# ORIGIN AND REDUCTION OF ACOUSTIC NOISE IN VARIABLE-RELUCTANCE MOTORS

by

DERRICK E. CAMERON

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements
for the degrees of

Bachelor of Science

and

Master of Science in Electrical Engineering

at the

Massachusetts Institute of Technology
January 1990

Signature of Author ______________________________
Department of Electrical Engineering and Computer Science
January 23, 1990

Certified by ______________________________
Jeffrey H. Lang
ıgineering and Computer Science
Thesis Co-supervisor

Certified by ______________________________
Stephen D. Umans
Principal Research ~~Engineer~~, Electrical Engineering and Computer Science
[illegible] supervisor

Accepted by ______________________________
Arthur C. Smith, Chairman
Departmental Graduate Committee
Electrical Engineering and Computer Science

# ORIGIN AND REDUCTION OF ACOUSTIC NOISE IN VARIABLE-RELUCTANCE MOTORS

by

DERRICK E. CAMERON

Submitted to the Department of Electrical Engineering and Computer Science on January 23, 1990 in partial fulfillment of the requirements for the degrees of Bachelor of Science and Master of Science in Electrical Engineering

## ABSTRACT

This thesis investigates the origin of acoustic noise in the variable-reluctance motor (VRM) and discusses issues involved in predicting and reducing it at the design stage of the motor. A discussion of the physical structure and operating principles of the VRM is given as a prelude to the presentation of a list of hypothetical noise generating mechanisms in the VRM. An experimental procedure for determining which of these mechanisms is dominant in the VRM is then described.

The experimental procedure is performed on an experimental 1/2-hp VRM and shows that radial vibrations of the stator driven by radial magnetic forces in the VRM are responsible for producing the majority of the acoustic noise. In particular, the most noise is produced when harmonics of the radial forces coincide with any of the mechanical resonant frequencies of the stator. The experimental VRM is found to have resonant frequencies at 2604 Hz, 9200 Hz, and 14.2 kHz. The experiments also reveal that it is possible to reduce the acoustic noise level at a given speed by correctly choosing the control strategy. The problem of reducing the acoustic noise, therefore, involves developing models for determining the resonant frequencies and mode shapes of the stator and the radial magnetic forces acting on it. With such models, the acoustic noise can be predicted as a function of the stator geometry, the electromagnetic behavior of the motor and inverter, and the control strategy, allowing the control strategy that minimizes the acoustic noise to be determined.

The model for determining the resonant frequencies and mode shapes of the stator is based on the three dimensional theory of elasticity and uses the Rayleigh-Ritz method and Lagrange's equation to derive the frequency equation of the stator cylinder. This equation takes the form of a generalized eigenvalue problem where the resonant frequencies of the stator are the square roots of the eigenvalues, and the mode shapes are determined from the eigenvectors. While the model correctly predicts the mode shapes of the stator, it is not detailed enough to accurately predict the values of the resonant frequencies. Thus, while the model is not useful for determining the optimum control strategy for reducing the acoustic noise, it is useful for determining whether the resonant frequencies are in the audible range and how they change as the geometry of the stator is changed.

The model for determining the magnetic forces is comprised of equations that govern the operation of the VRM at constant speed and equations that govern the operation of the inverter and controller. The current-flux relationship is modeled by an analytic function determined from the experimental VRM. Numerical simulations of these equations show that while the current and torque of the VRM can be predicted accurately enough to determine quantities such as the peak current and average torque, the radial force cannot be predicted accurately enough to determine the acoustic noise. This is due to the requirement that harmonics of the radial force as high as the 7th must be accurately predicted.

Thesis Supervisors: Dr. Jeffrey H. Lang
Dr. Stephen D. Umans
Titles: Associate Professor of Electrical Engineering
Principal Research Engineer

## ACKNOWLEDGEMENTS

I would like first to thank my thesis co-advisors Prof. Jeffrey Lang and Dr. Stephen Umans for their friendship, support, and guidance. Throughout the course of this research, they both contributed useful insights which helped me to keep going even when I thought progress was impossible. I am grateful for their all their patience and encouragement.

I would also like thank to my office-mates Steve Leeb, Brett Miwa, and Ray Sepe for all of their help and friendly encouragement. Our many discussions were not only technically informative, but at times, offered comic relief from the everyday routine at MIT.

There are several other people whose assistance was invaluable to me. I would like to thank John Apostolopoulos for performing many of the initial experiments of this project. I would also like to thank Anthony Caloggero for constructing the acoustic chamber used in this thesis. I especially appreciate the assistance he gave me in the machine shop. Many of the problems associated with the logistical aspects of this thesis were solved with the assistance of Sharon Leah Brown and Barbara Smith. I am grateful the extra effort they put in for me on several occasions.

A special thanks goes to my parents for their love, support and constant encouragement.

This thesis was supported by the Research and Development Center of the General Electric Company under a research grant dated November 18, 1987. I am grateful for the motors that the company donated to us, and for the assistance that Dr. Gerald Kliman and Dr. Thomas Jahns gave throughout the duration of the project. I would also like to thank Dr. Jim Wilson for his suggestion concerning the use of random noise to reduce the acoustic noise of the VRM, which is described in Chapter 6.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The variable-reluctance motor has become increasingly popular in recent years. It has been shown to have comparable performance to induction and permanent-magnet-synchronous motors in terms of efficiency, torque per volume, and inverter power rating over a reasonable power range [22,21,16]. In addition, it is relatively simple and inexpensive to construct, is particularly reliable, and provides a wide range of desirable operating characteristics through the use of different control algorithms. This makes it desirable in many variable-speed applications [16,2].

Most of the literature on VRM drives has concentrated mainly on the design of the motor, inverter, and controller with the explicit goal of improving control, efficiency, torque production, drive flexibility and cost [21,2,22,15]. As improvements have been achieved, there has been increasing incentive to use VRMs in domestic products that have strict acoustic noise emission standards. Although the generation of acoustic noise is a characteristic of all electric motors, it seems to be especially problematic in variable-reluctance motors. This is a serious problem, because many experts recognize that, depending on the level and duration, acoustic noise can be an irritant or even a threat to a person's hearing. Federal, state, and local agencies, therefore, have established standards stating how much

| Maximum daily exposure (hours) | Sound level (dB) A-weighting |
|---|---|
| 8.0 | 90 |
| 6.0 | 92 |
| 4.0 | 95 |
| 3.0 | 97 |
| 2.0 | 100 |
| 1.5 | 102 |
| 1.0 | 105 |
| 0.5 | 110 |
| 0.25 or less | 115 max |

Table 1.1: Permissible noise exposure.

noise is acceptable. One such standard, which is part of the Walsh-Healey Public Contracts Act of May 1969 [5], is given in Table 1.1. The noise of an experimental VRM used in this thesis is well above 90 dB even at low current levels when measured at a distance of 20 cm.

Because it can be a threat to a person's health, or at least a sufficient irritant, the acoustic noise of a VRM has the potential to offset the advantages of a VRM drive over an induction motor drive. Therefore, it has become clear at this time that attention must be focused upon integrating acoustic noise considerations into the overall design process for a VRM, so that appropriate tradeoffs can be made between the traditional performance characteristics such as cost, size, rating, and efficiency, and the acoustic noise produced by the VRM. In order to do this, however, the acoustic noise generating mechanisