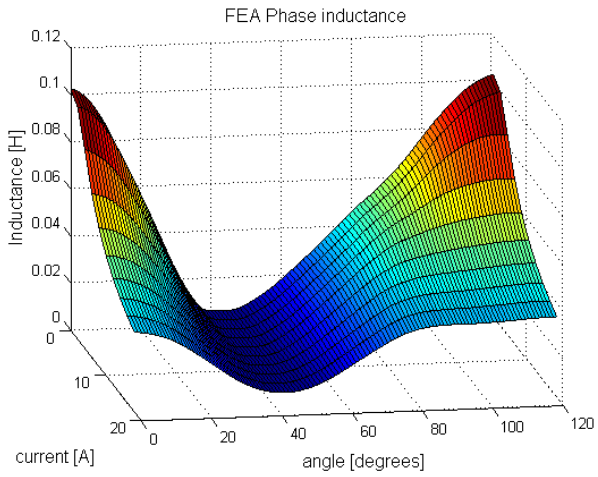


Figure 3.5. eZdSP ™ board

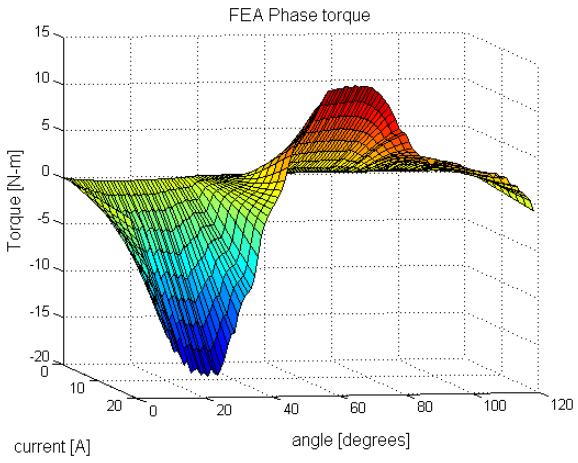
4 SRM Model Characterization

Unlike PI and other simple linear controllers known for robust control of systems with uncertain or unknown models, higher bandwidth controllers are highly dependent upon the accuracy of a system model. If we wish to design the highest bandwidth controller, such as the deadbeat controller, it is of paramount importance that we have an accurate model of the system. System characterization for switched reluctance motors has been a subject of ongoing study because the SRM electromagnetic model is highly nonlinear.

Inductance data can be found experimentally or calculated by a model through finite element analysis (FEA) for a particular motor and then represented in parametric form by equations that closely fit the data. Although FEA software yields good inductance calculations, it does not consider transient effects such as eddy currents or radial forces that lead to inductance variations caused by air gap variation from stator ovalization or bearing looseness. The FEA software uses 2D analysis so 3D characteristics such as motor end effects are not considered. Furthermore, there is always some discrepancy between ideal (FEA) machine saturation characteristics and those actually measured. FEA models simply do not model every physical motor detail, so it is important to determine the relative contribution from unmodeled dynamics for the particular motor before designing a deadbeat controller based on ideal FEA inductance data. FEA inductance and torque are shown in Figure 4.1 for the 6/3 SRM under test in this thesis. The FEA data was produced by colleague Keunsoo Ha in [13].



FEA inductance data as a function of current and mechanical angle



FEA torque data as a function of current and mechanical angle

Figure 4.1. Inductance and torque data from FEA

Investigating different inductance measurement techniques also lends insight into the viability of online inductance identification, which is essential to adaptive control. This thesis discusses the significance of online system identification for deadbeat control more in the last chapter.

Since the exact magnetic characteristics of an SRM are unknown and highly nonlinear, full characterization of the magnetic dynamics of an SRM is a very tedious task. This would require excitation of the machine for every operating condition and every type of dynamic operating situation that could be expected during motor operation. Taken to the extreme, experimental magnetic characterization attempts to answer the question, "What does the current do if X voltage is applied to the machine phase under Y circumstances?" by applying every possible voltage excitation for every possible operating condition. In other words, excitation signals applied to characterize a system are selected to excite every known dynamic of the system. Unknown dynamics may be discovered by an infinite variety of random excitations. However, this would generate an enormous amount of data far too cumbersome to implement in any motor controller. Thus, we seek a somewhat simplified model that estimates the machine characteristics under the excitation conditions expected for motor operation. Note, that if the

expected operating range for a motor is very narrow, such as one particular speed and current, an inductance model may be determined for only that excitation condition and used in a controller with exceptional performance. If we desire good performance over every possible range of operating conditions and motor dynamics, however, then we must sacrifice model precision and accuracy for versatility by adopting a more generalized model approximation with simplified subcomponents. Although the primary objective of the deadbeat controller is to provide optimal transient response and tracking, it also aims to perform well under a wide range of operating conditions.

For most SRM motors, including the motor used in this thesis, the phase inductance may be considered a function of its current and rotor angle only. Mutual inductance between other phases is normally not modeled because it is relatively small, and because phase excitation overlap is minimized in practice. Among proposed parametric SRM models are Fourier series models [14], [15] and [16], spline models [4], and geometric models [17]. However, since parametric models require many calculations involving complex algebra and trigonometric functions, they are often too complex to implement in a real-time controller. For this thesis, a 2D lookup table with 2D linear interpolation algorithm is used to represent phase inductance data. With enough current and angle points, an arbitrary inductance function can be represented with a high degree of precision and a low computational burden. More memory is required to store the data, but memory storage for such a lookup table is generally not a problem for a modern controller.

Since inductance and flux are independent of rotor speed, the data is normally measured by a locked rotor measurement where the phase current response to various voltage excitations is observed. There are many methods for measuring the inductance of an SRM. Two in particular are investigated in this thesis, and will be referred to as the voltage integration method [18], [19] and the incremental inductance method, which is developed partly in [1] and [12]. The latter method is based on the current rise time and fall time methods developed for online incremental inductance measurements. Both methods are modified to account for phase resistance and so may be considered resistance-independent.

4.1 Voltage Integration Method

This is perhaps the simplest and most versatile method for calculating phase inductance. Both sides of (2.2) are integrated to yield flux:

$$\lambda\left(\theta(t_f), i(t_f)\right) = \lambda\big(\theta(t_0), i(t_0)\big) + \int_{t_0}^{t_f} \big(v(t) - Ri(t)\big)dt \tag{4.1}$$

If the rotor is locked, $\theta(t_f) = \theta(t_0)$, then we may obtain the flux as a function of current by integrating v-iR for each rotor angle independently. If the initial flux is zero, implying initial current is zero, then the flux is:

$$\lambda_\theta\left(i(t_f)\right) = \int_{t_0}^{t_f} \big(v(t) - Ri(t)\big)dt \tag{4.2}$$

For a given rotor angle it is desired to calculate the flux at equal current intervals up to the maximum current desired, i_{max}.

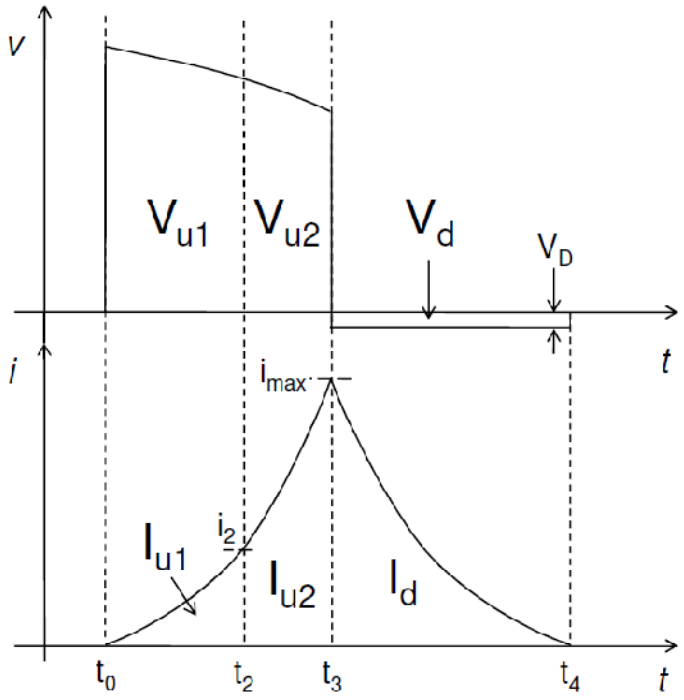


Figure 4.2. Voltage and current waveforms for voltage integration method

In terms of the definite integrals of the phase voltage and current curves in Figure 4.2, the flux at current i_2 can be calculated as:

$$\lambda_\theta(i_2) = V_{u1} - I_{u1}R \tag{4.3}$$

Phase resistance may be calculated very simply from (4.1) by integrating phase voltage and phase current over the entire excitation period to obtain integrals V and I. At the beginning and ends of the excitation period current is zero and so flux is zero. Thus, (4.1) becomes

$$0 = \int_{v(t_0),i(t_0)=0}^{v(t_f),i(t_f)=0} \big(v(t) - Ri(t)\big)dt = V - IR$$

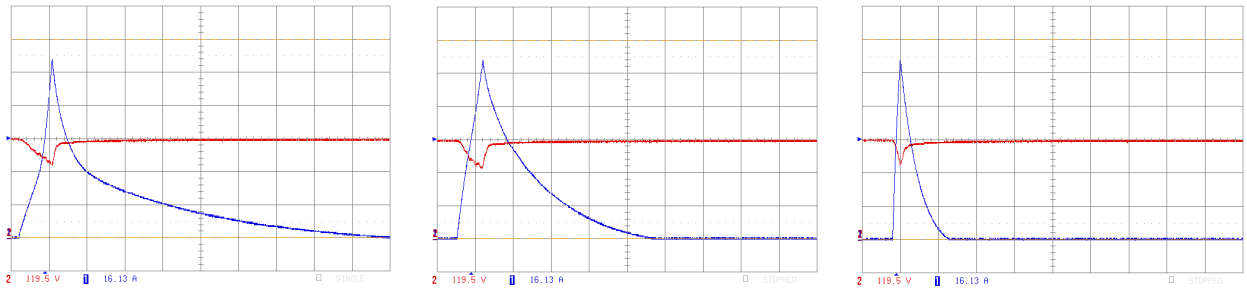
and resistance is simply:

$$R = V/I \tag{4.4}$$

The relation holds regardless of whether the rotor is locked, so the measurement may be applied during motor operation to measure phase resistance very accurately.

Since the asymmetric converter was available and the TI microcontroller was set up to sample capacitor voltage and phase current, it was the best instrument to use to implement the voltage integration test. As seen in Figure 4.2, a full voltage pulse is applied with no initial phase current. With each computation cycle, phase voltage and current variables are incremented by the amount sampled. As each desired current is reached, the difference between the voltage integral and resistive voltage drop integral are stored as flux in memory. When current reaches the final current value, i_{max}, the last flux value is computed and the asymmetric converter voltage is set to freewheeling mode rather than applying negative bus voltage. This allows for more accurate computations because the diode voltage is known more precisely than the DC voltage and because the current decays to zero at a slower rate during freewheeling, allowing higher sampling resolution. Current and voltage are integrated during freewheeling and resistance is then calculated for use at the next rotor angle. An initial pulse at the start of the algorithm measures resistance for use in the first flux measurement at the first angle. When the user rotates the rotor at a comfortable speed of about one measurement per second, resistance is invariant between pulses, so thermal drift, although negligible for the SRM motor under test, is well accounted for.

The microcontroller sample rate for this measurement was set as high as computation latency would allow, at 160kHz. This provides more sample points to integrate, filtering out noise. The bus voltage for this test was set at 60V, providing transients that were not too fast to measure accurately, but not so low as to increase pulse duration, causing significant resistive phase heating. Flux was calculated at 1 amp intervals up to 16 amps. Calculations during freewheeling assumed a diode voltage drop of .8 volts. Position feedback from the encoder enabled the entire measurement process to be automated by incrementing the angle command after every pulse and waiting for the user to turn the rotor to the next angle before applying the next pulse. Flux was calculated every two degrees for a total of 60 points, resulting in a 16X60 flux array which could be copied off the processor through Code Composer Studio into Matlab or Excel and converted to inductance by dividing each element of the array by its corresponding current. Since the magnetic characteristics are linear for low current (<2A), the inductance at i=0 is assumed to be the same as that at i=1. The automated method is quick, quiet, and introduces no switching noise into measurements.



a) aligned position b) partially aligned position c) unaligned position

Figure 4.3. Scope DC bus voltage(red) and phase current(blue) for the voltage integration test

4.2 Incremental Inductance Method

The incremental inductance measurement technique is mathematically similar to the voltage integration technique but delivers a fundamentally different voltage excitation. This method might also be called the small signal inductance method, or the hysteresis method. A digital hysteresis controller with hard switching ($\pm V_{DC}$) is used to establish a triangle wave with

a target peak-to-peak amplitude, Δi, about an average current, I. The current rise and fall times are measured for a sufficient number of triangle ripples so as to filter out measurement noise. If the current wave is sufficiently linear over the switching period, the data can be used to find incremental inductance measurements that are independent of phase resistance.

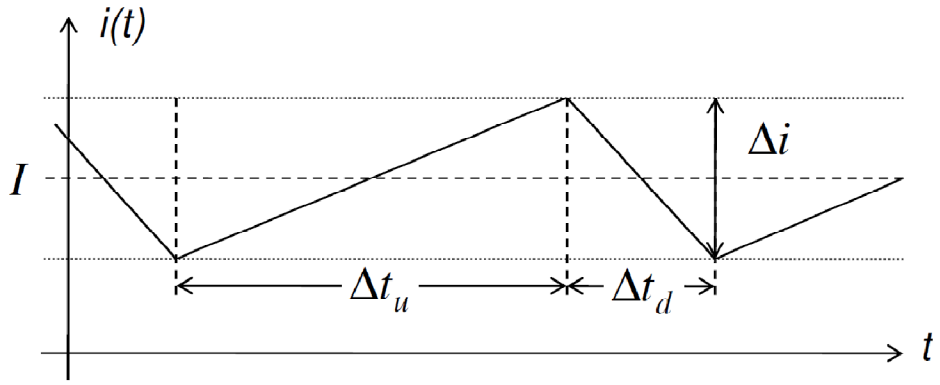


Figure 4.4 Incremental inductance measurement current wave

When the rotor is locked, equation (2.4) simplifies to:

$$\frac{\partial \lambda}{\partial i}\frac{\partial i}{\partial t} = v - Ri$$

The first partial, $\frac{\partial \lambda}{\partial i}$, is the incremental inductance. Note that incremental inductance is not the derivative of inductance with respect to current or time. Rather it is the ratio of the flux differential to the current differential. In comparison, inductance is the ratio of the flux to the current. If the partial derivative of current with respect to time is represented by finite terms, the equation can be reorganized to give:

$$\frac{\partial \lambda}{\partial i} = (v - Ri)\frac{\Delta t}{\Delta i} \tag{4.5}$$

This equation could be used to describe the flux during the rise and fall sections of the triangle wave in Figure 4.4 provided the inductive phase voltage, $v - Ri$, is approximately constant, such that the current ripple may be considered linear. The capacitor voltage, v, is constant over a short period. If the small ripple assumption, $\Delta i \ll I$, holds, then the resistive voltage drop, RI, is approximately constant and so is phase voltage. In addition, if I is very

small, then $v \gg Ri$, and the inductive phase voltage is still constant regardless of whether the ripple is small. However, since the incremental inductance varies with current, the current ripple should be small such that incremental inductance variance is negligible. The ripple magnitude should be at least as small as the increment between test currents. The linearity of the triangle waveform of Figure 4.6 demonstrates that the assumption of linearity is quite valid.

While the phase voltage is the positive bus voltage, the current change is positive:

$$\frac{\partial \lambda}{\partial i} = (V - RI)\frac{\Delta t_u}{\Delta i} \tag{4.6}$$

Alternatively when the phase voltage is the negative bus voltage, $-V$, the current is decreasing, so change in current is negative:

$$\frac{\partial \lambda}{\partial i} = (-V - RI)\frac{\Delta t_d}{-\Delta i} \tag{4.7}$$

Since the change in current over one period is zero, the change in flux is also zero, so the flux rise and fall must be equal and opposite. Equating the right sides of (4.6) and (4.7) gives:

$$(V - RI)\frac{\Delta t_u}{\Delta i} = (V + RI)\frac{\Delta t_d}{\Delta i}$$

Solving for resistance yields:

$$R = \frac{V}{I}\frac{(\Delta t_u - \Delta t_d)}{T}\ ,\ \ T = \Delta t_u + \Delta t_d \tag{4.8}$$

Similarly, adding (4.6) to (4.7) yields another expression:

$$2\frac{\partial \lambda}{\partial i} = (V - RI)\frac{\Delta t_u}{\Delta i} + (V + RI)\frac{\Delta t_d}{\Delta i}$$

Substituting (4.8) and rearranging gives:

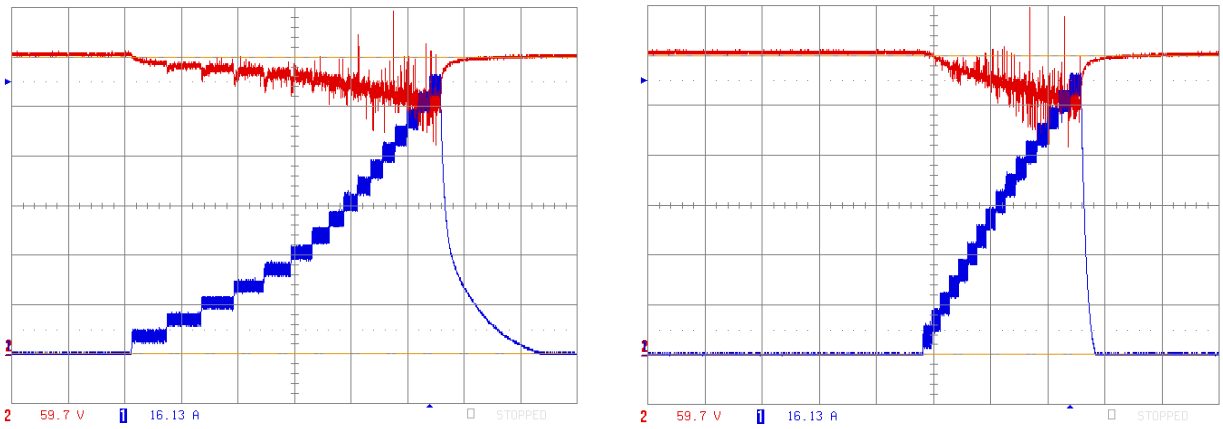
$$\frac{\partial \lambda}{\partial i} = \frac{VT}{2\Delta i}\left(1 - \left(\frac{\Delta t_u - \Delta t_d}{T}\right)^2\right) \tag{4.9}$$

Equation (4.8) is most accurate when I is large such that the resistive voltage drop is a significant fraction of the total phase voltage. It cannot estimate resistance well when current is small such that $\Delta t_u - \Delta t_d$ is small. In contrast, (4.9) provides a good estimate of incremental

inductance when the assumptions of current linearity discussed previously are satisfied. Although it is not shown for the sake of simplicity, each parameter in (4.9) corresponds to a particular current and angle. Integrating the expression with respect to current yields flux, from which inductance is derived as $L = \lambda/i$:

$$\lambda(\theta, i) = \int_0^i \frac{V(\theta, i)T(\theta, i)}{2\Delta i(\theta, i)}\left(1 - \left(\frac{\Delta t_u(\theta, i) - \Delta t_d(\theta, i)}{T(\theta, i)}\right)^2\right)di \tag{4.10}$$

Implementing the incremental inductance method is only slightly more complex than the voltage integration method. The converter and dsp were programmed to implement hysteresis control centered around an initial current command. The controller applied 32 current ripples for which rise times and fall times were averaged and stored, the current command was incremented, and another 32 current ripples were applied. This continued until the maximum current was reached, at which point the rotor angle was incremented and another series of ripple steps were applied. The operation is seen in Figure 4.5.



a) aligned inductance b) unaligned inductance

Figure 4.5. DC bus voltage and current waveforms for incremental inductance measurement

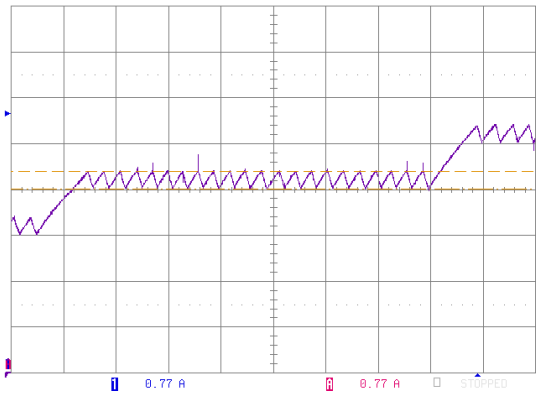


Figure 4.6. Close-up of current ripple with .77 amp ripple corresponding to a .75amp ripple command

It was found that the actual current ripple magnitude, Δi_{true}, was larger than the dsp measured ripple, Δi_{dsp}, especially at maximum ripple frequency at unaligned inductance. This is probably caused by attenuation from the current filter, and eddy current attenuation of higher frequency ripple, along with other unknown influences. This was corrected by experimentally finding the relationship between actual and dsp measured current ripple and using this to calculate the actual current ripple to use in the incremental inductance equation (4.9). The correction factors and corresponding exponential curve fit for the ratio of actual ripple to dsp ripple are shown in Figure 4.7.

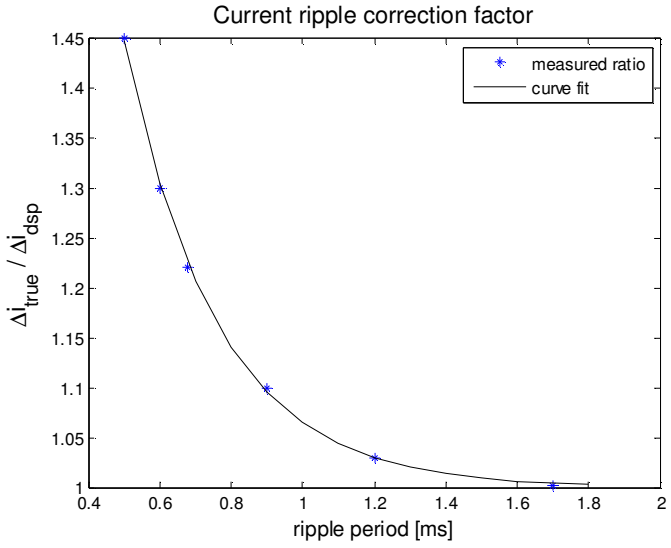


Figure 4.7. Ripple correction ratios and curve fit

The following curve fit represents the correction, with the ripple period, t_r, in milliseconds:

$$\frac{\Delta i_{true}}{\Delta i_{dsp}} = 1 + 3.1e^{-3.872 t_r}$$

4.3 Comparison of Inductance Characterization Results

A comparison of inductance measurements taken by the incremental inductance method and voltage integration method are shown in Figure 4.8.

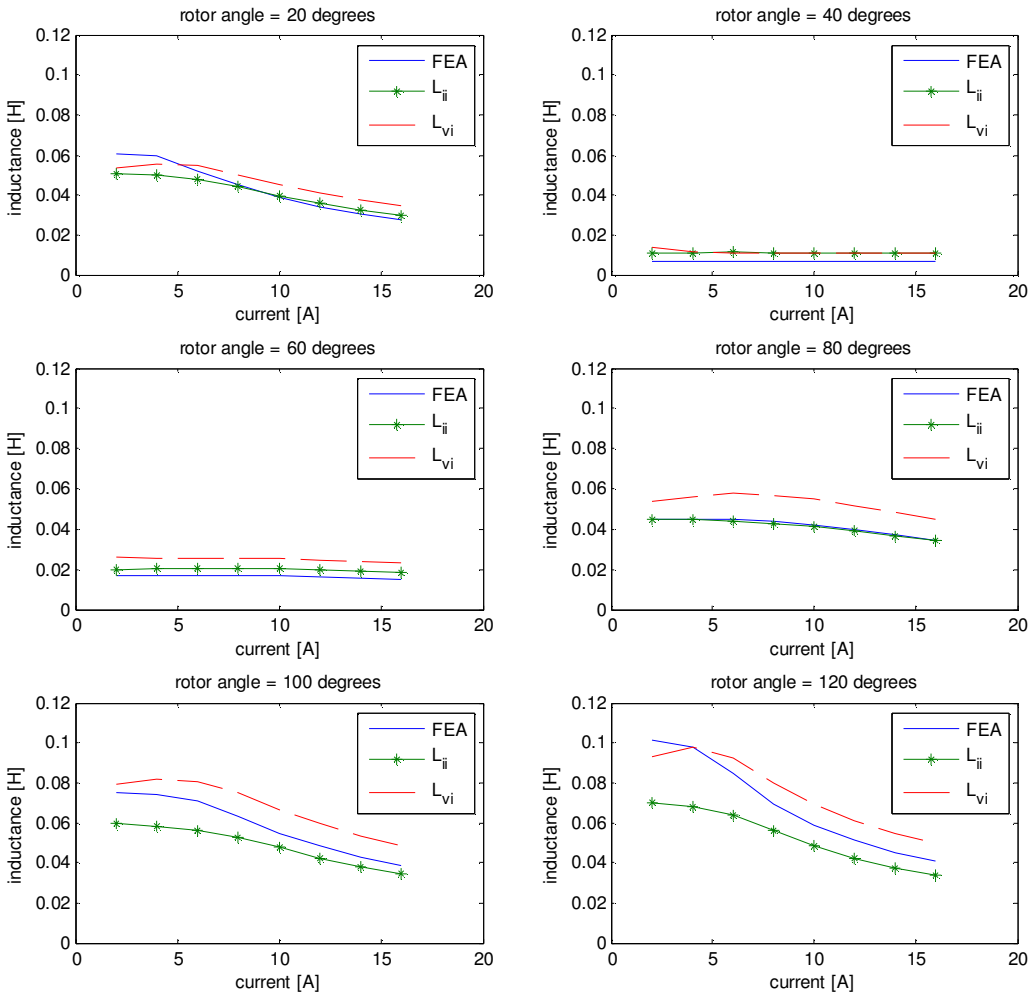


Figure 4.8. Comparison of inductance measurements taken by incremental inductance method, L_{ii}, and the voltage integration method, L_{vi}.

Although disagreement with FEA data is expected, both measurement techniques would be expected to yield the same inductance data if the true SRM electromagnetic model for this particular motor had the structure of (2.2). The discrepancies between data from the two experimental methods seen in Figure 4.8 appear to indicate a more complex model than assumed. Even so, the relative difference between data from all three methods does not exceed 30%. For aligned inductance measurements, FEA data falls roughly between the two measurement data sets, yet for unaligned inductance measurements, FEA is consistently lower than measured for both sets. This suggests that the unaligned inductance is significantly higher than expected, around 10mH rather than 6mH.

The inductance data results prompt the following thoughts:

- Each measurement method excites different system modes. The pulse current measurements from flux method may be influenced by eddy currents more than the small ripple measurements, while the small ripple measurement may yield a more accurate means of measuring inductance, especially at low currents. The voltage integration inductance data exhibits the characteristic roll off at low current that is seen in literature [6], approaching inductance values approximated by the incremental inductance method. This may suggest the presence of a transient corresponding to a change in the rate of current, or rather the current acceleration resulting from a change in phase voltage. Such a change in phase voltage would occur only at the begging of the flux integration pulse at low inductance, but continually during the incremental inductance pulse.
- Two measurement techniques resemble two states for a step current controller: the flux integration applies constant voltage causing current to rise as quickly as possible, as it would during the initial transient in a current step. This data might be considered more accurate for predicting large signal responses. The small ripple inductance method holds average current constant with a small current ripple imposed, just as would be expected for constant (or almost constant) current operation. Thus, this data might be considered more accurate for small signal responses. A controller might obtain the best transient and steady-state current results by using the inductance data taken from the corresponding state of system excitation. Data from both methods would be stored in memory, and during rising transients, data from the flux method would be used, and during steady current operation data from the small ripple method would be used.

- Rate of change of current is proportional to bus voltage – taking measurements at different bus voltages should lend insight into model frequency dependence. Eddy current losses are known to be proportional to the square of flux density and its time rate of change[20]. Magnetic hysteresis losses, both for large signals and small signals are also most likely a prominent factor.

More measurements are necessary to find a better inductance model but this is outside the scope of this thesis. FEA data is used for the deadbeat controller implementation since it generally falls between predictions from experimental methods.

5 Common Control Design Issues to Both PI and DB Algorithms

5.1 PWM Switching Scheme

The PWM switching scheme has significant bearing on system efficiency and current signal quality. As PWM frequency increases, current ripple decreases proportionally, but switching losses increase proportionally. Although sampling rate is limited by control algorithm latency, the PWM rate is not limited if multiple PWM pulses are applied per control period. More pulses per period would allow for a smoother application of voltage and a smoother current change during the period. In this case, current could be sampled without regard to the effects of inter-period ripple. However, this type of multi-rate control will not be considered because the switching losses incurred are unreasonably high, so controllers will be implemented using single-rate control. The symmetric pulse method will be used for both PI and deadbeat controllers in this thesis to mitigate the issue of inter-period ripple, while offering low switching losses.

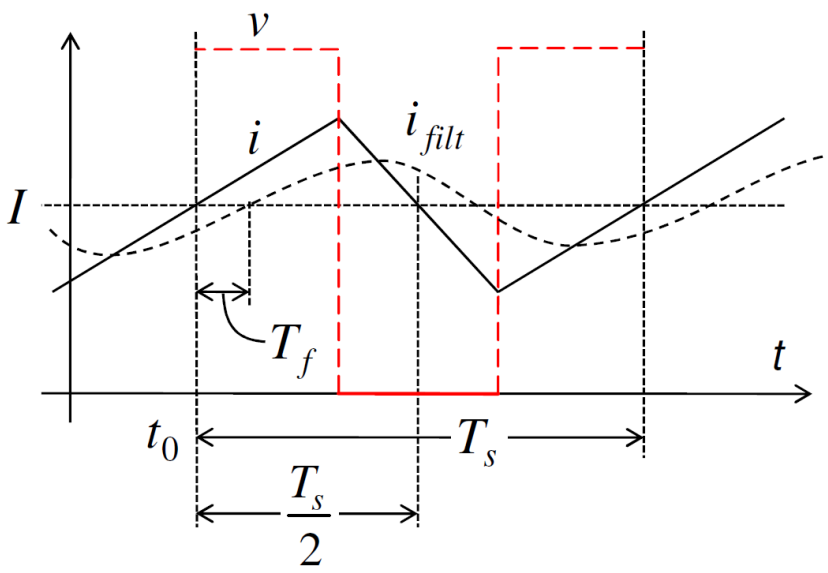


Figure 5.1. PWM symmetric voltage pulse scheme with current and filtered current.

The symmetric pulse PWM scheme centers a voltage pulse of either VDC(the dc link capacitor voltage) or zero at the middle of a switching period, at $T_s/2$. Sampling occurs at the beginning of the PWM period and optionally in the middle. If a zero pulse is used, then the duty cycle of the pulse is 1-d and the default state is VDC rather than zero. The zero pulse method is

used in this thesis with sampling occurring once at the beginning of the period, as shown in Figure 5.1.

This symmetric pulse PWM scheme has a few advantages over asymmetric PWM. Voltage transients of up to 2V in the control hardware, or 60% of the entire A2D dynamic input range, occur at each switching instant due to crosstalk between the power and signal electronics stages. The transients are very short, around 2us, so they are relatively harmless as long as sampling and switching never coincide. The symmetric zero voltage pulse scheme centers all switching around the middle of the PWM period, so switching cannot occur during sampling unless a duty cycle of zero is applied.

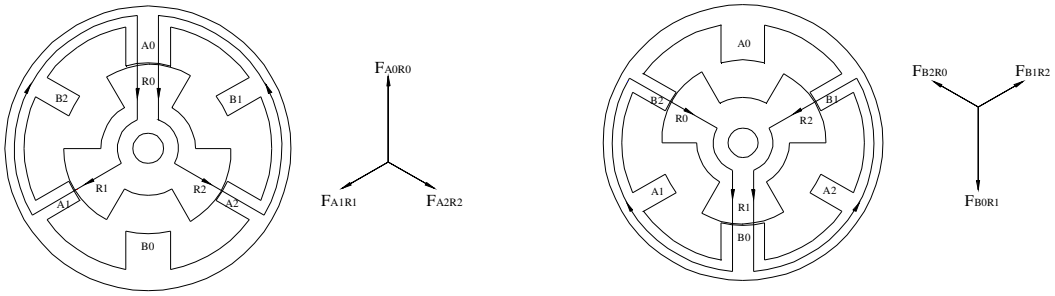
Another important benefit from centered switching is that sampling occurs at instants during which the instantaneous current is equal to the average current. This is especially advantageous if current ripple is very large or if a high bandwidth controller is used. If there is no current feedback filter, the optimal time to sample is at the beginning of the PWM period or in the middle. If the PWM duty cycle is very close to zero or 1, the user has the option of choosing to sample at the instant that is farthest from the switching instant. To do this, however, computation latency must be limited to less than one half the PWM period.

As seen by the filtered current wave in Figure 5.1, if there is a current feedback filter, a simple form of filter compensation may be implemented by delaying the sampling time by T_f, the effective filter delay occurring at the switching frequency. The next chapter on linear analysis shows that if the filter is first order and the cutoff frequency is high enough, the samples of the phase current will be approximately equal to delayed samples of the filtered current, if the delay is chosen according to the filter cutoff frequency. In the general case, the delay between filtered current and actual current is not constant. However, this compensation technique is very attractive for two reasons: 1) it is very simple to implement and incurs no extra computational complexity from the controller if the PWM peripherals allow phase shifting as they do in the TI C2000 MCUs. 2) While current estimation may be applied by passing the feedback current through the inverse of the known filter model, this implies lead compensation, which will amplify noise. Thus, linear filter compensation methods are undesirable because they negate the function of the current filter, which is to reduce feedback signal noise. On the downside, increasing sampling delay decreases the time available for controller calculations, because the

PWM command for the next PWM period must be delivered before the end of the current period. Simulation of current filter delay compensation is presented in section 8.2.2.

5.2 Commutation Control

In order to apply positive torque for all rotor angles, the machine is designed such that the rate of inductance increase in the forward rotating direction is smaller than the rate of decrease in inductance (the rate of increase in the reverse motoring direction). Thus, it is possible to apply positive torque over a larger angular displacement than negative torque, resulting in overlap between the range of motoring operation of each phase. This enables smoother operation with lower torque ripple in the forward motoring direction and ensures that the motor can be started from any initial rotor angle. On the other hand, braking and reverse motoring operation is restricted to a smaller angular range, so braking performance will be low and reverse motoring may be impossible without initially positioning the rotor angle within the region of increasing inductance. The machine is not well suited for smooth four-quadrant operation. As commutation is not the central focus of this thesis, it will be concerned with first quadrant operation only.



(a) Phase-A windings are excited (b) Phase-B windings are excited

Figure 5.2. Flux path and normal force diagram when poles are aligned (Adapted from [11])

Figure 5.2 illustrates the aligned rotor positions for phase A and B. It is of interest to note that this motor has been designed to minimize radial forces to reduce vibration, noise and bearing deterioration. Motor noise has been one of the largest detractors of SRM technology.

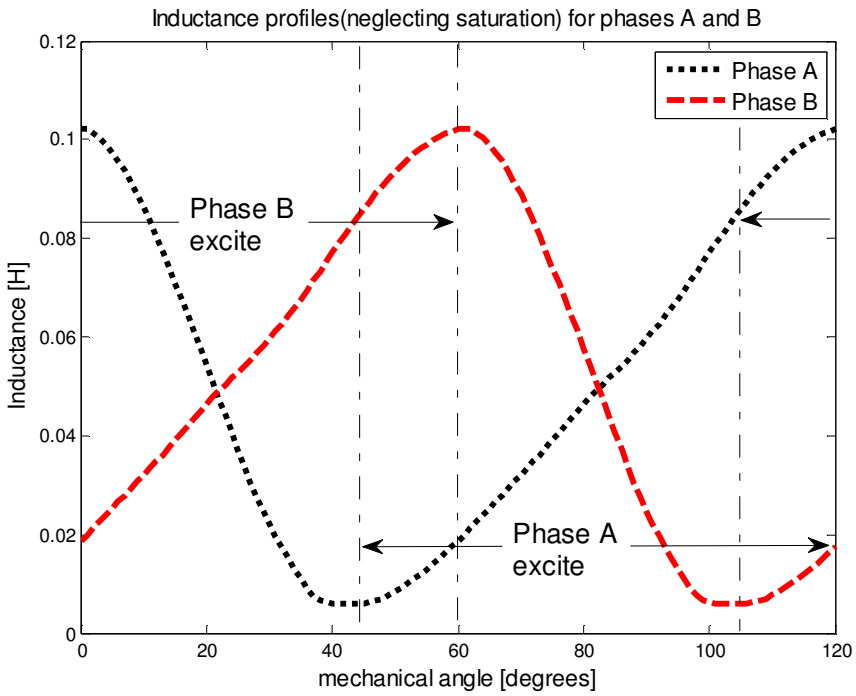


Figure 5.3. Inductance profiles for both each phase with i=2

Figure 5.3 shows the inductance profile for phase A and B for a current of 2A. The profile for phase B is simply phase A shifted by 60 degrees. To ensure positive torque only, commutation and excitation angles are often chosen such that phase A current is zero outside of "Phase A excite" and phase B current is zero outside of "Phase B excite". Motor operating speed, aligned and unaligned inductance values, and bus voltage are prominent factors influencing the choice of commutation and excitation angles. For this thesis, however, they are chosen experimentally to yield a low current command for nominal and half speed operation. The excitation and commutation angles used for all PI and DB control algorithms in this thesis are 44° and 92° respectively.

5.3 Speed Control

The speed controller for both PI and DB current control algorithms uses a PI speed controller with simple anti-windup. When the speed error falls below the error limit, ω_L, the controller integrates error. The error limit ω_L = 200rpm was chosen experimentally to provide

low overshoot and to reduce the dynamic range of the integrator state to prevent fixed point overflow in the dsp.

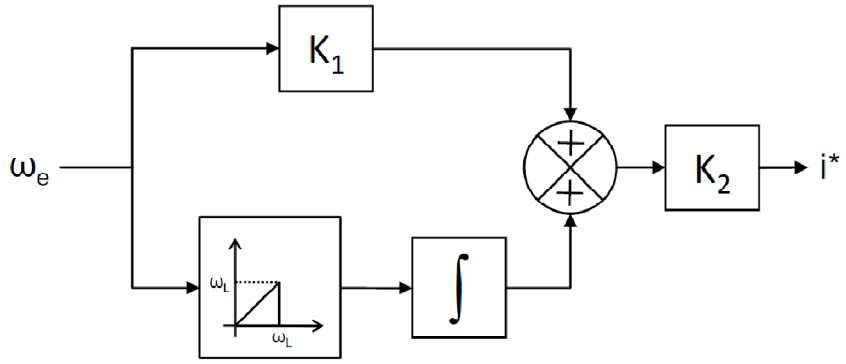


Figure 5.4. Speed control loop with anti-windup type integration

6 Linear Control Design

6.1 SRM Linearization

Linearization is used to obtain a fundamental understanding of the basic electrical dynamics from a stability standpoint. It allows us to investigate the effects of the feedback filter and propagation delay. First order low-pass transfer functions will be assumed for the plant, filter and delay. Linearized equations are derived as follows.

From (2.5) the electromagnetic equation describing the SRM is:

$$v = \left(L(\theta, i) + i\frac{dL(\theta, i)}{di}\right)\frac{di}{dt} + i\left(\frac{dL(\theta, i)}{d\theta}\omega + R\right) \tag{6.1}$$

It is desired to obtain a linearized approximation to this equation for some steady-state rotor speed and current command, ω_0 and i_0. Assume that the speed ripple at steady-state is negligible and so speed is regarded as constant. A nominal steady-state voltage, v_0, is associated with the steady-state current command. The effects of saturation may be considered negligible for nominal operation, so the derivative of inductance with respect to current is zero. Since inductance varies with position, and position is coupled to mechanical dynamics which will not be considered, the steady-state inductance may be taken as its average over one electrical rotation[1]:

$$L_0 = \frac{L_a + L_u}{2}$$

where $L_a$is the maximum inductance at the aligned position and L_u is the minimum inductance at the unaligned position. The steady-state derivative of inductance is nearly constant in the absence of saturation, as seen in Figure 5.3, and is calculated by:

$$\frac{dL_0}{d\theta} = \frac{L_a - L_u}{\theta_a - \theta_u}$$

with $\theta_a$and $\theta_u$corresponding to the angles at aligned and unaligned inductances respectively.

Making the preceding substitutions gives:

$$v_0 + \tilde{v} = (i_0 + \tilde{\imath})\left(R + \frac{dL_0}{d\theta}\omega_0\right) + L_0\frac{d(\tilde{\imath} + i_0)}{dt}$$

Conveniently, the equation is already linear, so there is no need to linearize with respect to a steady state operating point. The linear equation is then,

$$v = i\left(R + \frac{dL_0}{d\theta}\omega_0\right) + L_0\frac{di}{dt} \tag{6.2}$$

Thus, the steady-state plant is effectively an RL circuit with the following transfer function:

$$P(s) = \frac{i}{v} = \frac{1/R_{eq}}{1 + \frac{s}{\omega_p}}$$

with

$$R_{eq} = R + \frac{dL_0}{d\theta}\omega_0 \text{ and } \omega_p = \frac{R_{eq}}{L_0}$$

The equivalent resistance, R_{eq}, is the sum of the phase resistance and a term corresponding to back emf.

6.2 Current Feedback Filter

The presence of excessive noise in the current feedback circuitry has made an analog current feedback filter necessary. This is modeled by a first order low-pass filter with cutoff frequency, ω_f (rad/s):

$$F(s) = \frac{1}{1 + \frac{s}{\omega_f}}$$

The filter frequency is calculated experimentally by measuring the delay between the filter output and a current ramp signal. The time domain equation of the above current filter is:

$$\frac{di_{filt}}{dt} = \left(i - i_{filt}\right)\omega_f \tag{6.3}$$

where i_{filt} is the output of the current filter. Since a first order filter has a constant steady-state error to a ramp, the filter signal will be parallel to the current signal and thus has the same slope. In the case of a constant ramp current, the filter derivative will be constant, so an equivalent discrete time equation can be written as:

$$\frac{i_{filt}(t+\Delta t)-i_{filt}(t)}{\Delta t}=\big(i(t)-i_{filt}(t)\big)\omega_f$$

If the time step is taken such that $i_{filt}(t+\Delta t)=i(t)$, or simply to be the delay required for the filter to reach the current, then the filter cutoff frequency can be calculated very simply and accurately as:

$$\frac{1}{\Delta t}=\omega_f \tag{6.4}$$

6.3 PI Control Design

The steady-state plant parameters for the motor under test and the experimentally determined current filter cutoff frequency are given in Table 6.1.

Table 6.1 Linearized Plant Parameters

L_a	.103H
L_u	.006H
$\theta_a-\theta_u$	~π/3
R	1.2Ω
$dL_0/d\theta$	.09262
L_0	.0545H
ω_f	31400 rad/s (5000hz)

Table 6.2. Plant parameters for three rotor speeds:

	500rpm	1800rpm	3600rpm
ω_0	52.36 rad/s	188.5 rad/s	377.0 rad/s
R_{eq}	6.05 Ω	18.7 Ω	36.1Ω
ω_p	111 rad/s	343.1 rad/s	662 rad/s

Note: The inductance curve is actually increasing over an angular displacement of about 70 degrees, not π/3, or 60 degrees. However, the average slope over the range is greater than the slope that would be calculated by using the aligned and unaligned inductance angles. The tangent line to the average slope intersects angles at aligned and unaligned inductances with a difference of roughly π/3, so 60 degrees is used. With this knowledge of the plant and feedback frequency characteristics, we may design a PI controller that yields the best results.

The linear block diagram of the system for one phase is:

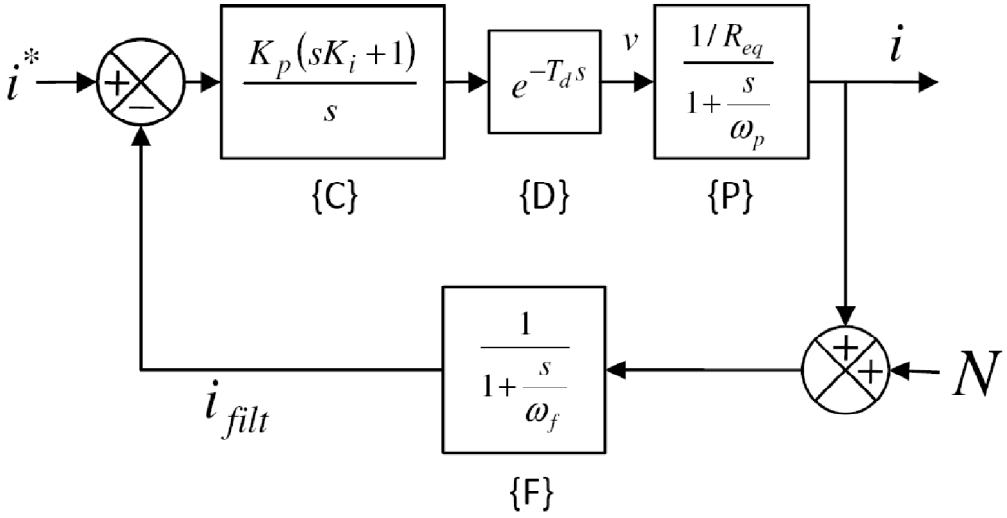


Figure 6.1. Frequency domain diagram of current control system with linearized motor

To accurately model the digital PI controller, delay must be considered in the loop model. This is shown by the exponential, {D}, following the controller, {C}, in Figure 6.1. The PI controller sampling time is f_c, and the sampling period is T_c. Latency is introduced by one cycle of computation time required to compute the control signal for the next sampling instant. Delay may also be attributed to the PWM, and the exact delay depends on the modulation scheme (asymmetric vs. centered pulse). For the purposes of this analysis, and later deadbeat analysis, we will assume a total controller delay of one cycle: $T_d = T_c$. The linearized SRM plant is denoted by {P} and the current feedback filter is {F}.

The PI controller gain, K_p, is chosen to provide fast transient current response to a step and the gain, K_i, is the time constant of the controller zero. The controller integrator drives

steady-state error to zero but is not necessary if zero steady-state error is not important, as might be the case if an outer speed loop compensates for inner current loop steady-state error. In such a case, a simple proportional controller, K_p, is sufficient.

In the case of a PI controller, K_i is chosen as the inverse of the plant cutoff frequency such that the controller zero and plant pole cancel:

$$K_i = 1/\omega_p \tag{6.5}$$

The forward loop gain, CP, is then simply K_p/sR_{eq}. The PI controller will be designed to ensure a phase margin (PM) sufficient to provide a good balance between overshoot and transient performance. The integrator contributes -90° of phase at all frequencies. For a loop phase lag of PM, the sum of filter and delay phase lag at crossover must be 90° - PM. The delay contributes $\omega_0 T_d$ radians of phase lag. Therefore, we choose the loop crossover frequency, ω_{cr}, to satisfy the following:

$$PM = 180 - 90° - \text{atan}\left(\frac{\omega_{cr}}{\omega_f}\right) - \omega_{cr} T_d \frac{360}{2\pi}$$

The controller gain, K_p, is chosen to set the loop magnitude to one at crossover frequency:

$$|CDPF| = \frac{K_p}{\omega R_{eq}} \frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_f}\right)^2}}$$

$$|CDPF|_{\omega=\omega_{cr}} = \frac{K_p}{\omega_{cr} R_{eq}} \frac{1}{\sqrt{1 + \left(\frac{\omega_{cr}}{\omega_f}\right)^2}} = 1$$

$$K_p = \omega_{cr} R_{eq} \sqrt{1 + \left(\frac{\omega_{cr}}{\omega_f}\right)^2} \tag{6.6}$$

PI gains and corresponding loop gain crossover frequencies for two operating speeds are shown in Table 6.3 below. The crossover frequency, which may be considered the closed loop bandwidth, is 1374Hz. In comparison, the fundamental phase excitation frequency for a two phase SRM with P_r poles per phase is $f_{ph} = P_r \omega / 2\pi$. At 3600rpm nominal speed the two-pole-

per-phase 6/3 SRM has a phase frequency of 180Hz. This is considered the minimum required bandwidth for a current controller.

Notice, since the pole, K_i, cancels the motor pole, which is dependent on operating speed, the loop gain crossover is independent of plant dynamics. Thus, transient performance for PI control is dependent on the current filter pole and controller delay alone. The relative influence of each of these is seen by a comparison of closed loop current responses to a unit step command, in Figure 6.2. The plot in parts a) and b) use different PI controllers designed to set PM = 65° for loops with 1 and 2 delay cycles respectively.

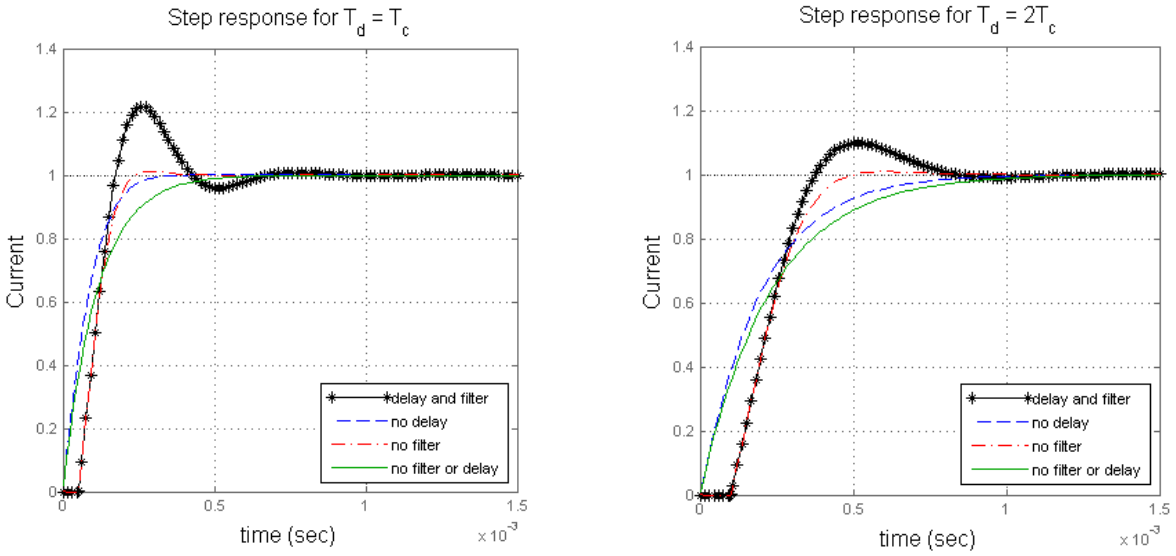
Table 6.3. PI gains and crossover frequency for three operating speeds for $f_c = 20kHz$ and PM = 65° with one cycle latency:

Nominal inductance estimate	Parameter	1800rpm	3600rpm
$L_0 = \frac{L_a + L_u}{2}$	K_i	.0029	.0015
	K_p	167000	323000
$L_0 = L_u$	K_i	.00032	.00017
	K_p	167000	323000
$L_0 = anything$	ω_{cr}	8633rad/s (1374Hz)	

It should be noted that to choose PI gains to optimize a phase excitation current step, the unaligned inductance should be used for nominal inductance rather than average inductance because the current step transient occurs almost entirely within the unaligned region. The final PI parameters for simulation in Table 6.4 were chosen using the unaligned inductance in Table 6.3 above, with the exception that the value for K_p has been halved for each case to reduce overshoot in simulation.

Table 6.4. PI control values used in simulation

1800rpm:	$K_i = .00032, K_p = 80000$
3600rpm:	$K_i = .00017, K_p = 160000$



a) Step response for 1 cycle delay

b) Step response for 2 cycle delay

Figure 6.2. Step Responses for linear system with pure time delay demonstrating effects of filter and processor delay.

Some performance can be gained by using a current estimator to predict phase current from the filtered current. However, in a linear analysis, the estimator cancels the filter plant, which amplifies filter output noise to the level of the phase current noise, rendering the current filter useless. If a continuous linear system model is assumed, then in fact current estimation is pointless. However, implemented current estimation would involve cascading an analog current filter with a digital current estimator with sample and hold input. Even if the current estimator is designed as the inverse of the current filter, the transfer function of the two in cascade is *not unity*. Phase current estimation, i.e. filter compensation, will be considered more in section 8.2.2.

6.4 Noise Analysis

The closed loop transfer function of the system, T, assuming a linear representation of D such as by a Pade approximation, is:

$$T = \frac{i}{i^*} = \frac{CDP}{1 + CDPF}$$

Most signal noise is injected before the current filter, as indicated in Figure 6.1. The transfer function, D, relates phase current to feedback noise disturbance and is the negative of the product of the closed loop transfer function and the filter transfer function:

$$D = \frac{i}{N} = \frac{-CDPF}{1 + CDPF} = -TF$$

Since the transfer function bandwidth is roughly 25% of the filter bandwidth, the magnitude of D is approximately equal to T with the exception of a slightly lower bandwidth and an additional high frequency pole. The frequency plot of D indicates that, for frequencies below the closed loop bandwidth, the system is as sensitive to feedback noise as it is to the current command. The PI controller can be expected to deliver good disturbance rejection if the majority of the noise disturbance to be rejected is higher than the closed loop bandwidth. The open loop transfer function with and without delay is shown in Figure 6.3. A graphical comparison of D, T and T without delay are shown in Figure 6.4.

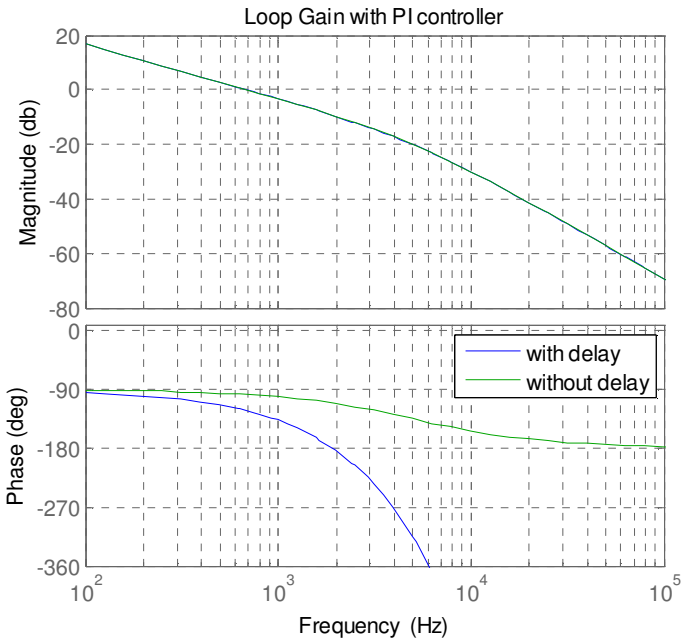


Figure 6.3. Comparison of loop gain with and without latency delay.

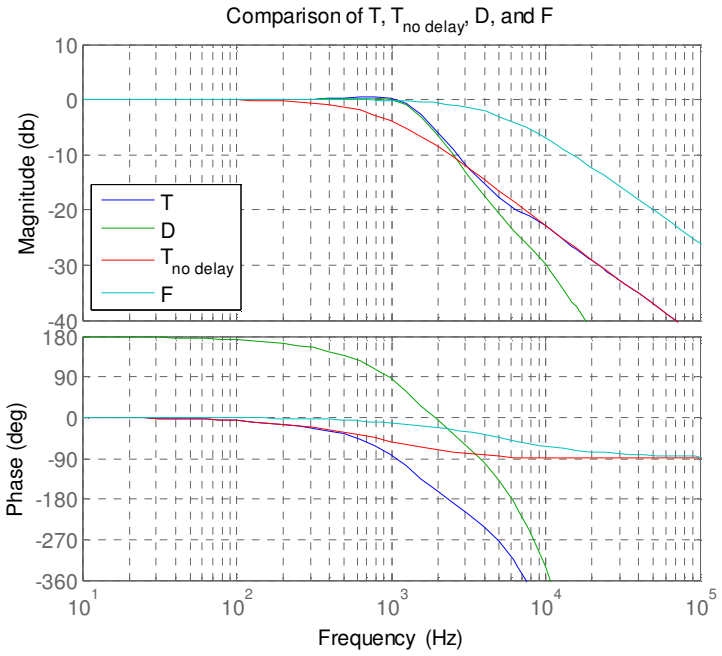


Figure 6.4. Bode plots for closed loop system with and without latency delay compared with the noise disturbance

Unfiltered current feedback noise was characterized by direct measurement of its power spectral density (PSD) on an oscilloscope. The PSD-spectrum is represented by the red waveform, and is a function of frequency starting at DC and extending linearly to 1.25MHz. The PSD reaches the noise floor at approximately 800Khz. From this PSD, almost all the noise appears to occur from 12Khz to 700Khz. Thus the 5Khz filter pole is well justified as it ensures a noise disturbance cutoff frequency around the 1.2Khz closed loop bandwidth, a decade below the bulk of the noise.

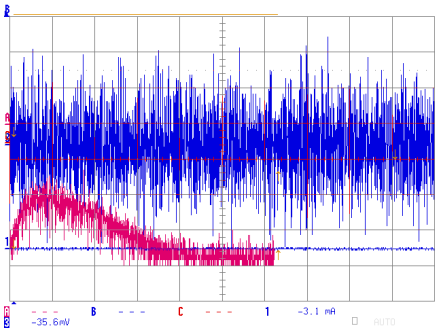


Figure 6.5. Noise signal and corresponding PSD from DSO screen.

6.5 PI Current Control Antiwindup

This linear analysis has not considered the effects of nonlinear actuator control saturation, which tends to degrade transient response in implementation. If the input voltage saturates, the control law assumes that the control is higher than it actually is, resulting in overcompensation. To counteract this, integrator antiwindup was used in PI current controllers in simulation and implementation. Integration occurs only when the magnitude of the control is less than its limit, v_{sat}. This may be understood by the following PI control diagram:

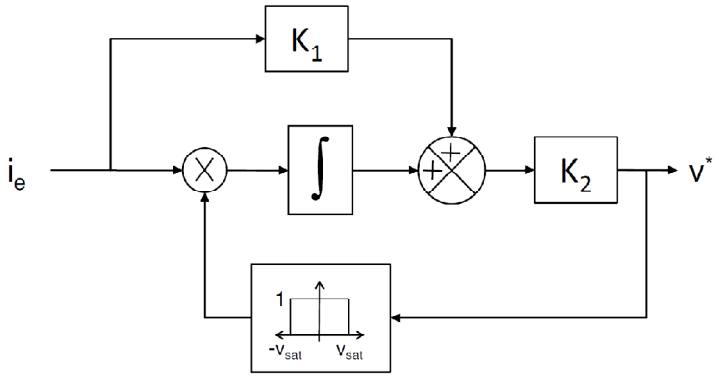


Figure 6.6. Antiwindup in the PI current controller

7 Deadbeat Current Control

7.1 Introduction

Deadbeat control is defined as essentially any digital control algorithm that achieves and maintains zero error in a minimal number of control cycles. Thus it satisfies, $y(kT_s) = 0$ for some k greater than an integer, n. Deadbeat control implies nothing about the output between samples, so it does not necessarily preclude inter-sample ripple. Deadbeat control is a mathematical limit, impossible to achieve in implementation, so a controller is considered deadbeat if its response is close to deadbeat. In practice, a control is considered deadbeat if it drives error to within 1 or 2 percent of zero in a few control periods with minimal overshoot (less than 5%). This presents a considerable challenge, since the convention in digital control is to sample at 20 to 40 times the required bandwidth. Thus, deadbeat control is best applied by meticulous time-domain design that considers time delay and inter-sample dynamics.

The linear deadbeat control problem was originally based on plant inversion(i.e. pole-zero cancellation), provided the plant is minimum phase and other requirements, such as those of Ragazzini's method, are satisfied [21]. In recent years, deadbeat control has been expanded to general controllable systems [22]. However, there is little research in the area of nonlinear deadbeat control. Furthermore, this thesis is not concerned with investigating deadbeat theory so much as it is concerned with exploring the usefulness of the deadbeat algorithm for SRM current control.

The electrical system of interest has one state, current, which may also be expressed as flux or inductance. Rotor speed and position comprise the two mechanical dynamic states but because these have large time constants they may be measured directly, treated as independent time-varying inputs to the system. Therefore the system effectively has one state and so current equations of the SRM involve a nonlinear system of relative degree one, meaning there is one integration from control input to state. Further, the plant linearization at any operating point is minimum phase. These two qualities reduce the theoretical complexity of the control problem.

As suggested by PI control analysis, a linear controller has performance limitations. Perhaps the two most dominant control hurdles in SRM current control are compensation of the digital controller latency and compensation for the time-varying inductance. These are both

nonlinear characteristics. Although linear latency compensation is possible, it would require the added complexity of at least two more states by a second order Pade approximation. However, a nonlinear deadbeat controller is discrete and designed in the time domain, so it is well suited for delay compensation.

Deadbeat control attempts to mitigate these and the following SRM control challenges:

1. Computation and PWM delay. A predictive deadbeat algorithm attempts to correct this.
2. Nonlinear inductance equations, which cause bad current tracking for PID control methods. An experimentally measured 2D inductance profile is stored in a lookup table.

On the other hand, deadbeat control comes with its own set of challenges that must be considered:

1. Sensitivity to plant parameters, effect on robustness and stability.
2. Computationally more complex.
3. No effective integral action for good steady-state tracking to compensate for parameter or model mismatch.

The deadbeat controller will be simulated and implemented and then its performance will be assessed in terms of transient performance and the above concerns. In chapter 12, adaptive modifications will be proposed to solve challenges 1 and 3.

7.2 Integration Techniques for SRM Deadbeat Current Control Design

The deadbeat control law is taken from the recursive discrete equations for the electromechanical system. Since deadbeat control is concerned with driving a variable to zero in just a few control periods, it is very important to give close attention to the timing of the control algorithm. If a system's transients are much longer than the time step used by an iterative solution, we may be assured that even simple numerical methods will generate an accurate solution. As the time step approaches zero, a system's response between iterations approaches linearity and the iterative solution approaches the true solution. However, since deadbeat control by definition involves transients lasting only a few iterations, all variables except the capacitor

voltage, v_C, the rotor velocity, ω, and the phase resistance, R, can undergo large nonlinear changes per iteration. Rather than a continuous control problem, or a discrete control problem, deadbeat control may be considered a piecewise-continuous control problem. Thus, we find that the simplest numerical methods are inadequate for providing the recursive equations for deadbeat control. Before finding a discrete approximation for this equation, it may be helpful to briefly review numerical integration.

Given an ordinary differential equation,

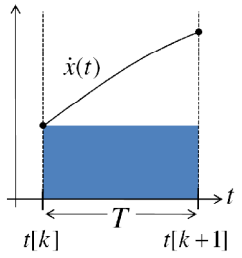
$$\dot{x} = f(x(t), t)$$

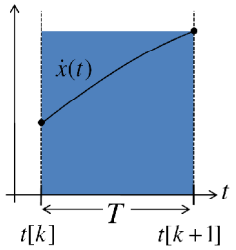
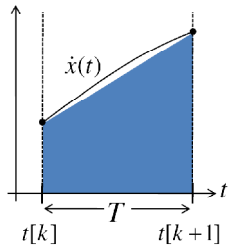
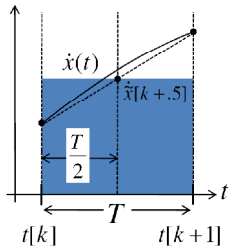
a future state can be solved exactly by the second fundamental theorem of calculus:

$$x(t_0 + T) = x(t_0) + \int_{t_0}^{t_0+T} \dot{x}\, dt \tag{7.1}$$

where T is the period of one numerical iteration. Most numerical methods originate with this equation. The accuracy and complexity of a solution depends on the numerical integration technique used to approximate the integral on the left side of (7.1). The simplest iterative approximation is the first order Euler approximation, or rectangular approximation, which estimates the integral by assuming the integrand is constant between samples. The value of the integrand during a sample may be taken to be the sampled value at the beginning of the period or at the end. These two variations are called the forward rectangular approximation and backward rectangular approximation respectively. In general, either approach yields similar error.

Table 7.1. Comparison of numerical integration methods.

Forward rectangular approximation: $$x[k+1] \cong x[k] + \dot{x}[k]T$$ or $$\dot{x}[k] \cong \frac{x[k+1] - x[k]}{T}$$	

Backward rectangular approximation: $$x[k+1] \cong x[k] + \dot{x}[k+1]T$$ or: $$\dot{x}[k+1] \cong \frac{x[k+1] - x[k]}{T}$$	
Trapezoidal approximation: $$\frac{\dot{x}[k] + \dot{x}[k+1]}{2} \cong \frac{x[k+1] - x[k]}{T}$$	
Midpoint(modified) approximation: $$\dot{\tilde{x}}[k+.5] \cong \frac{x[k+1] - x[k]}{T}$$ where $$\dot{\tilde{x}}[k+.5] = \dot{x}(x[k+.5])\big\vert_{x[k+.5] \cong \frac{x[k]+x[k+1]}{2}}$$	

Applied over many time steps, the two Euler (rectangular) integrations yield global error proportional to T, while the error by the trapezoid or midpoint methods are proportional to T^2 [23]. First order forward or backward Euler approximations are inadequate to predict dynamics of systems with large signal variation within a calculation interval, so we consider the second order trapezoid or midpoint rule. More accurate higher order discrete solutions would be possible by applying a higher order solution, such as the Runge-Kutta method. However, such higher order methods are designed for ordinary differential equations, whereas the motor model is technically a PDE. Furthermore, the relatively simple differential equation system model used for deadbeat control has a first order system linearization that will not benefit much more from higher order methods. Since high speed real-time control is desired, higher order methods are very undesirable because performance gains become diminishingly small for a much larger computational expense.

The midpoint method assumes that the average of the derivatives at the start and end of the period are approximately equal to the derivative at the middle of the period. Thus, it requires only one computation of the derivative, while the trapezoidal method requires two. The modified midpoint method further assumes that the states at the middle of the period are the average of the states at the ends of the period. The modified midpoint method is unnamed but employed by Springbob and Holtz in [10] for the case of a deadbeat current controller for a PM machine.

To be more precise, if the system, $\dot{x} = f\big(x(t)\big)$, is LTI with output f and input x, then the discrete differential, $\frac{x[k+1]-x[k]}{T}$, is approximated as an average of the outputs by the trapezoidal method and the output of the average of the inputs by the modified midpoint method. By definition of linearity, then, both approximations give identical equations if f is linear. For a nonlinear system, the modified midpoint method yields a simpler equation because it invokes one instance of $f(x(t),t)$, while the trapezoidal method invokes two that cannot necessarily be simplified. So, in the linear case:

$$\frac{\dot{x}[k]+\dot{x}[k+1]}{2} = \frac{f(x[k])+f(x[k+1])}{2} = f\left(\frac{x[k]+x[k+1]}{2}\right) = \dot{x}\left(\frac{x[k]+x[k+1]}{2}\right)$$

Note that the backward rectangular method is usually not desirable over the forward rectangular methods because a future state appears on the right hand side of the equation. As can be seen in the trapezoid and midpoint methods, a future state always appears on the right hand side of the equation. To deal with this, a reasonable estimate for a future state is calculated from present states. This will be referred to as the predictive aspect of the controller. This is used not only to derive a causal equation, but also to handle delay, as will be seen later. In general, if the discrete equation contains states that occur multiple periods in the future, those states are replaced by approximations in terms of prior states until the right side of the equation can be expressed in terms of present or past state samples only. It is helpful to note that applying the trapezoidal approximation to a continuous linear system is equivalent to applying the Tustin (bilinear) substitution in order to convert a system in the s domain to the z domain [21]. The substitution is as follows:

$$H_T(z) \cong H(s)\big|_{s=\frac{2z-1}{Tz+1}}$$

In order to investigate the benefits of the modified midpoint approximation and predictive deadbeat control, three control laws will be proposed and simulated. The first will be derived

from Euler's approximation, the second will be derived from the modified midpoint approximation, and the third will be derived from the modified midpoint method with a prediction to account for latency delay.

7.3 Control Law A

The electromechanical system dynamics are given by (2.3):

$$i\frac{dL}{dt} + L\frac{di}{dt} = v - Ri$$

Applying the forward Euler approximation and solving for the voltage command, the control input, gives:

$$v[k] \cong i[k]\left(R + \frac{L[k+1] - L[k]}{T}\right) + L[k]\frac{i[k+1] - i[k]}{T}$$

Inductance will be replaced with its effective reactance, $X = L/T$, determined by the control frequency. Assuming a deadbeat response in one time period, implying zero error at k+1, the future current, $i[k+1]$, is equal to current command $i^*[k+1]$. The voltage, $v[k]$, becomes the voltage command, $v^*[k]$, to be computed. The voltage command is applied by a PWM pulse from instant k to k+1, and is the average voltage of the pulse over the period. The future inductance, $L[k+1]$, is computed by lookup table interpolation with inputs being the future current command and rotor position determined by linear extrapolation of velocity.

$$v^*[k] = i[k](R + X[k+1] - 2X[k]) + i^*[k+1]X[k] \tag{7.2}$$

and

$$X[k+1] = X(i^*[k+1], \theta + \omega T) \tag{7.3}$$

For simplicity, the notion of reactance will be used to replace the ratio of inductance per period. The assumption of (7.3) will be used for all deadbeat control laws unless otherwise noted. From the voltage command, the duty cycle is chosen assuming the DC capacitor voltage is constant over the control period:

$$d^*[k] = \frac{v^*[k]}{v_C[k]} \tag{7.4}$$

7.4 Control Law B

Now, the modified midpoint approximation can be applied by augmenting control law A as follows:

$$v^*[k+.5] = i[k+.5]\left(R + \frac{L[k+1]-L[k]}{T}\right) + L[k+.5]\frac{i[k+1]-i[k]}{T}$$

where,

$i[k+.5] = \frac{i[k]+i^*[k+1]}{2}$ and $L[k+.5] = \frac{L[k]+L[k+1]}{2}$

As in control law A, the derivatives of (2.3) are computed over the interval between samples k and k+1. However, the corresponding state is now taken to be the average of the state samples at k and k+1 instead of at k. The digital current controller is implemented to apply the voltage command at sampling instants only, so it is constant between samples. Thus, the average pwm voltage, $v[k+.5]$, is equal to $v[k]$. Applying these substitutions and the notion of reactance and simplifying the above expression gives the discrete equation:

$$v[k] \cong i[k](R/2 - X[k]) + i[k+1](R/2 + X[k+1]) \tag{7.5}$$

Thus the voltage command is a function of the present current and the current command for the next period:

$$v^*[k] = i[k](R/2 - X[k]) + i^*[k+1](R/2 + X[k+1]) \tag{7.6}$$

The final duty cycle is calculated as in control law A.

7.5 Control Law C

Control law B is based on the unrealistic assumption that the voltage command, $v^*[k+1]$, may be applied immediately after sampling currents at instant k. Due to significant processor computation latency, this is impossible. In practice, computation time is often greater than half the control period (or should be for good CPU usage), so the voltage command cannot be applied

until the beginning of the next cycle, at k+1. This delay of one period must be compensated by the control law. The signal timing is shown in Figure 7.1.

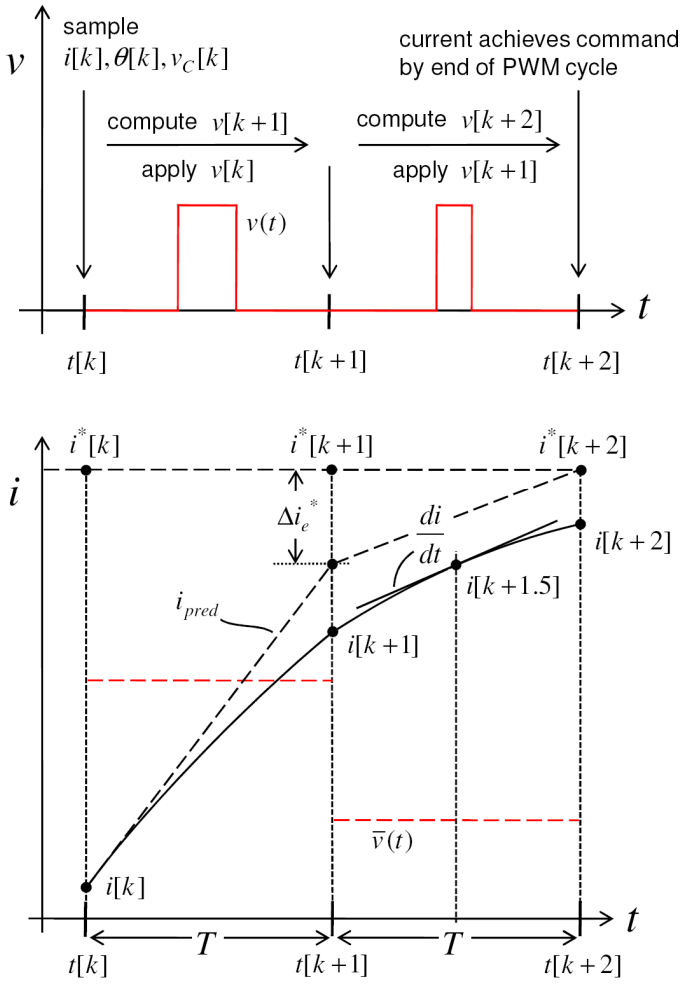


Figure 7.1. Sampling timing with respect to large signal current and inductance waveforms.

The state samples taken at instant k are used to calculate the control voltage during the period between samples k+1 and k+2. At k+1, the controller sets the PWM value to the commanded duty cycle. Over the period from k+1 to k+2 the PWM voltage pulse determines the current trajectory up to $i[k+2]$. In general, the average voltage within the PWM period is unknown and dependent on modulation scheme. It is not until time k+2 that an average voltage of v* has been applied. Therefore, an effective delay of two periods exists - a period of computation time (latency) and a period of PWM averaging - between the sampling instant and control result.

At instant k, the phase voltage, $v[k]$, is known but has not been applied yet(or is just starting to be applied). $v[k]$ determines the current trajectory from k to k+1, so the future current, $i[k+1]$, must be predicted. Then control law B may be shifted one period ahead in time and applied to this prediction to compute the future voltage command, $v^*[k+1]$, needed to drive current error to zero at k+2. Generally, the current command is known two periods ahead of time, so this poses no problem. Notice that this prediction assumes that there will be error at $i[k+1]$, otherwise the current will be equal to the command and there will be no need to predict it. Thus, control law C assumes a deadbeat time of two periods. Again, remember that the voltage, $v[k]$, is the voltage applied from k to k+1. The current prediction may be solved directly from (7.5):

$$i[k+1] = \frac{v^*[k] + i[k](X[k] - R/2)}{(X[k+1] + R/2)} \tag{7.7}$$

Shifting (7.5) one period forward and assuming zero error in two periods provides the equation for control law C:

$$v^*[k+1] = i[k+1](R/2 - X[k+1]) + i^*[k+2](R/2 + X[k+2]) \tag{7.8}$$

The reactances at k+1 and k+2 are computed assuming the current will equal the command at those times, with rotor angle predicted by linear extrapolation (i.e. rotor velocity assumed constant), as in (7.3). A more accurate estimate for x[k+1] could be calculated by assuming the current prediction of (7.7) at time k+1. However, simulation and implementation show nearly negligible performance gains after applying the estimate, so this step is considered optional.

7.6 Special Cases of Control Law C

A simpler implementation of control law C could be constructed by assuming that the current reaches the command in just one period. This would be a reasonable assumption if there were no unpredicted disturbances between instant k and k+2. Thus, the error at k+1 and k+2 are both zero, that is $i[k+1] = i[k+2] = i^*[k+2]$. This would compensate for latency delay while eliminating the need for a current prediction:

$$v^*[k+1] \cong i^*[k+2](R - X[k+1] + X[k+2]) \tag{7.9}$$

However, since this equation is not a function of the present current state, it is useless if error is nonzero. It is essentially open loop, as it cannot correct for current error. This command may be considered a "coasting" command, since it prescribes the voltage necessary to cancel the back emf and resistive voltage drop, such that flux-producing voltage is zero and current remains unchanged (assuming it is initially at the current command). A smart algorithm might switch from the original implementation of control law C to this one at steady-state to free up processing power and reduce instability arising from prediction error. However, this is not investigated in this project.

Another interesting case is the one where the phase resistance is very small relative to the phase reactance. In other words, the angle of the complex phase impedance is very small at the switching frequency. If for example, L=100mH and R=1, then at 10Khz X=1000 and R=1. In this case, the resistance may be neglected and the control law with current prediction substituted to give (7.10). A purely inductive SRM control is less nonlinear and clearly simpler to implement. This case will be used in Chapter 10 to investigate parameter sensitivity.

$$v^*[k+1] \cong -v^*[k] - i[k]X[k] + i^*[k+2]X[k+2] \tag{7.10}$$

Furthermore, if both current command and inductance variation over two control periods is small relative to their magnitudes, such that $i[k] = i[k+2]$, then the equation reduces to:

$$v^*[k+1] \cong -v^*[k] + i_e[k]X[k] \tag{7.11}$$

Rearranging and dividing by two gives:

$$\frac{v^*[k+1] + v^*[k]}{2} \cong \frac{i_e[k]X[k]}{2} \tag{7.12}$$

which implies that the average voltage command over two periods is proportional to the product of current error and inductance. Since the plant response is inversely proportional to inductance, this suggests that a simple proportional controller with a gain that cancels inductance nonlinearity lies at the kernel of deadbeat control law C. This assumption has reduced the system to a linear plant driven by proportional current control. This is an oversimplification, however. The problem with (7.11) is that it neglects the very significant back-emf voltage that arises from

inductance change over two control periods. Still, (7.13) allows us to make the very important conclusion that at the deadbeat controller is a proportional controller. It is important to note that there is no mechanism to integrate current error, so steady-state current error resulting from disturbances or parameter error cannot be corrected.

Because the deadbeat control solution only specifies the system output sampled at kT, it does not preclude inter-sample ripple. In the case of linear systems, inter-sample ripple may occur if the closed loop system contains poles with imaginary parts equal to multiples of the sampling frequency, 1/T[21]. At these frequencies the sampled resonance value will be constant no matter what the phase relationship between sampler and resonant mode, thus producing an unobservable system. However, being a linear feedback phenomena, only the linear feedback filter poles are capable of contributing to inter-sample ripple. All other poles of the nonlinear SRM are time variant. Furthermore, it can easily be shown for a linear deadbeat controller that there will be no steady-state ripple if the control law is constant at steady-state. So inter-sample oscillation can be prevented by preventing control oscillations.

7.7 Summary of Deadbeat Control Design

The following provides a summary of deadbeat control design:

1) Use a first order or higher numerical integration approximation to derive discrete recursive equations to accurately simulate system dynamics up to the time at which zero error may be achieved.
2) Choose the time step to be roughly an order of magnitude smaller than plant time constants. Deadbeat control bandwidth is proportional to control frequency, so noise will dominate the control and cause the control to saturate if switching frequency is chosen too high. A fixed computation time also limits maximum switching frequency. If the control period is too large and response becomes too slow or system dynamics become highly nonlinear and unpredictable within a control period and instability may result. For single rate control, PWM current ripple increases with control period, imposing a performance and audible noise penalty at lower control frequencies.
3) Solve for the input variable, assuming the future output error is zero.
4) Identify future variables that appear in the algorithm and if possible recast the equations by a different integration method to minimize the occurrences of future variables.

5) Using the simulation equations, derive prediction equations for future variables in terms of present variables.
6) Simplify resulting equations if possible and desired. If the computational burden is high, eliminate any dynamics that may be regarded as negligible. If the application does not require performance at a broad range of operating conditions, it may be desirable to eliminate controller compensation that corrects for dynamics that may not be encountered(i.e. consider a simplified model that describes an expected subset of operating conditions). Such computational simplifications may allow a shorter control period for better transient performance.

8 PI and Deadbeat Simulation

A simulation of the asymmetric converter circuitry and SRM was conducted using a hand-coded numerical solution in Matlab. The simulation modeled the DC link capacitor supply voltage by a full bridge rectifier driven by an ideal single phase AC voltage source. Ideal models were used for all rectifier and asymmetric converter diodes and transistors. The diode forward voltage drop was taken to be .7V. Simulation recursive equations were derived by the Euler method and a .5uS time step was used for all simulation. The two phase SRM with no mutual phase coupling presented in chapter 1 was modeled by numerical representations of (2.3) and (2.6). Inductance and torque data from software FEA was stored as a lookup table with 16 columns corresponding to current from 1 to 16 amps and 120 rows for each of the 120 mechanical degrees composing one electrical rotation. The 2D linear interpolation algorithm described in [1] was used to interpolate between data points for both simulation and deadbeat control.

The first order low-pass current filter with 5Khz bandwidth characterized in the previous linear analysis was also simulated. Uniformly distributed white noise was added to the feedback current signal, with a 1A noise envelope added before and a .1A noise envelope added after the current filter to model the noise seen in the implemented feedback circuitry. Single pulse switching at 20Khz was used for the simulation and implementation for the PWM controller for both PI and deadbeat controllers.

8.1 SRM Simulation with PI Control

The PI controller was simulated using the parameters from Table 6.4. Both deadbeat and PI current controllers with speed control were simulated at 1800 and 3600 rpm. As described earlier, a slow response speed controller was used to ensure a constant steady-state current command during the excitation period to provide a measure of step response. The PI controlled current response is shown in Figure 8.1.

It can be seen that there is a tradeoff between current overshoot and tracking performance. By lowering the proportional or integral gain, overshoot may be decreased, but at the expense of current tracking. From a control perspective, the SRM phase has the inconvenient characteristic that its inductance is lowest at the start of the excitation period and it's back emf is

zero (with excitation beginning at the onset of rotor unalignment), while thereafter the inductance increases, establishing back emf. At the beginning of the current step, when error is large, linear control prescribes a large voltage command to drive the current to zero. However, if the initial current step is very good, subsequent tracking would be sluggish because the plant inductance increases to almost 10 times the initial inductance and back emf now opposes the voltage command. In other words, the ideal controller would have a slower initial current rise response but a much faster response and integration during steady-state. A linear controller does not provide both, and so linear control performance is limited for a nonlinear plant.

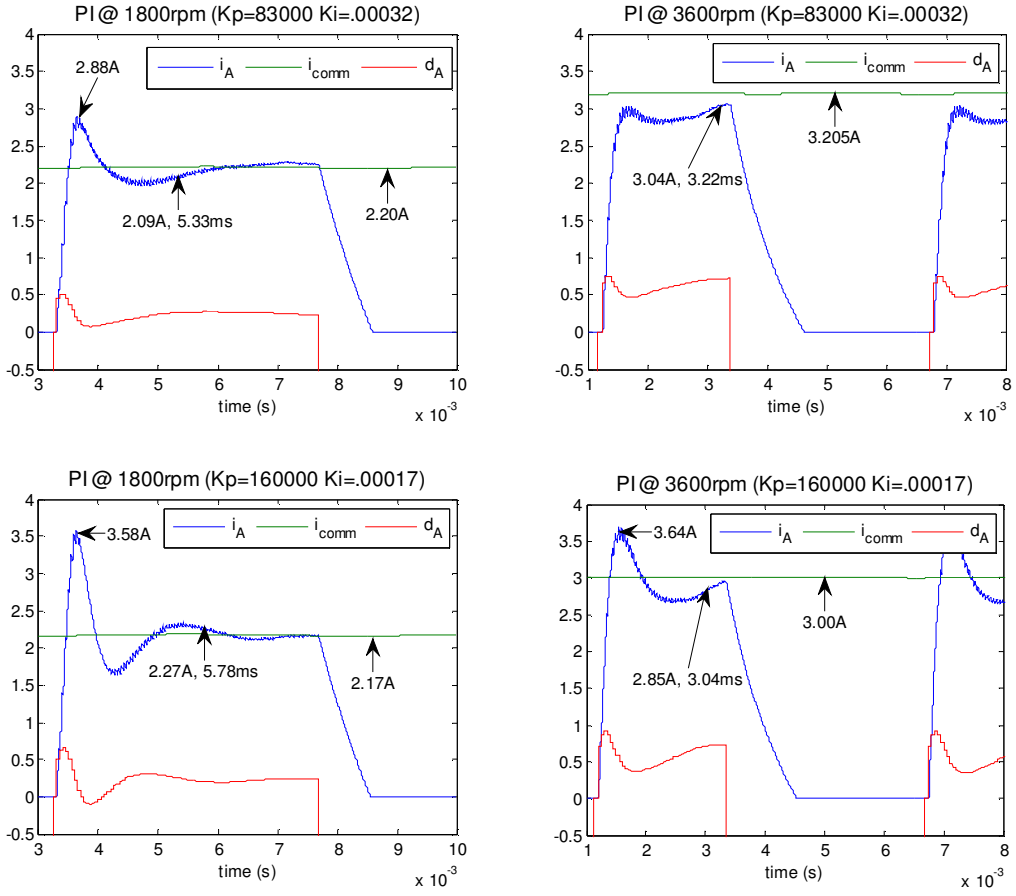


Figure 8.1. PI current control step response for 1800 and 3600 rpm. PI gains in row 1 are designed for 1800rpm. PI gains in row 2 are designed for 3600rpm.

8.2 SRM Simulation with Deadbeat Control

It is desirable to use simulation to investigate various aspects of the deadbeat controller performance and compare these with PI performance. Among these are transient response, sensitivity and noise. It is hypothesized that noise sensitivity will be high with fast transient response.

8.2.1 Simulation of Control Law A,B,C

Deadbeat current control algorithms A-C are simulated at 1800 and 3600 rpm and shown in Figure 8.2. Step responses for controls A and B show large overshoot. This would be the expected result since there is delay in the loop that is not compensated for. In linear terms, this extra delay reduces the loop phase margin, increasing overshoot and settling time and decreasing stability. Despite significant differences in control laws A and B, the step response of each is very similar because neither assumes a latency delay. The predictive control used on control law C produces a very small overshoot and settling time. In linear terms, predictive control is similar to lead compensation, which contributes phase to increase the phase margin.

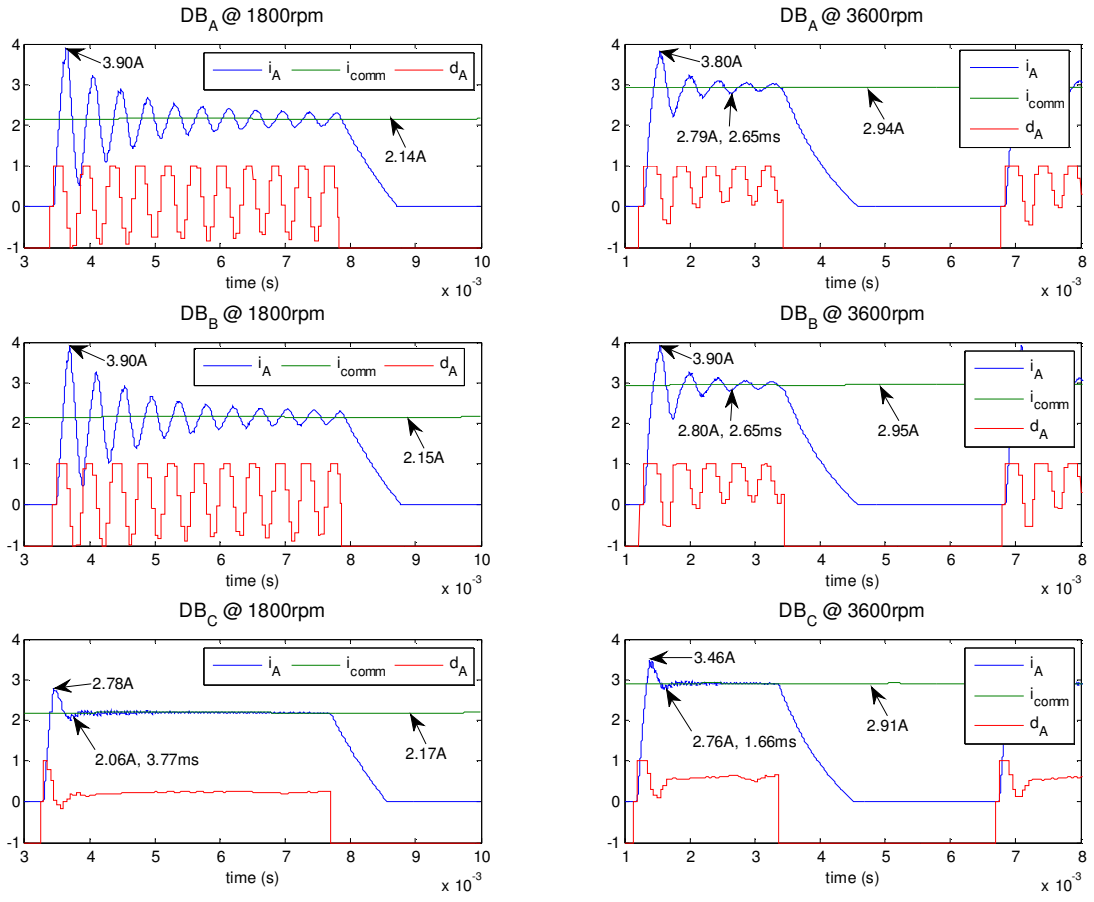


Figure 8.2. Deadbeat current control A,B and C step responses for 1800 and 3600 rpm.

8.2.2 Deadbeat Control law C with Current Filter Compensation and Noise

The presence of noise in the current feedback circuitry necessitates the use of a current filter. As seen in chapter 5, the filter changes the dynamics of the loop at higher frequencies, causing the overshoot accompanying control law C, seen in Figure 8.2. It may be reduced by applying current feedback filter compensation. However, it is essential to investigate the effect of such a compensator on noise. Two uniformly distributed white noise signals, a 1A source added before the current filter and a .1A noise source added after the filter, were simulated to imitate the noise levels observed in the current feedback circuitry. Rather than attempting to accurately model noise frequency characteristics, the uniformly distributed white noise at aforementioned

levels was chosen as a worst-case model and therefore can be expected to provide conservative results.

The current filter model is given previously in the linear analysis and may be used to derive a current estimator by applying the trapezoid rule to (6.3), shifting the time index one period back, and solving for $i[k]$:

$$i[k] \cong \frac{2}{T\omega_f}\left(i_{filt}[k] - i_{filt}[k-1]\right) + i_{filt}[k-1] + i_{filt}[k] - i[k-1]$$

Thus, we have an estimate for the present current state based on the present and past samples of the filtered current. Using this to estimate the phase current for control law C in simulation gives the step response seen in Figure 8.3. Intuition might suggest that filter compensation may be pointless, since a continuous filter compensator would cancel the analog (continuous) current filter dynamics, nullifying its functionality as a filter. This might lead one to question the need for a current filter in the first place. However, as the current feedback noise increases, high control gain causes the voltage command to exceed positive and negative voltage limits, and the functionality of the deadbeat controller is lost. Proper analysis of this problem requires an evaluation of the transfer characteristics of a continuous lowpass filter cascaded with its discrete inverse. For the purposes of this paper, Figure 8.3 suffices to demonstrate the benefit of the current filter compensator in the presence of noise, so this will not be investigated further.

For a 5Khz filter cutoff frequency and 20khz sampling rate the filter time constant is on the order of the sampling period, so a higher order filter compensator would be expected to optimize performance. The step response using a trapezoidal implementation of the current filter compensator does not completely eliminate overshoot even in the absence of noise, as seen in the top right.

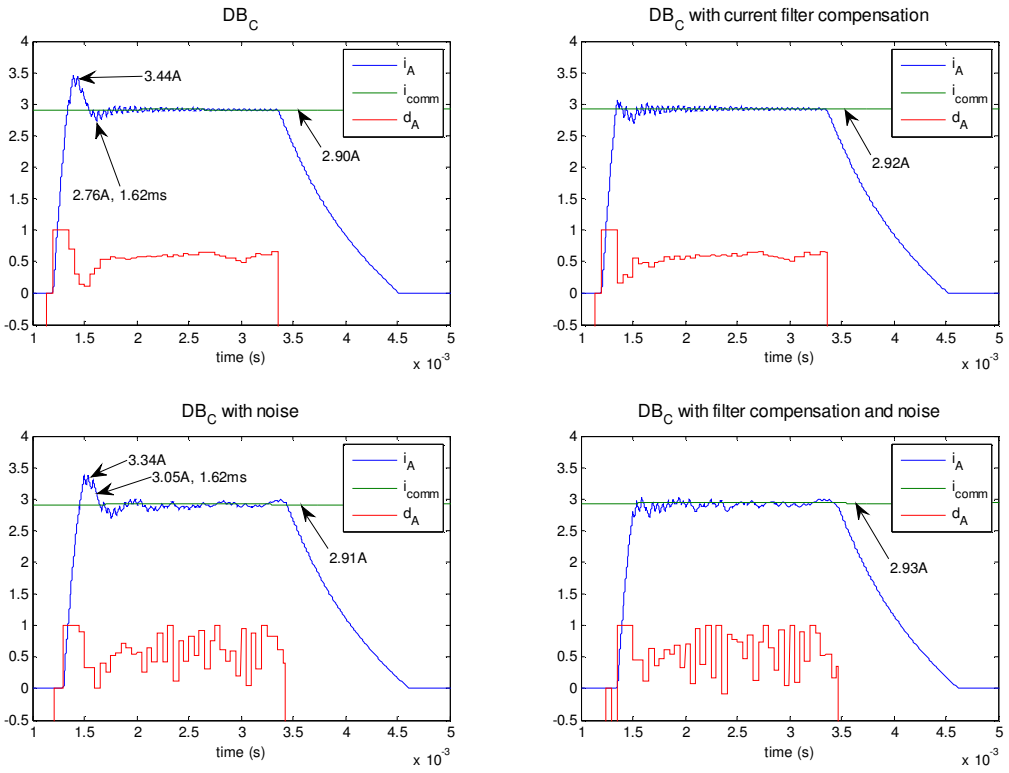


Figure 8.3. Deadbeat control law C showing the effects of noise and filter compensation.

An alternate "current filter delay compensator" was discussed in section 5.1. From Figure 5.1, it is seen that the centered-pulse PWM method ensures that sampling occurs when the instantaneous value of the PWM current is equal to its inter-ripple average. As shown in Chapter by the derivation of, the filtered current signal at sampling instants delayed by $1/\Delta t = \omega_f$ will be approximately equal to the phase current samples at the beginning of a control period. Thus, filtered current is sampled Δt seconds after the beginning of a control period. If Δt is less than the free cpu time at the end of a control period, the delay compensator may be implemented without having to increase the sampling period. In effect, delay compensation reduces latency delay to cancel current filter delay in order to maintain the 1 period delay (originally attributed to latency only) assumed by control law C.

The most desirable quality of the filter delay compensation method is that it does not amplify noise. The linear filter compensator is a discretization of a lead compensator, and as such differentiates the feedback signal, amplifying high frequency noise. The delay compensator,

being a delay, has a unity gain transfer function and so does not amplify signals at higher frequency. Thus, noise reduction is greater for the cascaded delay compensator and filter, reducing voltage command noise and making it less likely that the voltage command will saturate. Figure 8.4 shows a significant reduction in control signal noise for the delay compensator, at the expense of a small overshoot.

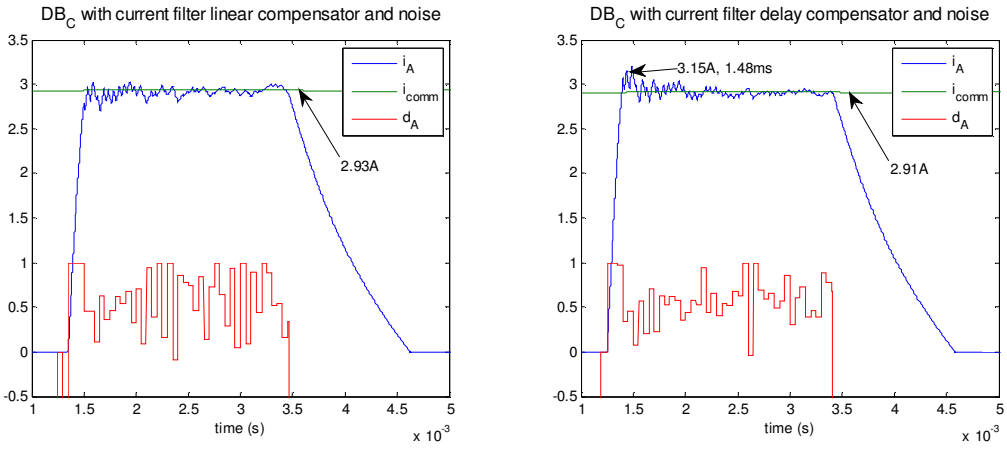


Figure 8.4. Comparison of linear and delay current filter compensation in the presence of noise.

8.3 Transient Performance Results

Figures 7.1 through 7.4 indicate maximum current overshoot (%OS), 95% settling time (T_S), and the current command (i^*), where applicable. Assuming a second order system model, the following equations from [24] allow computation of closed loop bandwidth for a step response:

$$\zeta = \frac{-ln(\%OS/100)}{\sqrt{\pi^2 + ln(\%OS/100)^2}} \tag{8.1}$$

$$\omega_n = \frac{-ln\left(.05\sqrt{1-\zeta^2}\right)}{\zeta T_s} \tag{8.2}$$

Here, ζ is the damping ratio of the equivalent second order system, and ω_n is the bandwidth of the closed loop response in rad/s.

The measure of bandwidth is somewhat misleading, since the closed loop current controller is not linear or a second order system. The back emf changes throughout the excitation cycle, so even if current were to settle immediately, the controller would have to continue to counteract back emf to maintain zero steady-state error. Thus, it is useful to employ another measure of transient performance: tracking error. To evaluate the tracking performance of PI and deadbeat control, the rms current error is calculated over an excitation interval, E, by the following:

$$i_{e,rms} = \sqrt{\frac{\sum_E (i^* - i)^2}{n}} \tag{8.3}$$

Current error during the initial current rise time does not reflect control law performance if the control law is saturated, so current error is summed for only control periods for which the control is not saturated. This relation might also be called the current standard deviation from the command, since is takes the same form as the common equation used for standard deviation, with the difference that the current command would be the average current value. RMS current error is more useful than average of current error magnitude because it effectively applies a larger weighting to large error attributed to faulty control than to error caused by low amplitude noise. This conveniently minimizes the influence of noise in the tracking calculation. As seen in Table 8.1, the rms current error for control law C with noise is about the same as without.

Given physical limitations of loop delay, the optimal deadbeat controller would have a bandwidth corresponding to a settling time of two periods (1 latency delay, 1 pwm delay) and an overshoot of zero. From (8.2) this corresponds to a bandwidth of 7.3kHz. This may be considered the maximum bandwidth possible when the sampling frequency is 20Khz and there is a two cycle delay. However, this is still an unreasonable performance limit since for the case of the DUT, the control voltage is usually saturated for the first two sampling periods of phase excitation. For an inductive load, the current is proportional to the volt-seconds applied, so a higher voltage limit or longer sampling period would reduce the number of periods of saturated control. Discounting saturated periods, the upper bandwidth limit is 5.1kHz. The rise times shown in Table 8.1 include rise time during which the voltage control is saturated, so the bandwidth estimates are quite conservative.

Table 8.1. Summary of PI and deadbeat simulation current step response performance, measured at 3600rpm unless otherwise specified.

Simulation Step Response Transient Performance	$\%OS$	T_s (ms)	ζ	ω_n (rad/s)	ω_n (Hz)	$i_{e,rms}$ (A)
PI, low gain, 1800rpm	31	2.05	0.35	4274	680	0.39
PI, low gain	1	1.95	0.83	2216	353	0.83
PI, high gain, 1800rpm	65	2.45	0.14	9029	1437	0.47
PI, high gain	21	1.85	0.44	3774	601	0.73
DB A, 1800rpm	82	4.05	0.06	11741	1869	0.4
DB A	29	1.35	0.37	6199	987	0.24
DB B, 1800rpm	81	4.05	0.07	11061	1760	0.42
DB B	32	1.35	0.34	6643	1057	0.25
DB C, 1800rpm	28	0.55	0.38	14872	2367	0.19
DB C	19	0.45	0.47	14831	2360	0.11
DB C + linear filter comp	3	0.2	0.74	22827	**3633**	0.09
DB C + noise	15	0.35	0.52	17417	**2772**	0.13
DB C + noise + linear filter comp	1	0.2	0.83	21603	**3438**	0.09
DB C + delay filt comp	8	0.25	0.63	20716	**3297**	0.1
Ideal deadbeat, command saturates first two cycles	0	0.15	0.99	32319	5144	NA
Ideal deadbeat, unlimited voltage command	0	0.1	0.99	48478	7716	NA

9 PI and Deadbeat Control Implementation

The success of the deadbeat controller relies heavily on the accuracy of the SRM model used to develop the control law. If this model does not accurately represent the actual SRM under test, or if the control law is highly sensitive to model errors, performance may suffer. The primary purpose of implementation is to assess the accuracy of the model-based control law and the effects of model inaccuracies on deadbeat control. Deadbeat model parameter sensitivity and noise sensitivity will be investigated further by implementation results. It is also desired to assess PI and deadbeat control methods on a cost or practicality basis. PI and deadbeat control implementation will be compared in terms of the cpu computation time required to compute the control law.

9.1 Microcontroller Setup and Motoring Operation

The PI controllers and all three deadbeat controllers were implemented by fixed-point 32 bit arithmetic in a single c++ program. Thus, the user need only change a variable to change the control law, and each trial is ensured to be using identical microcontroller and peripheral configurations. Wherever possible, hardware configurations are chosen to meet simulation. The digital PWM is configured for centered pulse operation and set for 20Khz operation. The A/D is set to sample phase A and B currents and the DC link capacitor voltage at the beginning of each control period. Analog averaging occurs during sampling over only a few cpu cycles to keep A/D delay low and approximate an ideal sampler. Immediately following A/D conversion and retrieval of data from the A/D registers, a subroutine is called to compute rotor speed and position from raw data from the quadrature encoder peripheral.

Unlike in simulation, rotor angle information is not absolute with respect to the physical rotor since it is provided by encoder pulses rather than binary code. Thus the rotor must first be rotated to a known angle and the angle variable initialized in software before entering motor run mode. By applying dc current to one of the phases, the motor will rotate to the aligned position where phase inductance is maximum. While at this point, the rotor position is initialized to the angle assigned to aligned inductance in the inductance lookup table. It is imperative that the dc excitation period last until after any rotor oscillations cease.

Figure 9.1 shows current in phase A and rotor rpm for an alignment period of 1.5 seconds and subsequent acceleration to 3600rpm. The motor is run as described in simulation, accelerating up to speed (3600rpm base speed or 1800rpm half speed). The speed variable is filtered to remove speed ripple and the filtered signal is regulated by PI control to produce a constant current command with respect to each excitation period. More control and configuration details can be found in the controller c++ code in Appendix B. Documentation for the TMS320C2808 microcontroller and supporting code libraries can be found on the Texas Instruments website.

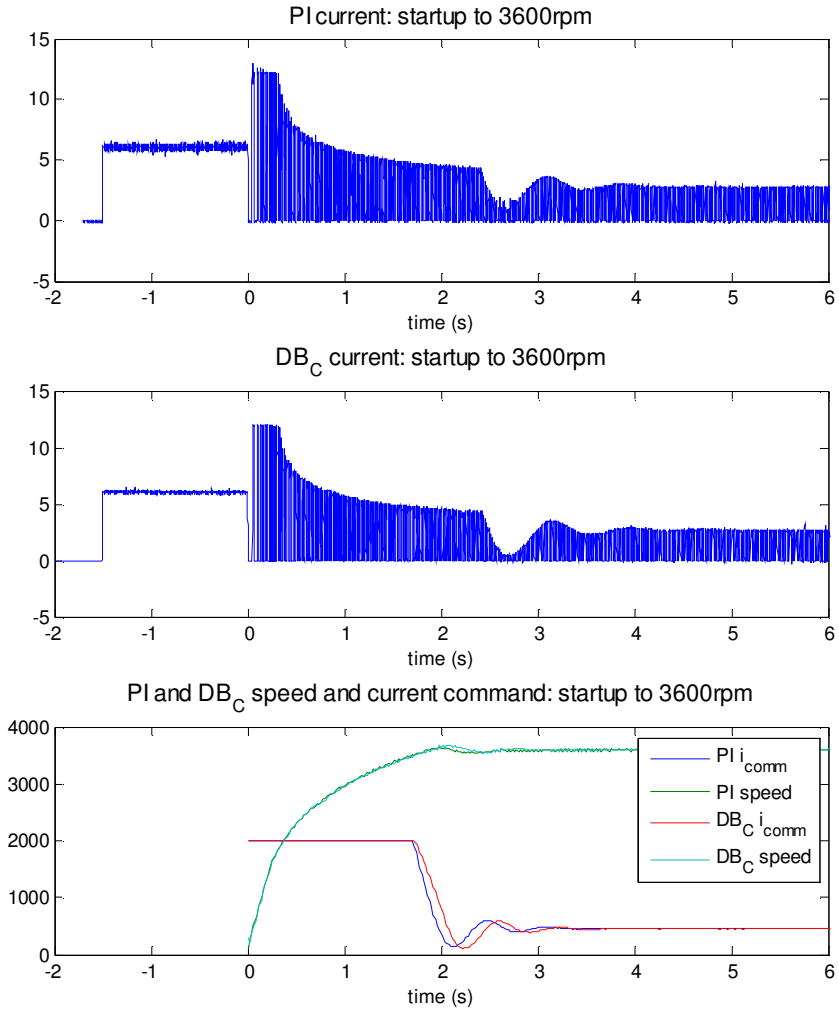


Figure 9.1. Starting and acceleration to 3600rpm for PI and DBC controllers.

9.2 Hysteresis Control Implementation

In the course of evaluating the feasibility of deadbeat control, it would be negligent to disregard another well-established current controller that performs quite well. Hysteresis control, or "bang-bang" current control, has proven to be a very effective controller in simple motor current control applications and it is by far the simplest control algorithm. This current control method was used to measure inductance by the incremental inductance method discussed in section 4.2. Current is regulated at the desired value by applying full control voltage of the appropriate polarity to drive current toward the command whenever the absolute value of current error exceeds a limit, in this case, .125A (or .25A current band).

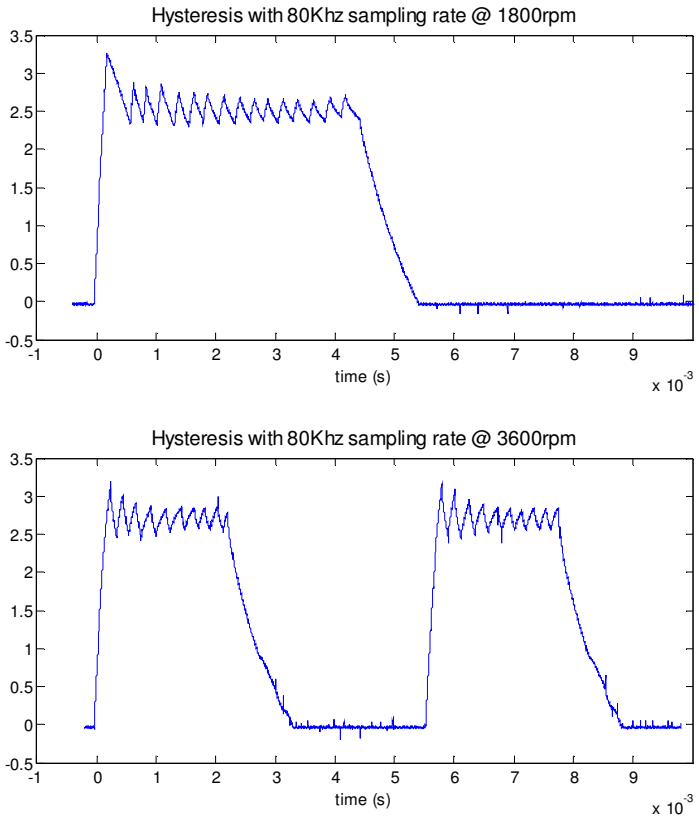


Figure 9.2. Hysteresis current control for 1800 and 3600 rpm.

Figure 9.2 shows the performance of the hysteresis current controller implemented on the TI C2808 controller. Even though the number of cpu cycles required to implement the hysteresis

algorithm is very small, the PI speed control, A/D conversion code, commutation control and encoder interpreter algorithm place a substantial burden on the processor. As a result, total controller latency limits the sampling (i.e. control) frequency to 80kHz. Figure 9.2 indicates a current ripple ranging from about .25A to greater than .5A. The current ripple exceeds the current band command because of the attenuation and delay introduced by the current feedback filter. Thus, it is expected that the ripple is largest at the start of the excitation period when inductance is smallest, such that the current time rate of change is highest and current filter has maximum effect. Although transient rise time is maximum, current ripple is unacceptable and manifests itself by loud raspy acoustics. Ripple magnitude may be reduced at high switching frequencies by removing the current feedback filter, but it cannot be smaller than the current noise envelope, which is quite large (roughly .5 amps). Thus, current noise becomes the limiting factor for hysteresis control in terms of current ripple performance. Due to its simplicity and high frequency operation, hysteresis control is best implemented by analog circuitry. Even if current feedback noise were to be reduced, hysteresis current control is plagued by high switching losses and unpredictable switching frequency characteristics. Both of these issues are superseded by PWM implementations, discussed next.

9.3 PI Control Implementation

PI control was implemented on the TI C2808 microcontroller with the PI gains found in Table 6.3 with $L_0 = L_u$. These are the gains originally calculated by linear analysis and are equal to simulation gains, with the exception that the K_P gain is one half of that used in simulation. This accounts for the approximately doubled natural frequency of the current ripple in Figure 9.3. The code for the PI controller with antiwindup logic is seen in Appendix C. The PI controller of Figure 6.1 requires some scaling in order to convert it to unitless input and output variables used by the TI microcontroller. The scaling procedure is the same as that implemented in the next section for the deadbeat control laws.

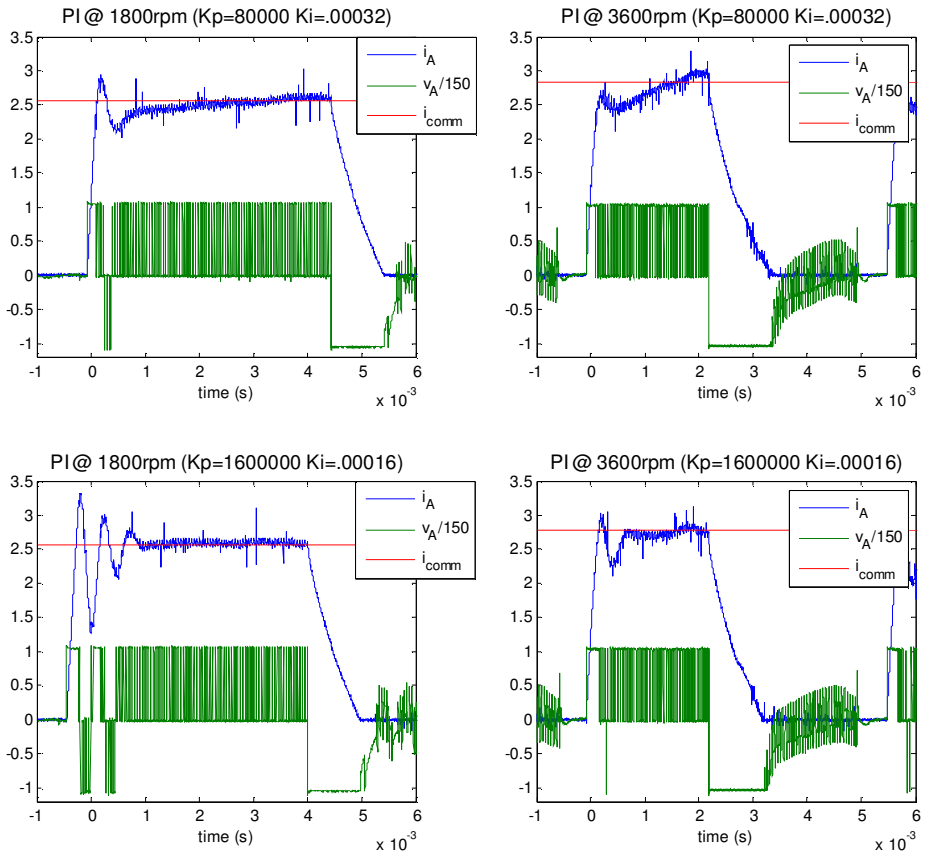


Figure 9.3. PI control results

9.4 Deadbeat Control Law Scaling

Deadbeat control equations in Chapter 7 are not suitable for digital implementation. Raw sampled data from the A/D registers must be scaled to correctly represent electrical units. After calculating the control law and a duty cycle falling between -1 and 1, this value must be scaled up to an integer command to provide to the PWM. These digitally implemented scaling constants must be lumped together if possible to minimize the number of multiplications and to prevent extra data resolution loss due to increased digital signal dynamic range. Control input and output variables, duty cycle, current and capacitor voltage, are related to their binary equivalents (indicated by a subscript b) by the following relations:

$$d^* = d_b^* \frac{K_D}{T}, \qquad i = i_b K_I, \qquad v_C = v_{C_b} K_V \tag{9.1}$$

Substituting these into (7.2) and (7.4) and expressing reactance in terms of inductance(L/T) allows control law A to be expressed as:

$$\frac{d_b^*[k+1]K_D}{T} = \frac{i_b[k]K_I\left(R + \frac{L[k+1]}{T} - 2\frac{L[k]}{T}\right) + i^*{}_b[k+1]K_I\frac{L[k]}{T}}{v_{C_b}[k]K_V} \tag{9.2}$$

Solving for the binary duty cycle command gives an expression in terms of "binary resistance" and "binary inductance":

$$d_b^*[k+1] = \frac{i_b[k](R_b + L_b[k+1] - 2L_b[k]) + i^*{}_b[k+1]L_b[k]}{v_{C_b}[k]} \tag{9.3}$$

where the binary resistance and binary inductance are defined as:

$$R_b = RTK_b, \qquad L_b = LK_b, \qquad K_b = \frac{K_I}{K_D K_V} \tag{9.4}$$

Notice, the only parameter in (9.3) that changes with control period is the binary resistance. This arises from the convenient fact that the binary duty cycle command for the TI PWM varies proportionally with the PWM period. Thus, the lookup table for L_b need not be changed for a change in the control period. Applying the same principles to control law B gives:

$$d_b^*[k+1] = \frac{i_b[k]\left(\frac{R_b}{2} - L_b[k]\right) + i^*{}_b[k+1]\left(\frac{R_b}{2} + L_b[k+1]\right)}{v_{C_b}[k]} \tag{9.5}$$

and for control law C:

$$d_b^*[k+1] = \frac{i_b[k+1]\left(\frac{R_b}{2} - L_b[k+1]\right) + i^*{}_b[k+2]\left(\frac{R_b}{2} + L_b[k+2]\right)}{v_{C_b}[k]} \tag{9.6}$$

Expressing the current predictor for control law C in terms of inductance and substituting (9.1) gives the following:

$$i_b[k+1]K_I = \frac{v_b{}^*[k]\frac{K_D}{T}K_V + i_b[k]K_I\left(\frac{L[k]}{T} - R/2\right)}{\left(\frac{L[k+1]}{T} + R/2\right)}$$

where $v^*[k] = d^*[k]v_C[k] = d_b^*[k]\frac{K_D}{T}v_{C_b}[k]K_V = v_b{}^*[k]\frac{K_D}{T}K_V$. Solving for $i_b[k+1]$ and rearranging gives:

$$i_b[k+1] = \frac{v_b{}^*[k]\frac{K_D K_V}{K_I} + i_b[k](L[k] - TR/2)}{(L[k+1] + TR/2)} = \frac{v_b{}^*[k] + i_b[k](L[k]K_b - RTK_b/2)}{(L[k+1]K_b + RTK_b/2)}$$

Substituting (9.4) gives the dsp-compatible form for the current prediction of control law C:

$$i_b[k+1] = \frac{v_b{}^*[k] + i_b[k](L_b[k] - R_b/2)}{(L_b[k+1] + R_b/2)} \tag{9.7}$$

9.5 Deadbeat Results

Control laws A, B and C, as shown in section 9.4 were implemented and used to regulate phase current for the current commands corresponding to 1800 and 3600 rpm. The filter delay compensators proposed in section 8.2.2 have not been implemented. As in simulation, experimental phase current waveforms driven by control law A and B are nearly indistinguishable. Thus, the response to control law A will suffice to demonstrate both responses. Figure 9.4 shows current and normalized voltage waveforms for control laws A and C captured from an oscilloscope.

Comparing these results to the predicted simulation results from Figure 8.2 leads to a few observations. First, the oscillation predicted for control laws A and B is present at roughly the predicted oscillation frequency, although at a lower amplitude than predicted. Initial ringing from current rise decays as predicted, but only to a point, such that the response resembles a bistable system with undecaying oscillation. This may suggest a large-signal system linearization with stable poles but a small-signal linearization with unstable poles, such that the nonlinear system has a quasi-stable equilibrium in the vicinity of the current command. In other words, the real part of the system poles drift toward equilibrium at zero. Secondly, it is important to note that the

overshoot of control law C is almost negligible, as opposed to the approximately 20% overshoot predicted by simulation.

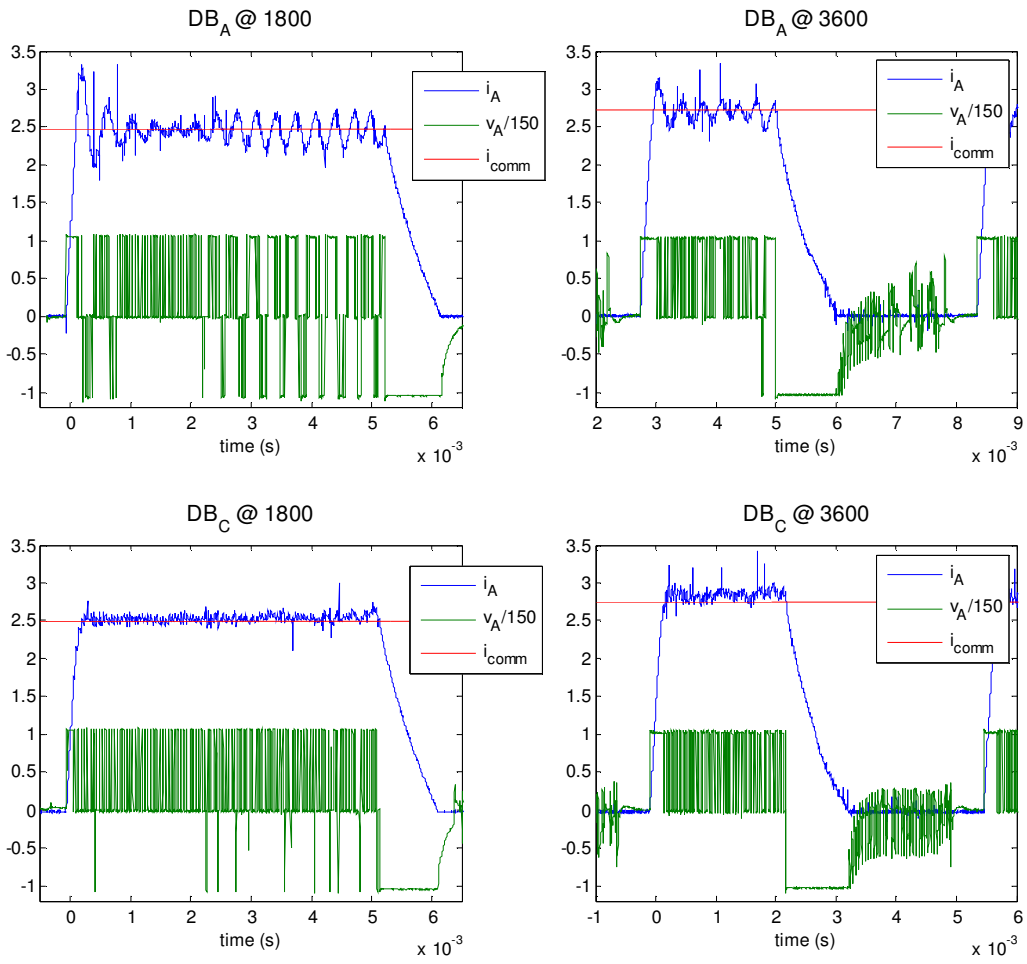


Figure 9.4. Deadbeat current response to control law A and C

Both of these observations appear to indicate higher system damping than expected. Underestimated phase resistance might be the cause. However, this is highly unlikely because a significant resistance error would result in significant tracking error, yet the apparent tracking error is small. It is known that unmodeled eddy currents contribute significant losses in motor cores. A core with well designed laminations and electrical steel attempts to minimize their effect, but they cannot be eliminated. These induced motor core currents are proportional to the

rate of change of current and so work to oppose a change in current. It is most likely that eddy currents are responsible for the damping, since the damping in Figure 9.4 corresponds to current transients and not steady-state tracking. The effects of model parameter error will be investigated more in Chapter 10, Deadbeat Control Sensitivity.

For completeness, a larger signal step response was taken during operation at a slower speed to allow time to reach steady-state. Figure 9.5 shows an implementation of the PI and deadbeat current responses to an initial small step followed by a large step in the middle of the excitation period. Deadbeat current tracking appears to have offset error, for both low and high steps. This may be due to parameter error, which will be investigated in Chapter 10.

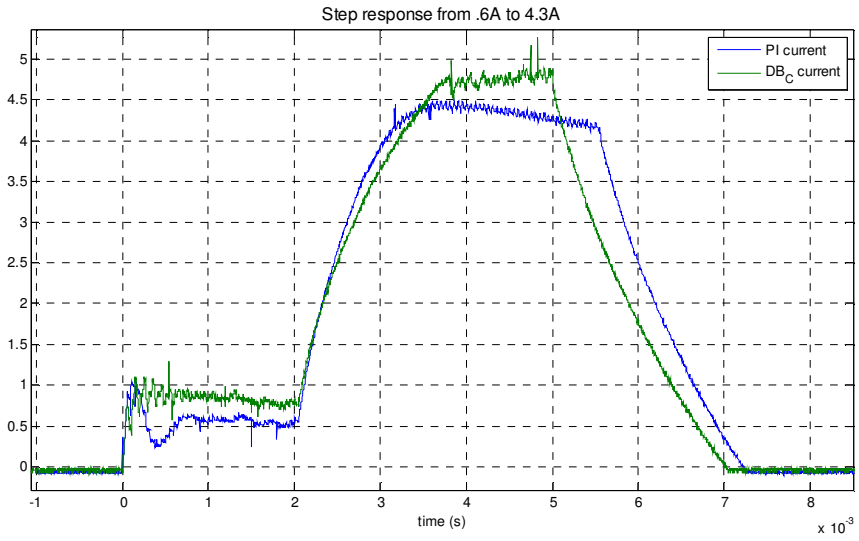


Figure 9.5. Large signal step response comparison of PI vs deadbeat.

9.6 Benchmarking

It is essential to evaluate the deadbeat controller in terms of algorithm computation time, because this determines the maximum control frequency that may be applied for a given current controller and processor. If the control frequency limit is lower than the optimal control frequency, performance will suffer. Thus, for a deadbeat controller to be useful, it is important to demonstrate that it is simple enough to implement in place of a PI controller - that is, without requiring the cost of more processing power. Much of the computational time required, such as

A/D conversion, encoder data processing, commutation algorithms, current limiting control logic, and PI speed control, is common to both PI and deadbeat current controllers. Only the control law algorithm differs between the different controllers under test, and so only its code determines differences in overall controller latency. In this section, current control law C will be compared with PI current control in terms of latency.

The number of additions/subtractions, multiplications and divisions required to implement a controller gives an indication of latency. For the TI C28xx processor architecture, a standard multiplication requires 4 execution cycles, a division requires 63 execution cycles and an addition or subtraction only one cycle. Divisions or multiplications by a power of two are implemented by an arithmetic shift requiring only one cycle. The number of arithmetic operations and corresponding total cycles associated with the PI controller and deadbeat control law C are compared in Table 9.1. Control law computations are taken to be those required to compute the duty cycle command from the current command. Additional computations are required by the inductance lookup table interpolation function, which uses the algorithm described in [1] and coded in Appendix D. Additional table index calculations are required prior to calling the lookup table. These numbers are based on the code found in Appendices C and D.

Table 9.1. CPU burden for PI and deadbeat control algorithms

Code section	Type of operation	Number of Operations	
		PI control	Deadbeat control law C
Control Law	addition/subtraction	2	6
	multiplication	5	3
	division	0	2
Lookup table	addition/subtraction	N/A	13
	multiplication		3
	division		0
Lookup table index calculations	Addition/subtraction		2
	multiplication		5
Number of calls to lookup table:			3
Total execution(cpu) cycles:		**22**	**241**

By lumping all scaling constants into the binary units for resistance and inductance, the number of arithmetic operations is minimized for both control algorithms. Although the TI

processor has efficient DSP type multiply-and-accumulate(MAC) instructions, programming was implemented in c++, so the choice to use MAC instructions was left to the compiler. Deadbeat control has the computational disadvantage of two divisions and the need for interpolation algorithms. Because of these, the deadbeat controller requires roughly 10 times as many cpu cycles as the PI controller. Deadbeat control memory requirements are also significantly greater than for PI control. In this case, the lookup table is stored in program memory, and so memory access is as fast as it is for all other variables. In addition, the space requirements for the 8 X 60 array of 32-bit memory is generally not a problem for modern controllers.

Programming fixed point arithmetic often requires additional scaling (in addition to that presented in section 9.4) in order to maximize variable resolution and prevent overflow or underflow conditions. The need for scaling depends on the dynamic range displaced by control signals during processing by the control law. The number of scalings, in the form of multiplications, required for a given control law is difficult to predict before implementation. There are miscellaneous computations such as type conversions, arithmetic shifts, and signal error prevention logic. Ultimately, the number of cpu cycles required per control period is dependent on the processor architecture and programming compiler, and is difficult to estimate prior to implementation. The most conclusive means of evaluating cpu latency is to measure total cycles per period by the cpu clock. Table 9.2 shows average (nominal) and maximum cpu cycles required per period for each algorithm. The processor speed is 100Mhz, corresponding to 5000 available cpu cycles per control period at 20kHz. The 1460cycle maximum observed for the PI controller indicates that the PI controller code is about 3.2 times faster than the deadbeat controller code, with a maximum of 4654 cycles. Cpu utilization is 93% for the deadbeat controller and 29% for the PI controller.

Table 9.2. Total CPU cycles: PI vs deadbeat control

CPU cycles per period	PI control	Deadbeat control law C
Nominal cycles	720	3900
Maximum cycles	1460	4654

10 Deadbeat Control Sensitivity

It is perhaps an axiom of control theory that controller parameter sensitivity increases with transient performance. It is expected that the high performance deadbeat controller will be less robustness than its linear PI counterpart. A controller that works in the ideal world of simulation is no good if it is too sensitive to controller parameter error and current feedback noise to function correctly. This section will investigate simulation and experimental tests of deadbeat control parameter sensitivity and present equations for predicting error for both control parameters and current feedback noise. It will be assumed that the underlying control model is accurate enough so that current tracking error may be attributed primarily to parameter error in the control law rather than model inaccuracies. The validity of this assumption will be tested by comparing simulation and experimental tests of parameter sensitivity.

10.1 Controller Parameter Sensitivity

There are two parameters in the SRM electromagnetic model: phase resistance, R, and phase inductance (or alternatively, phase flux). The phase resistance is a simple scalar, easy to measure offline, and possible to estimate online by adapting methods presented in Chapter 4. As stated previously, the characteristics of both phases are considered identical. To the extent that the underlying SRM model is correct, the performance of the closed loop current response will depend on the accuracy of these parameters.

Of the two parameters, the inductance estimation is by far the more dominant for a relatively efficient SRM motor such as the 6/3 motor under test. In control law C, (7.7) and (7.8), phase resistance and inductance(in terms of reactance) always appear together as a sum or difference of $R/2$ and $X[k+n]$. For the phase resistance of 1.5Ω, R/2 is .75Ω. In comparison, the minimum reactance at 20kHz, X, corresponding to the minimum inductance of 6mH, is 120Ω, while the maximum reactance corresponding to the maximum inductance of 103mH is 2060Ω. Thus, the phase resistance never accounts for more than .6% of the total phase impedence! This will allow us to make the very convenient assumption that phase resistance is negligible for deadbeat current control of the motor under test.

In general, the error function, $e_{i,\theta}$, between the inductance parameter, $\hat{L}$, and the true inductance, L, varies with both current and rotor angle and is unknown. The subscripts indicate a function of current and rotor angle. If this error is taken to be multiplicative then we have:

$$\hat{L}_{i,\theta} = e_{i,\theta} L_{i,\theta}$$

While the value of the error function is unknown, it is beneficial to model its structure. This model may be unique to the inductance measurement technique and may consist of noise to a degree, but to a greater degree it usually consists of some smooth function correlating to the true inductance surface. For example, the inductance surfaces obtained by three methods in Figure 4.8 appear to differ mainly by some offset or scalar and to a lesser extent some complex relation in terms of inductance and rotor angle. Judging by the smoothness of the curves, random error is small. Similarly, it will be assumed that the measured inductance is a linear function of the true inductance, such that the inductance estimate may be represented by a scale and offset of the true inductance:

$$\hat{L}_{i,\theta} = e_{i,\theta} L_{i,\theta} = k_1 L_{i,\theta} + k_2$$

Translational model error, such as that resulting from an offset error in the rotor angle, will be ignored since it is negligible in practice. All measurement methods demonstrate a high degree of precision in estimating the angle of minimum and maximum inductance. This is good, considering translational error in the inductance parameter has caused instability in a deadbeat implementation.

It is useful to consider the likely case where estimated inductance is proportional to true inductance:

$$\hat{L}_{i,\theta} = e_{i,\theta} L_{i,\theta} = k_1 L_{i,\theta}$$

If phase resistance is neglected, k_1 factors out of deadbeat control law C [see (7.7) and (7.8)] such that a constant gain error is introduced into the control loop:

$$v^*{}_{R=0,L=\hat{L}} = k_1 v^*{}_{R=0,L=L}$$

The effect of inductance scaling error on current step response with deadbeat control was simulated for $k_1 = .5, .75, 1, 1.25,$ and $1.5.$ at 3600rpm. Figure 10.1 displays these responses with current and normalized voltage commands. Overshoot is attributed to the absence of current

filter compensation, which was not included to correspond to implementation results. It appears that voltage command ringing occurs when $k_1 = .5$. This dynamic may be attributed to the state(memory element) introduced in control law C by virtue of the fact that the voltage command is a function of the previous voltage command.

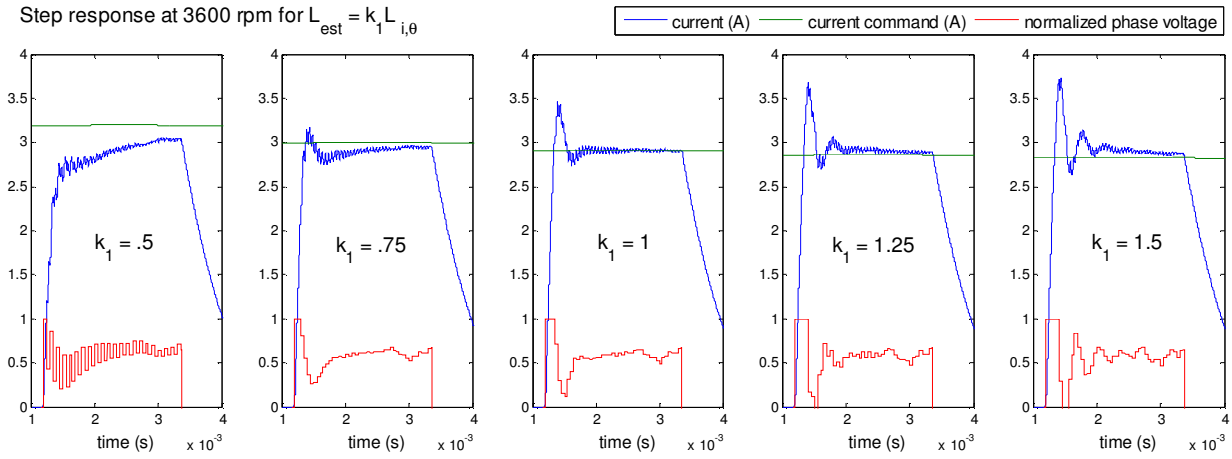
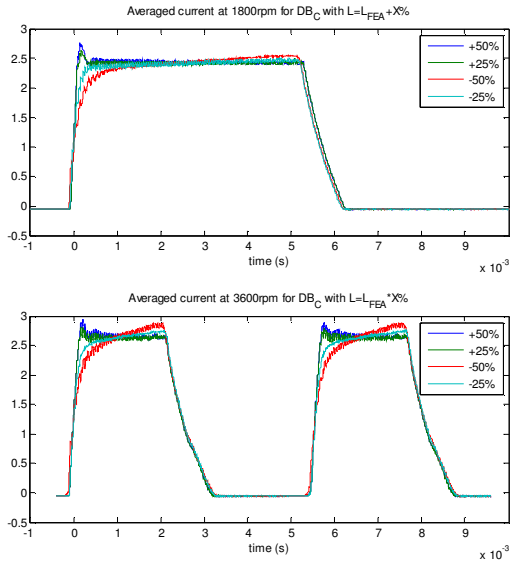
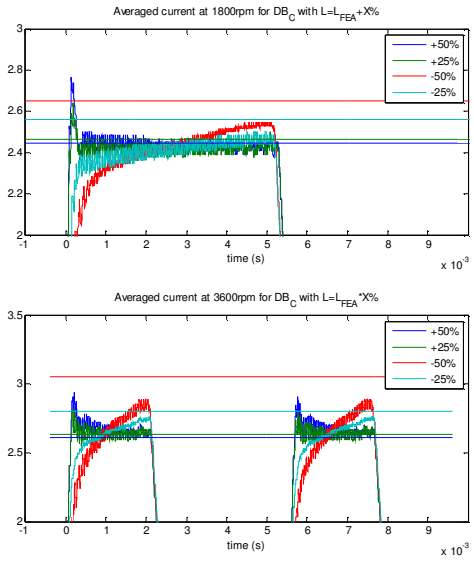


Figure 10.1. Current step response for various inductance estimation errors

The same conditions were implemented and current waveforms recorded by oscilloscope in Figure 10.2. The current waveforms, particularly in terms of current tracking, correspond closely to simulation. As seen before, a lower overshoot than predicted indicates a greater degree of transient current damping.



a) Current waveforms



b) Close-up of current waveforms with current commands

Figure 10.2. Oscilloscope current excitation for inductance estimate error at 1800 and 3600rpm averaged over 100 cycles.

To further quantify the effects of parameter error including resistance error, 25 simulations were conducted, including 5 widely varying resistance estimates for each of the 5 inductance estimates represented above. The normalized rms current error, as defined in (8.3), was computed for each simulation and graphed in Figure 10.3. This gives an indication of the effect of parameter error on current tracking ability. As expected, the large steady-state current error associated with inductance underestimation occurs at all values of resistance estimate, although it is reduced slightly by resistance overestimation. Despite increasing overshoot, the tracking error is lower for inductance overestimation. Also, for overestimated inductance error, Figure 10.3 suggests an increase in rms error that is proportional to R_{est}/R.

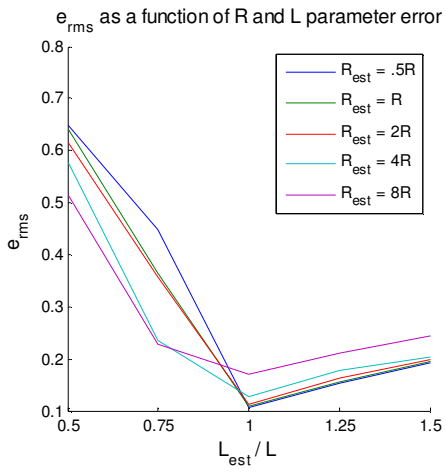


Figure 10.3. RMS current tracking error as a function of R and L estimate error

While graphs are useful, it would be more useful to obtain an expression that predicts the current error for a given current, current command, and inductance error. Maintaining the assumptions of negligible resistance, scalar inductance error, and an otherwise correct SRM current model, the current error is easily calculated. The following calculations assume that the current state, $i[k]$, is known. In the case of the deadbeat current controller used without current filter compensation, $i[k]$ is not equal to the phase current, particularly during transients. The following derivation does not predict the initial transient current overshoot and decay due to filter delay, but simulation has shown that it does predict the current error very well when ringing is ignored.

The simplified current prediction,

$$i[k+1] \cong \frac{v^*[k] + i[k]X[k]}{X[k+1]} \tag{10.1}$$

may be shifted by one period to give:

$$i[k+2] \cong \frac{v^*[k+1] + i[k+1]X[k+1]}{X[k+2]} \tag{10.2}$$

Substituting (10.1) into (10.2) predicts the current trajectory two periods ahead in terms of the initial current and the voltages, $v^*[k]$ and $v^*[k+1]$, applied during those periods:

$$i[k+2] \cong \frac{v^*[k+1] + v^*[k] + i[k]X[k]}{X[k+2]} \tag{10.3}$$

Equation (7.10), rewritten with the reactance estimate, $\hat{X}$, expresses the simplified control law C:

$$v^*[k+1] \cong -v^*[k] - i[k]\hat{X}[k] + i^*[k+2]\hat{X}[k+2] \tag{10.4}$$

Substituting (10.4) into (10.3) predicts the actual current at the end of the two cycles based on the control law using the reactance estimate. Note, this current would equal the command if $k_1 = 1$ and $k_2 = 0$. Thus the current error is the difference between the current command and this prediction:

$$i_e = i^*[k+2] - \left(\frac{-v^*[k] - i[k]\hat{X}[k] + i^*[k+2]\hat{X}[k+2] + v^*[k] + i[k]X[k]}{X[k+2]}\right) \tag{10.5}$$

Substituting $\hat{X} = k_1 X + k_2$ to express current error in terms of linear inductance error with offset gives:

$$i_e = i^*[k+2]\left(1 - k_1 + \frac{k_2}{X[k+2]}\right) + i[k]\frac{(X[k](k_1 - 1) + k_2)}{X[k+2]} \tag{10.6}$$

If the reactance error is considered to be scalar only, such that $k_2 = 0$, then

$$i_e = \left(i^*[k+2] - i[k]\frac{X[k]}{X[k+2]}\right)(1 - k_1) \tag{10.7}$$

10.2 Controller Noise Sensitivity

Like parameter error, noise also introduces performance-degrading error into the deadbeat controller. However, in contrast to relatively constant parameter error, noise error occurs at high frequency rather than DC. Current feedback noise is the largest noise source in the control system. The main concern regarding noise is its effect on the voltage input, because if noise is too great, excessive command saturation may reduce the deadbeat controller to a glorified hysteresis controller.

Figure 10.4 shows three normalized control voltage waveforms retrieved from data stored on the microcontroller. The graphs prompt the following observations regarding deadbeat noise sensitivity:

1) The voltage command of the PI controller is relatively insensitive to noise. This is expected, as the PI controller cutoff frequency is well below the sampling rate.
2) The deadbeat voltage command error is roughly proportional to switching frequency. At 10kHz, the maximum noise amplitude imposed on the control voltage is roughly .5. At 20kHz, the maximum amplitude is roughly .8.
3) Control voltage noise appears to be proportional to phase inductance, which increases during the excitation cycle. A given current error requires a greater voltage correction for a greater inductance. If the current error is attributed to feedback noise, then the voltage command will amplify that noise in proportion to inductance.

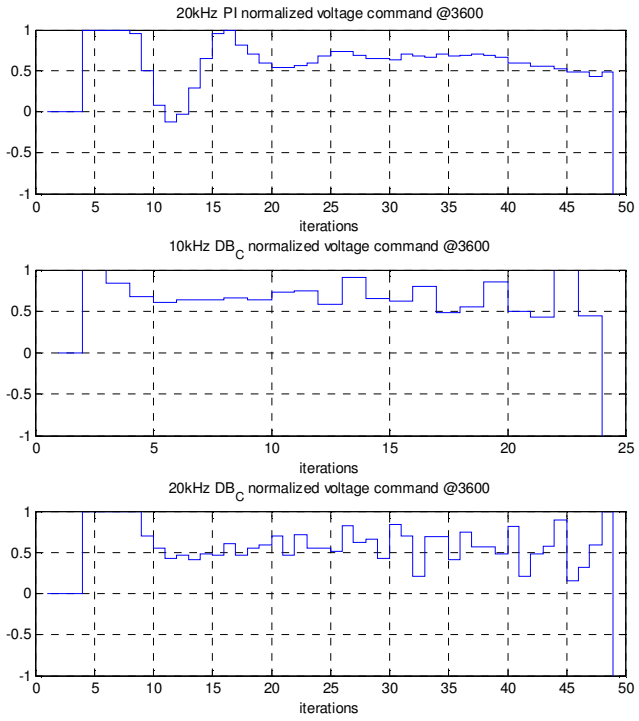


Figure 10.4. Experimental waveforms of normalized control voltage showing the effect of sampling frequency and control type in the presence of current feedback noise.

To impart precision to the above observations, the effects of current feedback noise may be investigated analytically in the same way that sensitivity to inductance error was investigated in Section 10.1. Also it will be assumed that the current state is known. is compensated to cancel filter dynamics, such that noise is the only appreciable source of error. If the current feedback error is attributed to additive noise only such that $\hat{\imath} = i + N_k$, and the current estimate is used to compute an inductance, $\hat{L}$, then the expressions in Table 10.1 may be used to predict current error, i_e, and normalized voltage command error, $d_e{}^*$. The general error approximation is given, along with the expression for the case of small signal noise. The derivations for these expressions may be found in Appendix E.

Table 10.1. Expressions for predicting phase current error and normalized voltage command error as a function of feedback error.

Error term	General approximation	Approximation assuming small noise signal
i_e	$\dfrac{\hat{\imath}[k]\hat{L}[k] - i[k]L[k]}{L[k+2]}$	$\dfrac{N_k \dfrac{\partial \lambda}{\partial i}[k]}{L[k+2]}$
$d_e{}^*$	$\dfrac{\hat{\imath}[k]\hat{L}[k] - i[k]L[k]}{TV_C}$	$\dfrac{N_k \dfrac{\partial \lambda}{\partial i}[k]}{TV_C}$

The expressions are consistent with observations 2 and 3. Regarding observation 1, the deadbeat controller has no explicit integrator. However, its sampling action has a crude filtering effect by virtue of the fact that any signal reconstructed by samples has no frequency components greater than twice the sampling frequency. This aliasing effect follows from Shannon's sampling theorem. The approximations for $d_e{}^*$ affirm observation 2, that the voltage command it is in fact proportional to control frequency. Observation 3, that the voltage command is proportional to inductance for a given current noise, is suggested by the presence of inductance in the numerator of the general approximation for $d_e{}^*$. However, the small noise signal approximation makes the very important clarification that in fact voltage command error is proportional to the incremental inductance alone (constant current noise). This is consistent with

observation 3, as incremental inductance, $\frac{\partial \lambda}{\partial i}$, is equal to inductance when the core is not saturating, which is generally the case at 1800rpm and 3600rpm operation.

11 Adaptive Modifications to the SRM Deadbeat Current Controller: Future Research

Deadbeat control has one glaring drawback: its inability to compensate for steady-state current error. This was seen clearly in the analysis of model parameter error in the previous chapter. PI control, adaptive control, iterative learning control and any control that directly or indirectly integrates output error has the capability to compensate for steady-state error to some extent. However, the deadbeat control law for the SRM contains no integrator. Nor is it a nonlinear class of proportional controller (if there is such a thing) since it is not strictly a function of current error. The voltage command is a function of multiple inputs: phase current, current command, capacitor voltage, and the previous voltage command. The absence of integrator action is not a total detriment to deadbeat design, however. Filtering and adaptation techniques introduce additional phase lag into the system, slowing transient performance, so if the plant model is known to a sufficient degree of certainty, adaptation dynamics may not be desired.

11.1 Online Model Parameter Estimation

Simple online model adaptation should be possible. Resistance may be easily and accurately calculated online by (4.4) at the cost of two additions per period and one division per excitation cycle. If the model reactance error is predominantly scalar, $\hat{X} = k_1 X$, then (10.7) could be used to calculate the error, k_1, directly. However, a more realistic case is that reactance has scaling and constant error, $\hat{X} = k_1 X + k_2$. The values of error parameters k_1 and k_2 may be found by the method of least squares. Substituting $X = \left(\hat{X} - k_2\right)/k_1$ into the right side of (10.1) gives a current prediction, i_{pred}, in terms of true reactance and the error parameters. The least squares summation,

$$\varphi(k_1, k_2) = \sum_{1}^{n} \left(i_n - i_{pred,n}\right)^2$$

will be minimum when the reactance error parameters are correct. This occurs when

$$\frac{d\varphi}{dk_1} = \frac{d\varphi}{dk_2} = 0$$

If the expression for i_{pred} is linear with respect to the error parameters, the parameters may be found easily and calculated real-time at the end of each excitation cycle. If the expression for i_{pred} is nonlinear, a numerical solution might be necessary, rendering the calculation too complex for a real-time controller.

If an adaptive control law is chosen to update the error parameters, it should be chosen to have very slow response or have a long update time, such as once every excitation cycle. This effectively allows the deadbeat control law to operate independently of the adaptive control law, and also filters noise from the calculation.

11.2 Iterative Learning Control

Another form of adaptive control is the iterative learning controller (ILC). Iterative learning control is used most commonly in applications requiring compensation to eliminate a periodic independent disturbance (i.e. external to the system). However, it may be employed to eliminate a dependent disturbance (i.e. coupled to system dynamics) if designed with a slow enough "learning" response, so as not to disturb the faster system dynamics. Such an ILC has been proposed for SRM current control in [5]. The appeal of the described ILC is its ability to employ a very simple control law and a parameter array to achieve zero current tracking error at steady-state operation. Each element of the parameter array corresponds to each control period of the excitation cycle during steady-state, i.e. constant motor rpm. The array may be considered an implicit model of the plant corresponding to the state trajectory, which is constant with respect to the phase excitation cycle at steady-state.

The iterative learning control law is:

$$u^k[n] = u^{k-1}[n] + \Gamma(i^*[n] - i^{k-1}[n]) \tag{11.1}$$

where k indicates excitation cycle and n indicates a control period within an excitation cycle. The control block diagram associated with this is:

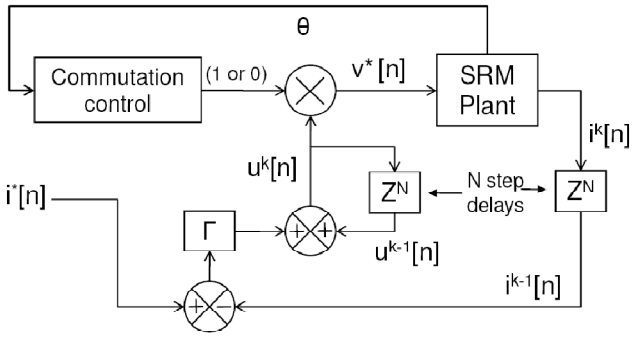


Figure 11.1. ILC control block diagram for SRM current control

A simulation using the control law of (11.1) with feedback noise by the control scheme of Figure 11.1 produces the phase excitation current response in Figure 11.2.

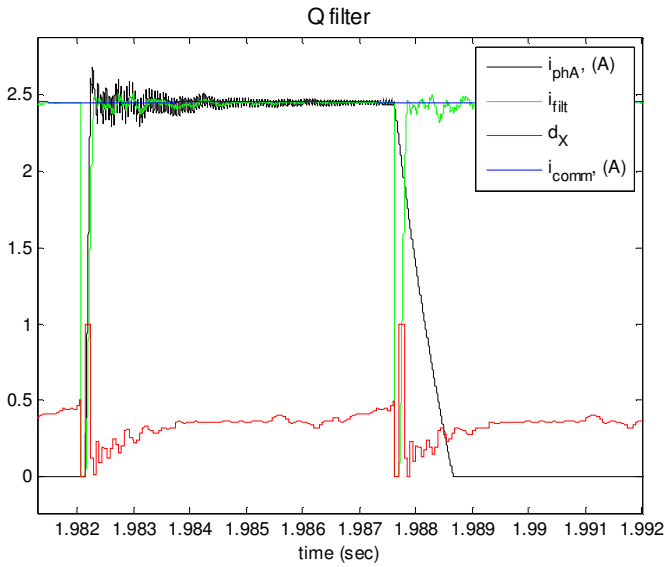


Figure 11.2. Excitation response simulation for ILC after parameter convergence

Figure 11.3 demonstrates convergence of 3 arbitrarily chosen adaptive parameters over 300 excitation cycles. The adaptive parameter array corresponds to the steady-state current-angle trajectory, so it may be processed after each excitation cycle to update the SRM inductance model array to be used for deadbeat control. Only the region of the inductance curve occupied by the steady-state operating region need be corrected to deliver high performance at steady-state. In contrast to the adaptive controller proposed to correct a linear inductance estimate error, the ILC method could be used to eliminate an inductance error of arbitrary structure (the shape of the

inductance profile after the process may not resemble the shape before). However, the details and viability of such an unfounded proposal will be left for future research.

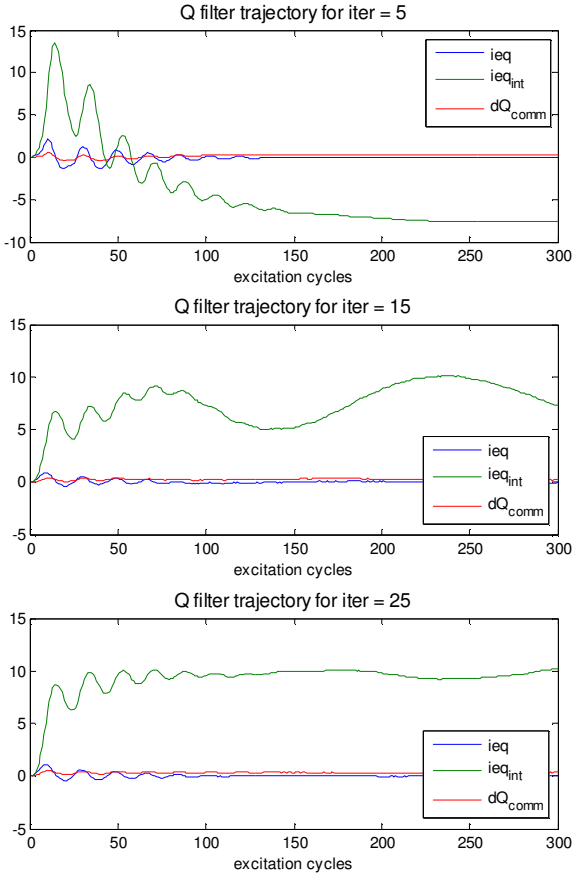


Figure 11.3. Convergence trajectories for three ILC adaptive gains, and current error, error integral and duty cycle command for each.

12 Summary and Conclusions

Deadbeat control is clearly a viable alternative to PI control. In summary, the following observations may be made:

- Linear analysis suggested that the upper limit on PI control bandwidth for the system in question is about 25% of the current feedback filter cutoff frequency. Yet simulation and implementation verified that this bandwidth is lowered by SRM nonlinearities.
- The deadbeat controller exceeded the performance of the PI controller. Closed loop current control bandwidth was improved from roughly 700Hz in the case of the PI simulation to 3000Hz in the case of the deadbeat simulation. Implementation results correspond closely to simulation.
- Unlike simulation, deadbeat implementation did not require current feedback compensation because there was no overshoot present. High transient damping forces, in the form of eddy currents, appear to be present in the true SRM model. This damping conveniently cancels the overshoot that might have otherwise appeared. There is little motivation to add an eddy current model and filter compensator to the deadbeat control law since the computational cost would be high and the added performance quite small.
- The deadbeat current control algorithm requires roughly 10 times more cpu cycles than the PI current control algorithm. Yet the ratio reduces to about 3 times when the entire motor control algorithm is considered.
- The effects of noise and parameter error do not cause instability in the deadbeat controller, but parameter error does cause significant current tracking error. Time-domain equations based on the current prediction and control law for law C were found to accurately predict this error.
- Noisy feedback circuitry is a weak link in the control system. The bulk of the noise disturbances must occur at frequencies greater than roughly 10X the closed loop bandwidth for good performance.

Deadbeat control design is not handled well by continuous control techniques. Continuous control design, including linear frequency design and nonlinear Lyapunov design,

applies well when sampling rates are 10 to 40 times faster than the closed loop system bandwidth. However, deadbeat control aims at bandwidths only 2-5 times higher than the control frequency. At this scale, it is much less effective to convert a continuous controller to a discrete one after designing it. Since response times are only a few control periods, small-signal linear models are inadequate, and nonlinear delay compensation is needed, it is most convenient to work in the time domain.

It is interesting to note that deadbeat stability was better than expected. This is probably because the majority of the system dynamics, namely lag-contributing linear current filter and digital delay, are easy to model precisely. A delay of a single control period accounts for the vast stability increase from control law B to control law C. The sensitivity to delay characterization suggests that it will be to the detriment of a control scheme that allows delay to vary from period to control period, even if it allows for a shorter average delay. Such might be the case for a PWM configured to begin a period immediately after the control law is computed, since computation times may vary.

This thesis has raised important questions. Can adaptive control add both flexibility and performance to an SRM deadbeat controller? What is the cause of control voltage oscillation that appears to occur for the case of inductance underestimation in Figure 10.1 and the large step response in Figure 9.5? What is the cause of significant tracking error present in the large step response? How does the deadbeat controller perform for multiple quadrant motor speed and torque control, particularly the regeneration operation? There is plenty of material for future research.

Appendix A: 6/3 SRM specifications

Number of stator poles	6
Number of rotor poles	3
Stator pole arc [deg]	36
Rotor pole arc [deg]	72
Minimum inductance, L_a, at 2A [mH]	6
Maximum inductance, L_u, at 2A [mH]	103
Phase resistance [Ω]	1.2
Air gap length [mm]	max: 0.7, min: 0.3
Number of turns [turns/phase]	110
Stack length [mm]	85
Rated voltage [VAC]	230
Rated torque [Nm]	4
Rated speed [rpm]	3,000
Rated current [A]	13
Inertia coefficient [Nm/rad/s]	0.0007
Friction coefficient [kg-m^2]	0.0003

Table A 1 6/3 SRM technical specifications (Adapted from [13])

Appendix B: Relevant C2000 F2808 microcontroller specifications and features

- High-Performance Static CMOS Technology
  - 100 MHz (10-ns Cycle Time)
- JTAG Boundary Scan Support(1)
- High-Performance 32-Bit CPU (TMS320C28x)
  - 16 x 16 and 32 x 32 MAC Operations
  - 16 x 16 Dual MAC
  - Harvard Bus Architecture
  - Atomic Operations
  - Fast Interrupt Response and Processing
  - Unified Memory Programming Model
  - Code-Efficient (in C/C++ and Assembly)
- On-Chip Memory
  - F2808: 64K X 16 Flash, 18K X 16 SARAM
  - 1K x 16 OTP ROM (Flash Devices Only)
- Boot ROM (4K x 16)
  - With Software Boot Modes (via SCI, SPI, CAN, I^2C, and Parallel I/O)
  - Standard Math Tables
- Clock and System Control
  - Dynamic PLL Ratio Changes Supported
  - On-Chip Oscillator
  - Watchdog Timer Module
- Any GPIO A Pin Can Be Connected to One of the Three External Core Interrupts
- Peripheral Interrupt Expansion (PIE) Block That Supports All 43 Peripheral Interrupts
- Three 32-Bit CPU Timers
- Enhanced Control Peripherals
  - Up to 16 PWM Outputs
  - Up to 6 HRPWM Outputs With 150 ps MEP Resolution
  - Up to Four Capture Inputs
  - Up to Two Quadrature Encoder Interfaces
- 12-Bit ADC, 16 Channels
  - 2 x 8 Channel Input Multiplexer
  - Two Sample-and-Hold
  - Single/Simultaneous Conversions
  - Fast Conversion Rate:
    160 ns - 6.25 MSPS (280x)
  - Internal or External Reference
- Advanced Emulation Features
  - Analysis and Breakpoint Functions
  - Real-Time Debug via Hardware
- Development Support Includes
  - ANSI C/C++ Compiler/Assembler/Linker
  - Code Composer Studio™ IDE
  - DSP/BIOS™
  - Digital Motor Control and Digital Power Software Libraries

Info taken from:
http://focus.ti.com/docs/prod/folders/print/tms320f2808.htm

Appendix C: TMS320C2808 SRM PI current controller function in c++

```
void current_control(void)
{
            //integral of error
            ie_int = ie_int + _IQmpyI32(_IQmpy(_IQmpyI32(_IQ(.000244141),ie),_IQ(i_tstep)),antiwindup); //divide error by 4096
            //voltage command
            vcomm = _IQmpyI32int(_IQmpy(_IQ(wc),ie_int) + _IQ(ie),kc); //return int for final answer

            //vcomm = _IQmpyI32int(_IQ(ie),9);  //P controller

            //antiwindup?
            antiwindup = (_IQabs(vcomm) <= PWM_PERIOD);

            if (vcomm >= PWM_Max){vcomm = PWM_Max-1;}
            if (vcomm <= -PWM_Max){vcomm = -PWM_Max+1;}

}
```

Appendix D: TMS320C2808 SRM motoring program with deadbeat current control in c++

```
//###########################################################################
// FILE:               deadbeat 3/6 SRM controller.c
// TITLE:              Deadbeat controller
// AUTHOR:             Ben Rudolph
// DATE:               4/2/2009
//###########################################################################
//
//=========================================================================
// Drive Specifications
//=========================================================================
//          Target Motor: Two-phase 6/3 SRM
//          Target Converter: Asymmetric
//          Rated Operation: 1800/3600 rpm, 120Vdc bus
//
//=========================================================================
//          Signal Descriptions
//=========================================================================
//          EPWM1A(GPIO0->P8.9):    PWM module for phase A
//          EPWM2A(GPIO2->P8.11):   PWM module for phase B
//          TZ1(GPIO12->P8.37):     Trip zone interrupt
//          TZ2(GPIO13->P8.17):     Trip zone interrupt
//          QEP resolution:         1024 CPR (4096 PPR)
//          QEPA (GPIO20->P8.6):    Encoder channel A
//          QEPB (GPIO21->P8.7):    Enocder channel B
//          ADCINA0(P9.2):          Phase A current
//          ADCINA1(P9.4):          Phase B current
//
//=========================================================================
//          System and Peripheral Configuration
//=========================================================================
//                      SYSCLKOUT(100MHz)/1=100MHz(10ns)
//                      High Speed peripheral Clock(HSPCLK): SYSCLKOUT(100MHz)/8=12.5MHz(80ns)
//                      PWM Mode: 40kHz, Symmetric PWM
//                      PWM Time-Base Clock (TBCLK): TBCLK = SYSCLKOUT/(HSPCLKDIV*CLKDIV)
//                      ADCLK Period: ADCLK= 80ns (12.5MHz), ADC module is clocked by HSPCLK
//                      Current Conversion Ratio: 20[A]=3[V](ADC input)=4096 (ADC digital value)
//
//=========================================================================
//          Control Strategies
//=========================================================================
//set up position and speed controller by:
//                      #define POSSPEED_DEFAULTS {0x0, 0x0,0x0,0x0,0x0,64,3,0,0x0,\
//                      12,0,30000,0,\
//                      0,0,0,\
//
// GLOBAL_Q --> 18
//###########################################################################


#include "DSP280x_Device.h"     // DSP280x Headerfile Include File
#include "DSP280x_Examples.h"   // DSP280x Examples Include File
#include "Example_posspeed.h"       // Position and Speed Measurement Include File


// Prototype statements for functions found within this file.
interrupt void adc_isr(void);
interrupt void epwm1_timer_isr(void);
interrupt void epwm1_tzint_isr(void);
interrupt void epwm2_tzint_isr(void);
void delay(void);
//void Gpio_select(void);
void InitADCConfig(void);
void InitEPwmTimer1(void);
void InitEPwmTimer2(void);
void InitEPwmTimer3(void);  // Set-up for synchronization of ADC SOC!!! DO NOT REMOVE!!!
void POSSPEED_Init(void);
void POSSPEED_Calc(POSSPEED *p);

//my current controller
void current_control(Uint16 i_ex);

//my inductance interpolation function and associated variables
int32 X_interp(_iq10 ang,_iq10 i);


Uint16 StartCount=0;
Uint16 CTRL_MODE=0;
Uint16 ia=0,ib=0,vdc=0;
Uint32 EPwm1TimerIntCount=0;
Uint32 EPwm1TZIntCount=0;
Uint32 EPwm2TZIntCount=0;
POSSPEED qep_posspeed = POSSPEED_DEFAULTS;
int16 cur_pos;
int32 cur_spd=0, cur_spd_pr=0;

// Drive System Paramters
#define PWM_PERIOD                  2500                    //fPWM  =  100Mhz/(2*PWM_PERIOD)-->PWM_PERIOD=100Mhz/(2*fPWM)//  PWM  Period  (160k->312,  80k-
>625,40k->1250,20k->2500,10k->5000)
#define PWM_10percent               500                     //10% PWM
#define PWM_Max                     PWM_PERIOD
#define     Duty_Max                PWM_Max                 // 100% of PWM_Max
#define ENC_PULSE                   1024                    // Number of encoder pulse per revolution (CPR)
int32 pwm_temp = 0; //intermediate check to make sure a zero duty cycle is not applied

//DB current control variables
Uint16 CTRL_LAW = 3;                                        //1->A, 2->B, 3->C
int32 i_comm = 167,i_pred=0;                                // ~1A
int32 vcomm=0,dcomm = 0;
int32 X_est[3]={0};                                         //stores inductance estimations
_iq10 i_index[3]={0};                                       //stores current estimations
```

```
_iq10 ang_index[3]={0};                                 //stores angle estimations
#define iadc2index                  .002994             //i_index/iadc=1/(167*2)
#define pos2index                   .043956             //360/(4096*2)
#define ticks_perT                  7                   //EQEP ticks per period, actually 4095/(60*20000), close to 7/2048, so left shift 11 afterward
#define KTR                         184                 //R*T/T_prime. KT = 306@10kHz & 153@20kHz
#define halfKTR                     92                  //KTR/2

Uint16 theta=0;                                         //wrapped cur_pos value
Uint16 LoopCount=0;
Uint16 n=0;
#define max_index                   400
int32 X0_traject[max_index] = {0};
int32 X1_traject[max_index] = {0};
int32 int32_traject[max_index] = {0};
int32 i_traject[max_index] = {0};
int32 v_traject[max_index] = {0};

//hysteresis and angle control variables
#define i_max                       2400                //~10 amps max
Uint32 n_PWMs = 0;                                      //number of PWMs counted
Uint16 angle_ex = 500;//512; //45 degrees
Uint16 angle_comm = 1050;//1251; //110 degrees

//PI speed control variables
int32 w_comm = 3600; //rpm
_iq we_int = 0, we = 0;
#define ww                          2
#define kw                          10
#define w_tstep                     .001
//speed filter
_iq w_filt=0;
#define s_k                         .01                 //speed filter gain

//tweak inductanc data
int32 L_change=1,L_offset=20000,eddy=0;

//inductance data:
//in terms of “binary inductance”, Lp. see documentation for units
const int32 X[60][10] = {{313016,301932,258365,212704,180060,156121,138145,124094,112846,103523},
                         {310664,299445,255037,210512,178493,154938,137105,123126,111913,102622},
                         {303404,292767,249702,207142,176014,152953,135299,121434,110271,101103},
                         {293544,283744,242125,201947,172167,149694,132360,118803,107948,99045.5},
                         {279952,271360,232254,194817,166713,145135,128407,115400,104957,96419.8},
                         {264774,257394,220875,186129,159769,139295,123514,111079,101148,92976.7},
                         {246680,240630,207375,175268,150941,131877,117191,105576,96289.5,88552.4},
                         {227195,222440,192842,163525,141114,123514,109951,99260.1,90542.1,83365.6},
                         {207346,203717,177634,151077,130573,114492,102115,92291.3,84256.6,77659.7},
                         {187085,184390,161540,138003,119400,104808,93583.1,84705.7,77515.5,71602.9},
                         {166764,164779,145074,124150,107497,94513.7,84575.2,76734.3,70440.2,65269.3},
                         {145893,144558,127921,110012,95414.8,84018.3,75424.7,68724,63316.5,58906.1},
                         {124885,124060,110494,95475.9,83043,73514.3,66317.5,60658.7,56139.4,52450.5},
                         {103762,103297,92564.9,80726.8,70566.4,63004,57209,52726.2,49145.9,46220.9},
                         {84543.6,84284.7,76103.2,66406.6,58441.5,52724.5,48286.2,44926.4,42216.8,40017.5},
                         {67794.9,67477.2,60457.8,53431.5,47981.8,43751.9,40583.5,38084.1,36052.8,34434.1},
                         {52681.4,52495.8,47751.1,42493.4,38691.6,35929.9,33847.7,32208.8,30963.8,29864.6},
                         {39681.6,39716,37757.1,34131.8,31668.6,29960.8,28691.7,27633.7,26794.8,26080.4},
                         {27590.6,27645.4,27475.8,26552.4,25574.8,24908.7,24279.2,23752.1,23311.5,22942.6},
                         {20657.6,20683.7,20701.8,20712,20717.3,20716.3,20707.6,20686.6,20646.5,20565.6},
                         {18788.4,18807.4,18822.6,18833.7,18841.7,18847.4,18851.5,18854.4,18856.1,18856.2},
                         {18179.6,18197,18211.4,18222,18229.6,18235.1,18239.3,18242.3,18244.5,18246.1},
                         {18335.1,18353.6,18368.7,18379.6,18387.4,18393,18397.1,18400.1,18402.3,18403.8},
                         {19241.3,19263.3,19280.3,19292.1,19300.3,19306,19309.9,19312,19310.9,19304.6},
                         {21417.4,21448.3,21468.9,21482,21489.3,21484.8,21452.9,21349.3,21176,21004.8},
                         {25821,25871.4,25897,25894.7,25778.2,25349.4,24806.3,24326.4,23863.6,23478.5},
                         {30875.6,30949.9,30980.8,30979.6,30808.8,30251.5,29165.2,28195.9,27378.2,26684},
                         {36602.4,36705.6,36742.4,36733.3,36486.5,35704.6,34188.2,32819.2,31648.5,30662.5},
                         {42946.7,43083.8,43126.9,43107,42778.5,41731.5,39938.8,38142.4,36584.8,35162.5},
                         {49813.7,49989.7,50038.7,50009.8,49614.8,48359.5,46270,43997,41972.3,40138.6},
                         {57457.3,57677.8,57730,57679.7,57155.8,55561.1,53031.9,50274.3,47832.8,45615.3},
                         {65311.2,65578.8,65631.1,65549.6,64866.1,62867.9,59931.6,56780.7,53835.6,51176.8},
                         {73418,73733.1,73778.1,73649.8,72755.4,70299.4,66838.6,63155.9,59816.8,56737.4},
                         {81571.9,81931.6,81963.2,81771.6,80620.3,77658.1,73690.3,69595.7,65704.1,62204},
                         {89766.3,90167.2,90175.5,89917.7,88509.9,85036.7,80573,75969.6,71587.7,67653.2},
                         {98183.6,98620,98592.6,98234,96482.3,92388.8,87368.8,82239.1,77399.2,72991.8},
                         {106733,107197,107119,106627,104442,99688.4,94042.5,88414.1,83073.3,78180.8},
                         {115447,115929,115778,115094,112361,106842,100510,94290.4,88421,82998.4},
                         {124024,124509,124264,123309,119918,113561,106478,99564.9,93150.7,87222.3},
                         {133306,133769,133343,131586,127013,119468,111454,103874,96956.4,90614.3},
                         {142278,142698,142040,139498,133508,124631,115616,107374,99927,93205.8},
                         {150419,150782,149918,146738,139433,129182,119120,110160,102238,95226.5},
                         {158321,158606,157497,153607,144853,133136,122084,112397,103994,96679.2},
                         {165641,165832,164399,159548,149196,136230,124294,114005,105202,97653.2},
                         {174009,174051,171690,164892,152706,138592,125902,115156,105926,98106.2},
                         {182833,182664,178834,169833,155848,140556,127166,115970,106431,98596.6},
                         {192153,191745,186799,175447,159187,142525,128379,116696,106898,98898.5},
                         {202284,201578,195385,181184,162245,144239,129427,117211,107159,98998.8},
                         {212926,211844,204024,186573,165008,145801,130255,117606,107473,98970.9},
                         {224547,222980,213066,191822,167606,147327,131096,118101,107757,99062.2},
                         {236220,234060,221753,196337,169972,148783,132016,118766,108258,99430.5},
                         {247605,244751,229713,200226,172100,150131,132985,119527,108867,99909.7},
                         {258324,254678,236560,203384,173867,151282,133881,120393,109442,100376},
                         {268279,263796,242424,206048,175464,152387,134756,121117,110027,100853},
                         {277510,272146,247392,208257,176818,153379,135567,121810,110623,101370},
                         {286157,279826,251656,210142,178095,154386,136441,122596,111342,102025},
                         {294490,287114,255329,211797,179296,155316,137400,123364,112057,102701},
                         {301043,292681,257517,212818,180074,155971,138003,123907,112565,103192},
                         {306226,296940,258782,213333,180491,156352,138355,124254,112919,103542},
                         {310039,299901,259050,213229,180442,156366,138389,124301,113002,103652}};


int32 count=0,runs=0,crap=0,overcurrent=0,runcurrent=0,cur_ctrls=0;
Uint32 t_s1=0,t_f1=0,t_s2=0,t_f2=0,X0=0,X1=0,v=0,t_diff=0,phB=0;


main()
{
            InitSysCtrl();

            EALLOW;
            SysCtrlRegs.HISPCP.all = 0x4;  // HSPCLK = SYSCLKOUT/8
            EDIS;

            InitEPwm1Gpio();
```

```
            InitEPwm2Gpio();
            InitEQep1Gpio();
            InitTzGpio();

            DINT;

            InitPieCtrl();

            IER = 0x0000;
            IFR = 0x0000;

            InitPieVectTable();

            EALLOW;  // This is needed to write to EALLOW protected registers
            PieVectTable.ADCINT = &adc_isr;
            PieVectTable.EPWM1_INT = &epwm1_timer_isr;
            PieVectTable.EPWM1_TZINT = &epwm1_tzint_isr;
            PieVectTable.EPWM2_TZINT = &epwm2_tzint_isr;
            EDIS;    // This is needed to disable write to EALLOW protected registers

            InitAdc();
            InitADCConfig();
            InitEPwmTimer1();
            InitEPwmTimer2();
            InitEPwmTimer3();

            IER |= M_INT1; // Enable CPU Interrupt 1
            PieCtrlRegs.PIEIER1.bit.INTx6 = 1;

            IER |= M_INT2;
            PieCtrlRegs.PIEIER2.bit.INTx1 = 1;
            PieCtrlRegs.PIEIER2.bit.INTx2 = 1;

            IER |= M_INT3;
            PieCtrlRegs.PIEIER3.bit.INTx1 = 1;

            EINT;          // Enable Global interrupt INTM
            ERTM;          // Enable Global realtime interrupt DBGM

            qep_posspeed.init(&qep_posspeed);

            i_comm = 1000; //align current is 6 amps
            CTRL_MODE = 1; //set this to choose which algorithm to run

            for(;;)
            {
                        asm("       NOP         ");
            }
}

interrupt void adc_isr(void)
{
            t_f2 = CpuTimer0Regs.TIM.all;
            t_diff = t_s2-t_f2;

            // Current and Voltage measurement
            ia = AdcRegs.ADCRESULT0>>4;                     // ia,  captured from ZERO SOCA of ePWM3
            ib = AdcRegs.ADCRESULT1>>4;                     // ib
            vdc = AdcRegs.ADCRESULT2>>4;        // vdc

            // Reinitialize for next ADC sequence
            AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;         // Reset SEQ1

            // Position and Speed measurement
            qep_posspeed.calc(&qep_posspeed);
            cur_pos = qep_posspeed.theta_raw;
            cur_spd = qep_posspeed.SpeedRpm_fr;
            //cur_spd_pr = qep_posspeed.SpeedRpm_pr;

            if ((ia >= i_max) || (ib >= i_max)) {CTRL_MODE=0;}

            switch (CTRL_MODE)
            {
                        case 1: //align phA rotor with 6 amps
                                    n_PWMs+=1;

                                    //assume angle and speed are zero:
                                    theta=0; cur_spd=0;
                                    current_control(ia);

                                    //phase A
                                    //must prevent writing a zero to cmpX register
                                    pwm_temp = PWM_Max*(dcomm<0)+dcomm;
                                    if(pwm_temp==0){pwm_temp=1;}
                                    EPwm1Regs.CMPA.half.CMPA = pwm_temp; //zero sign bit to get correct duty cycle
                                    pwm_temp = (PWM_Max)*(dcomm>=0);
                                    if(pwm_temp==0){pwm_temp=1;}
                                    EPwm1Regs.CMPB = pwm_temp;
                                    //phase B
                                    EPwm2Regs.CMPA.half.CMPA = 1;
                                    EPwm2Regs.CMPB = 1;

                                    if (n_PWMs >= 30000)//30000 --> 1.5 seconds
                                    {
                                                CTRL_MODE = 2;
                                                EQep1Regs.QPOSCNT = 0; //theta=0 corresponds to phA aligned position
                                                n_PWMs = 0;
                                                i_comm = 500; //set run current command
                                    }
                                    break;
                        case 2: //DB controller with speed control
                                    LoopCount++;
                                    if (LoopCount==40)//speed control
                                    {
                                                LoopCount=0;

                                                w_filt = w_filt + _IQmpy(_IQ(cur_spd) - w_filt,_IQ(s_k));
                                                we = _IQ(w_comm) - w_filt;

                                                if(we<_IQ(200))//integrate if vcomm is within possible range
                                                {
                                                            we_int = we_int + _IQmpy(_IQ(w_tstep),we);
                                                }
```

```
                                        i_comm = _IQ10mpyI32int(_IQ10mpy(_IQ10(ww),_IQtoIQ10(we_int)) + _IQtoIQ10(we),kw);
                                        if(i_comm > 2000){i_comm = 2000;}
                                        if(i_comm < 0){i_comm = 0;}
                                }

                                //wrap theta into 120 degrees(1365~=120 degrees)
                                theta = cur_pos; // cur_pos < 120
                                if (theta>=1365) {theta = theta-1365;} // 120 < curpos < 240
                                if (theta>=1365) {theta = theta-1365;} // 240 < curpos < 360

                                //phase A excitation with soft-switching
                                if ((theta>angle_ex) && (theta<angle_comm))
                                {
                                        if(phB){vcomm=0;} //reset vcomm at beginning of excite period
                                        phB=0;

                                        current_control(ia); //calculate dcomm
                                        //must prevent writing a zero to cmpX register
                                        pwm_temp = PWM_Max*(dcomm<0)+dcomm;
                                        if(pwm_temp==0){pwm_temp=1;}
                                        EPwm1Regs.CMPA.half.CMPA = pwm_temp;
                                        pwm_temp = (PWM_Max)*(dcomm>=0);
                                        if(pwm_temp==0){pwm_temp=1;}
                                        EPwm1Regs.CMPB = pwm_temp;
                                }
                                else //else commutate
                                {
                                        EPwm1Regs.CMPA.half.CMPA = 1;
                                        EPwm1Regs.CMPB = 1;
                                }

                                //phase B excitation with soft switching - phA shifted 60 degrees (682~=60 degrees)
                                if (theta>(angle_ex+682) || theta<(angle_comm-682))
                                {
                                        if(!phB){vcomm=0;} //reset vcomm at beginning of excite period
                                        phB=1;

                                        theta+=682; //theta with respect to phase B
                                        if (theta>=1365){theta-=1365;} // wrap around
                                        current_control(ib); //calculate dcomm
                                        //must prevent writing a zero to cmpX register
                                        pwm_temp = PWM_Max*(dcomm<0)+dcomm;
                                        if(pwm_temp==0){pwm_temp=1;}
                                        EPwm2Regs.CMPA.half.CMPA = pwm_temp;
                                        pwm_temp = (PWM_Max)*(dcomm>=0);
                                        if(pwm_temp==0){pwm_temp=1;}
                                        EPwm2Regs.CMPB = pwm_temp;
                                }
                                else //else commutate
                                {
                                        EPwm2Regs.CMPA.half.CMPA = 1;
                                        EPwm2Regs.CMPB = 1;
                                }
                                break;

                        case 3: //test-the-FUBAR-electronics mode

                                //phase A
                                EPwm1Regs.CMPA.half.CMPA = PWM_10percent;
                                EPwm1Regs.CMPB = PWM_Max+1;
                                //phase B
                                EPwm2Regs.CMPA.half.CMPA = PWM_10percent;
                                EPwm2Regs.CMPB = PWM_Max+1;

                                break;

                        default://turn all switches off
                                crap++;

                                //phase A
                                EPwm1Regs.CMPA.half.CMPA = 0; //high
                                EPwm1Regs.CMPB = 0; //low
                                //phase B
                                EPwm2Regs.CMPA.half.CMPA = 0; //high
                                EPwm2Regs.CMPB = 0; //low
                                break;
                }
                AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;       // Clear INT SEQ1 bit
                AdcRegs.ADCST.bit.INT_SEQ1 = 1;           // SEQ1 interrupt event occurred

                t_s2 = CpuTimer0Regs.TIM.all;

                PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge interrupt to PIE

                //t_diff = t_s2-t_f2;
}

interrupt void epwm1_timer_isr(void)
{
                EPwm1TimerIntCount++;

                // Clear INT flag for this timer
                EPwm1Regs.ETCLR.bit.INT = 1;
                // Acknowledge this interupt to receive more interrupts from group 3
                PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

interrupt void epwm1_tzint_isr(void)
{
                EPwm1TZIntCount++;

                // Acknowledge this interupt to receive more interrupts from group 2
                PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;
}

interrupt void epwm2_tzint_isr(void)
{
                EPwm2TZIntCount++;

                // Acknowledge this interupt to receive more interrupts from group 2
                PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;
}

void InitADCConfig(void)
```

```
{
            // Configure ADC
            AdcRegs.ADCTRL1.bit.CONT_RUN = 0;     // Setup Start-stop mode
            AdcRegs.ADCTRL1.bit.ACQ_PS = 2;                 // Acquisition window size
            //AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;   // Cascaded mode
            AdcRegs.ADCMAXCONV.all = 0x0003;      //only need 3 conversions on SEQ1!! but we'll do 4

            AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; // Setup ADCINA0 as 1st SEQ1 conv. => ia
            AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; // Setup ADCINA1 as 2nd SEQ1 conv. => ib
            AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x8; // Setup ADCINB0 as 3rd SEQ1 conv. => vdc

            AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 = 1;// Enable SOCA from ePWM to start SEQ1
            AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;  // Enable SEQ1 interrupt
            AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0;  // Set INT_SEQ1 at the end of every SEQ1
}

void InitEPwmTimer1(void)
{
   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
   EDIS;

// EPWM Module 1 configurations
   EPwm1Regs.TBPRD = PWM_PERIOD;                           // Period = (2*2500) TBCLK(100Mhz) counts
   EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;          // Symmetrical mode

   EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;             // Sync down-stream module
   EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;                         // Disable phase loading, Master Module
   EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
   EPwm1Regs.TBPHS.half.TBPHS = 0;

   EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;               // Select INT from Zero event
   //EPwm1Regs.ETSEL.bit.INTEN = INT_ENABLE;                       // Disable INT - no wasting CPU cycles
   EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;                     // Generate INT on 1st event

   EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
   EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
   EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; //CC_SHADOW;
   EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW; //CC_SHADOW;
   EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
   EPwm1Regs.AQCTLA.bit.CAD = AQ_SET;
   EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
   EPwm1Regs.AQCTLB.bit.CBD = AQ_SET;

   EALLOW;
   EPwm1Regs.TZSEL.bit.OSHT1 = TZ_ENABLE;                          // Enable TZ1 and TZ2 as one shot trip sources
   EPwm1Regs.TZSEL.bit.OSHT2 = TZ_ENABLE;
   EPwm1Regs.TZCTL.bit.TZA = TZ_HIZ;                                       // ePWM : High Impedance
   EPwm1Regs.TZCTL.bit.TZB = TZ_HIZ;
   EPwm1Regs.TZEINT.bit.OST = 1;                                           // Enable TZ interrupt
   EDIS;

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
   EDIS;
}

void InitEPwmTimer2(void)
{
   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
   EDIS;

// EPWM Module 2 configurations
   EPwm2Regs.TBPRD = PWM_PERIOD;                           // Period = (2*2500) TBCLK counts
   EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;          // Symmetrical mode

   EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
   EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;                          // Enable phase loading, Slave Module
   EPwm2Regs.TBCTL.bit.PRDLD = TB_SHADOW;
   EPwm2Regs.TBPHS.half.TBPHS = 0;

   EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
   EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;
   EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW; //CC_SHADOW;
   EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
   EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW; //CC_SHADOW;
   EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

   EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;
   EPwm2Regs.AQCTLA.bit.CAD = AQ_SET;
   EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;
   EPwm2Regs.AQCTLB.bit.CBD = AQ_SET;

   EALLOW;
   EPwm2Regs.TZSEL.bit.OSHT1 = TZ_ENABLE;                          // Enable TZ1 and TZ2 as one shot trip sources
   EPwm2Regs.TZSEL.bit.OSHT2 = TZ_ENABLE;
   EPwm2Regs.TZCTL.bit.TZA = TZ_HIZ;                                       // ePWM : High Impedance
   EPwm2Regs.TZCTL.bit.TZB = TZ_HIZ;
   EPwm2Regs.TZEINT.bit.OST = 1;                                           // Enable TZ interrupt
   EDIS;

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
   EDIS;
}

void InitEPwmTimer3(void)
{
   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
   EDIS;

   EPwm3Regs.TBPRD = PWM_PERIOD;
   EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;          // symmetric mode

   EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
   EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;                          // Enable phase loading, Slave Module
   EPwm3Regs.TBCTL.bit.PRDLD = TB_SHADOW;
   EPwm3Regs.TBPHS.half.TBPHS = 0;
```

```
   EPwm3Regs.ETSEL.bit.SOCAEN = SOC_ENABLE;                // Enable SOC on A group
   EPwm3Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;              // Select SOC from ZERO event on Up-count

   EPwm3Regs.ETPS.bit.SOCAPRD = ET_1ST;                    // Generate pulse on 1st event -> trigger ADC 2X per PWM cycle

   EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;
   EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
   EDIS;
}

void POSSPEED_Init(void)
{
//   EQep1Regs.QUPRD=50000;                                  // Unit Timer for 2000Hz
   EQep1Regs.QDECCTL.bit.QSRC=00;              // Quadrature-count mode

   EQep1Regs.QEPCTL.bit.FREE_SOFT=2;
   EQep1Regs.QEPCTL.bit.PCRM=01;               // Position Counter Reset on the maximum position
   EQep1Regs.QEPCTL.bit.UTE=1;                 // Unit Timeout Enabled
   EQep1Regs.QEPCTL.bit.QCLM=1;                // Latch on unit time out
   EQep1Regs.QPOSMAX=4*ENC_PULSE - 1;          // 4*CPR-1
   EQep1Regs.QEPCTL.bit.QPEN=1;                // QEP enable

   EQep1Regs.QCAPCTL.bit.UPPS=5;               // 1/32 for unit position
   EQep1Regs.QCAPCTL.bit.CCPS=6;               // 1/64 for CAP clock (prescaler)
   EQep1Regs.QCAPCTL.bit.CEN=1;                // QEP Capture Enable
           //not in KSH's code:
   //EQep1Regs.QDECCTL.bit.SWAP=1;             // Swap QEP inputs (same results as swapping A & B encoder signals

}

void POSSPEED_Calc(POSSPEED *p)
{
     long tmp;
     unsigned int pos16bval,temp1;
              _iq Tmp1,newp,oldp;

     p->DirectionQep = EQep1Regs.QEPSTS.bit.QDF;

            pos16bval=(unsigned int)EQep1Regs.QPOSCNT;
     p->theta_raw = pos16bval+ p->cal_angle;

//**** High Speed Calculation using QEP Position counter ****//

            if(LoopCount==39)
            {
            tmp = (long)((long)pos16bval*(long)p->mech_scaler);         // Q0*Q18 = Q18
                        newp=tmp;
                        oldp=p->oldpos;

                        if (p->DirectionQep==0)                                                 // POSCNT is counting down
                        {
                        if (newp>oldp)
                                    Tmp1 = - (_IQ(1) - newp + oldp);
                        else
                        Tmp1 = newp - oldp;
                        }
                        else if (p->DirectionQep==1)                                // POSCNT is counting up
                        {
                        if (newp<oldp)
                        Tmp1 = _IQ(1) + newp - oldp;
                        else
                        Tmp1 = newp - oldp;
                        }

                        if (Tmp1>_IQ(1))
                        p->Speed_fr = _IQ(1);
                        else if (Tmp1<_IQ(-1))
                        p->Speed_fr = _IQ(-1);
                        else
                        p->Speed_fr = Tmp1;

                        // Update the electrical angle
                        p->oldpos = newp;

                        // Change motor speed from pu value to rpm value (Q15 -> Q0)
                        // Q0 = Q0*GLOBAL_Q => _IQXmpy(), X = GLOBAL_Q
                        p->SpeedRpm_fr = _IQmpy(p->BaseRpm,p->Speed_fr);
                        //=======================================
            }

// Low-speed computation using QEP capture counter //
            if(EQep1Regs.QEPSTS.bit.UPEVNT==1)
            {
                        if(EQep1Regs.QEPSTS.bit.COEF==0) // No Capture overflow
                                    temp1=(unsigned long)EQep1Regs.QCPRDLAT;
                        else                                                                                                    // Capture overflow, saturate the result
                                    temp1=0xFFFF;
                                    p->Speed_pr = _IQdiv(p->SpeedScaler,temp1);
                                    Tmp1=p->Speed_pr;
                        if (Tmp1>_IQ(1))
                                    p->Speed_pr = _IQ(1);
                        else
                                    p->Speed_pr = Tmp1;
                        if (p->DirectionQep==0)
                                    p->SpeedRpm_pr = -_IQmpy(p->BaseRpm,p->Speed_pr);           // Q0 = Q0*GLOBAL_Q => _IQXmpy(), X = GLOBAL_Q
                        else
                                    p->SpeedRpm_pr = _IQmpy(p->BaseRpm,p->Speed_pr);            // Q0 = Q0*GLOBAL_Q => _IQXmpy(), X = GLOBAL_Q

                        EQep1Regs.QEPSTS.all=0x88;                                                                              // Clear Unit position event flag

            // Clear overflow error flag
            }
}

void current_control(Uint16 i_ex)
{
            //calculate current, angle and inductance estimates
            i_index[0] = _IQtoIQ10(_IQmpyI32(_IQ(iadc2index),i_ex));
            ang_index[0] = _IQtoIQ10(_IQmpyI32(_IQ(pos2index),theta));
            X_est[0] = X_interp(ang_index[0],i_index[0]);  //rotor alignment->angle~=0
```

```
            i_index[1] = _IQtoIQ10(_IQmpyI32(_IQ(iadc2index),i_comm)); //this need be calculated only after command is changed
            ang_index[1] = _IQtoIQ10(_IQmpyI32(_IQ(pos2index),theta + (ticks_perT*cur_spd>>11)));
            //if(ang_index[1]>=_IQ10(60)){ang_index[1]-=_IQ10(60);}//make sure we don't exceed angle index limits:
            X_est[1] = X_interp(ang_index[1],i_index[1]);  //rotor alignment->angle

            switch (CTRL_LAW)
            {
                        case 1:                 //control law A
                                    dcomm = ((KTR+X_est[1]-(X_est[0]<<1))*i_ex + X_est[0]*i_comm)/vdc;//A
                                    break;
                        case 2:                 //control law B
                                    dcomm = ((halfKTR-X_est[0])*i_ex + (halfKTR+X_est[1])*i_comm)/vdc;//B
                                    break;
                        case 3:                 //control law C

                                    //calculate new angles and X's for k=2
                                    ang_index[2] = _IQtoIQ10(_IQmpyI32(_IQ(pos2index),theta + (ticks_perT*cur_spd>>10))); //multiply angle offset by two,
or only left shift 10
                                    //if(ang_index[2]>=_IQ10(60)){ang_index[2]-=_IQ10(60);}//make sure we don't exceed angle index limits:
                                    X_est[2] = X_interp(ang_index[2],i_index[1]);//present i_index[1] is i_comm

                                    //predict current i[K+1]

                                    i_pred = (i_ex*(X_est[0]-halfKTR)+vcomm)/(X_est[1]+halfKTR);

//eliminated to reduce comp time to make 20Khz operation possible:
                                    //calculate better X[k+1] from current prediction
                                    //i_index[1] = _IQtoIQ10(_IQmpyI32(_IQ(iadc2index),i_pred));
                                    //X_est[1] = X_interp(ang_index[1],i_index[1]);

                                    vcomm = (halfKTR-X_est[1])*i_pred + (halfKTR+X_est[2])*i_comm;//C
                                    //vcomm = vcomm + (i_comm-i_pred)*eddy; //'eddy current' p compensation
                                    dcomm = vcomm/vdc;

                                    break;
            }

            //control limiting:
            if (dcomm > PWM_Max)//don't go higher than limit
            {
                        dcomm = PWM_Max;
                        vcomm = PWM_Max*((long) vdc);
            }
            else if(dcomm < -PWM_Max)//don't go lower than limit
            {
                        dcomm = -PWM_Max;
                        vcomm = -PWM_Max*((long) vdc);
            }

            if (CTRL_MODE==2) //run mode has begun
            {
                        count++;
                        if ((count==400) && (n < max_index))//take data after 5 seconds
                        {
                                    count=0;
                                    if(phB){i_pred=0;} //don't want i_pred for phB
                                    v_traject[n] = t_diff;
                                    int32_traject[n] = i_pred;
                                    X0_traject[n] = dcomm;
                                    X1_traject[n] = i_comm;
                                    i_traject[n] = ia;
                                    n++;
                        }
            }
}

int32 X_interp(_iq10 ang,_iq10 i)//2D interpolation algorithm
{
            // i and ang are scaled such that thier integer part correspond to
            // their index in the 10X60 inductance array
            static _iq10 d_ang;
            static int32 i_low,ang_low,f1,f2,f3,f4,y1,y2,f;

            if (i > _IQ10(9)) {i=_IQ10(9);} //i-index must not be exceeded
            i_low = _IQ10int(i); //convert to closest even number less than i

            ang_low = _IQ10int(ang); //this is also the index of the angle

            //test 50% higher inductance data
            f1 = X[ang_low][i_low];
            f2 = X[ang_low][i_low+1];
            f3 = X[ang_low+1][i_low+1];
            f4 = X[ang_low+1][i_low];

            d_ang = ang - _IQ10(ang_low);
            y1 = _IQ10mpyI32(d_ang,f4 - f1);
            y2 = _IQ10mpyI32(d_ang,f3 - f2);

            f = f1 + _IQ10int(y1 + _IQ10mpy(_IQ10(f2-f1)+y2-y1,i-_IQ10(i_low)));

            //if(L_change>0){f = f + (f>>L_change);}
            //else if(L_change<0){f = f - (f>>abs(L_change));}

            //f = (f-18000) + ((f-18000)>>L_change) + L_offset; //inductance sensitivity analysis

            return f;

}
```

Appendix E: Estimating current error and voltage command error attributed to feedback noise

Current feedback noise may be modeled with an additive white noise, $\hat{\imath} = i + N_k$, where N_k has a zero average, and varies randomly with time(k) between -A and A, where A is constant. The current feedback signal produces an inductance estimate, $\hat{L} = L + dL$, which can be expressed as $\hat{L} = L + \frac{dL}{di}N_k$ if the current noise is small with respect to the current. In terms of a reactance estimate, $\hat{X}_f$, the current trajectory may be calculated by substituting the voltage command into (10.3):

$$i[k+2] \cong \frac{-v^*[k] - \hat{\imath}[k]\hat{X}[k] + i^*[k+2]X[k+2] + v^*[k] + i[k]X[k]}{X[k+2]}$$

$$i[k+2] \cong i^*[k+2] - \frac{\hat{\imath}[k]\hat{X}[k] - i[k]X[k]}{X[k+2]}$$

Then, after factoring and cancelling the period, T, out of reactance, gives the current error in terms of inductance:

$$i_e \cong \frac{\hat{\imath}[k]\hat{L}[k] - i[k]L[k]}{L[k+2]}$$

If the current error is small, the error estimate may be simplified further:

$$i_e \cong \frac{(i[k] + N_k)\left(L[k] + \frac{dL}{di}[k]N_k\right) - i[k]L[k]}{L[k+2]} = \frac{N_k\left(L[k] + i[k]\frac{dL}{di}[k]\right) + \frac{dL}{di}[k]{N_k}^2}{L[k+2]}$$

Since current noise is small with respect to the current the quadratic term $\frac{dL}{di}{N_k}^2 \cong 0$. Furthermore, $L[k] + i[k]\frac{dL}{di}[k]$, is the expression for incremental inductance, $\frac{\partial\lambda}{\partial i}$, which is greater than zero. The final expression for phase current error in terms of current feedback error is:

$$i_e \cong \frac{N_k\frac{\partial\lambda}{\partial i}[k]}{L[k+2]}$$

The voltage command error, $v_e{}^*$, resulting from a current feedback error is found by subtracting the voltage command, $\hat{v}^*$, computed by $\hat{\imath}$ from the voltage command, v^*, computed by i.

$$v_e{}^* \cong v^*[k+1] - \hat{v}^*[k+1]$$

$$v_e{}^* \cong -v^*[k] - i[k]X[k] + i^*[k+2]X[k+2] - \left(-v^*[k] - \hat{\imath}[k]\hat{X}[k] + i^*[k+2]X[k+2]\right)$$

$$v_e{}^* \cong \hat{\imath}[k]\hat{X}[k] - i[k]X[k]$$

Expressing reactance in terms of inductance and normalizing the voltage command to the capacitor voltage gives the effective duty cycle command(not a true duty cycle, since it ranges from -1 to 1):

$$d_e{}^* \cong \frac{\hat{\imath}[k]\hat{L}[k] - i[k]L[k]}{TV_C}$$

For a small feedback, the normalized voltage command error may be expressed as:

$$d_e{}^* \cong \frac{N_k \frac{\partial \lambda}{\partial i}[k]}{TV_C}$$

References

[1] R. Krishnan, *Switched reluctance motor drives : modeling, simulation, analysis, design, and applications*. Boca Raton, Fla.: CRC Press, 2001.

[2] L. B. Amor, O. Akhrif, L. A. Dessaint, and G. Olivier, "Adaptive Nonlinear Torque Control of a Switched Reluctance Motor," in *American Control Conference, 1993*, 1993, pp. 2831-2836.

[3] L. Szamel, "Convergence test of model reference parameter adaptive SRM drives," in *Power Electronics and Applications, 2005 European Conference on*, 2005, pp. 10 pp.-P.10.

[4] S. A. Bortoff, R. R. Kohan, and R. Milman, "Adaptive control of variable reluctance motors: a spline function approach," *Industrial Electronics, IEEE Transactions on,* vol. 45, pp. 433-444, 1998.

[5] N. C. Sahoo, J. X. Xu, and S. K. Panda, "Low torque ripple control of switched reluctance motors using iterative learning," *Energy Conversion, IEEE Transactions on,* vol. 16, pp. 318-326, 2001.

[6] S. Buso, S. Fasolo, and P. Mattavelli, "Uninterruptible power supply multiloop control employing digital predictive voltage and current regulators," *Industry Applications, IEEE Transactions on,* vol. 37, pp. 1846-1854, 2001.

[7] D. G. Holmes and D. A. Martin, "Implementation of a direct digital predictive current controller for single and three phase voltage source inverters," in *Industry Applications Conference, 1996. Thirty-First IAS Annual Meeting, IAS '96., Conference Record of the 1996 IEEE*, 1996, pp. 906-913 vol.2.

[8] C. Jun-Seok, L. Seung-Yo, M. Hyung-Soo, and C. Gyu-Ha, "Analysis and design of modified deadbeat controller for 3-phase uninterruptible power supply," in *Power Electronics and Drive Systems, 1999. PEDS '99. Proceedings of the IEEE 1999 International Conference on*, 1999, pp. 1003-1009 vol.2.

[9] Y. A. R. I. Mohamed and E. F. El-Saadany, "Robust High Bandwidth Discrete-Time Predictive Current Control with Predictive Internal Model—A Unified Approach for Voltage-Source PWM Converters," *Power Electronics, IEEE Transactions on,* vol. 23, pp. 126-136, 2008.

[10] L. Springob and J. Holtz, "High-bandwidth current control for torque-ripple compensation in PM synchronous machines," *Industrial Electronics, IEEE Transactions on,* vol. 45, pp. 713-721, 1998.

[11] S.-G. Oh and R. Krishnan, "Two-Phase SRM With Flux-Reversal-Free Stator: Concept, Analysis, Design, and Experimental Verification," *Industry Applications, IEEE Transactions on,* vol. 43, pp. 1247-1257, 2007.

[12] M. Ehsani, I. Husain, and A. B. Kulkarni, "Elimination of discrete position sensor and current sensor in switched reluctance motor drives," *Industry Applications, IEEE Transactions on,* vol. 28, pp. 128-135, 1992.

[13] K. Ha, "Position Estimation in Switched Reluctance Motor Drives Using the First Switching Harmonics of Phase Voltage and Current," Blacksburg, Va.: Virginia Polytechnic Institute & State University, 2008.

[14] D. A. Torrey and J. H. Lang, "Modelling a nonlinear variable-reluctance motor drive," *Electric Power Applications, IEE Proceedings B,* vol. 137, pp. 314-326, 1990.

[15] H. Zhou, W. Ding, and Z. Yu, "A nonlinear model for the switched reluctance motor," in *Electrical Machines and Systems, 2005. ICEMS 2005. Proceedings of the Eighth International Conference on*, 2005, pp. 568-571 Vol. 1.

[16] H. Hannoun, M. Hilairet, and C. Marchand, "Analytical modeling of switched reluctance machines including saturation," in *Electric Machines & Drives Conference, 2007. IEMDC '07. IEEE International*, 2007, pp. 564-568.

[17] S. A. Hossain and I. Husain, "A geometry based simplified analytical model of switched reluctance machines for real-time controller implementation," *Power Electronics, IEEE Transactions on,* vol. 18, pp. 1384-1389, 2003.

[18] N. Radimov, N. Ben-Hail, and R. Rabinovici, "Inductance measurements in switched reluctance machines," *Magnetics, IEEE Transactions on,* vol. 41, pp. 1296-1299, 2005.

[19] V. K. Sharma, S. S. Murthy, and B. Singh, "An improved method for the determination of saturation characteristics of switched reluctance motors," *Instrumentation and Measurement, IEEE Transactions on,* vol. 48, pp. 995-1000, 1999.

[20] J. Jae-Woo, L. Sang-Ho, H. Jung-Pyo, K. Ki-Nam, C. Hyoung-Jun, and M. Sang-Hoon, "Optimum design for eddy current reduction in permanent magnet to prevent irreversible demagnetization," in *Electrical Machines and Systems, 2007. ICEMS. International Conference on*, 2007, pp. 949-954.

[21] G. Franklin, D. Powell, and M. Workman, *Digital Control of Dynamic Systems*, 3 ed.: Ellis-Kagle Press, 1998.

[22] S. h. Han, "The Properties of a Deadbeat Control," *Proceedings of the 11th CISL Winter Workshop, Seorak Mountain,* 1998.

[23] C. Dyer and P. Ip, "An Elementary Introduction to Scientific Computing," 2000.

[24] N. S. Nise, *Control Systems Engineering*, 4 ed.: John Wiley and Sons, 2003.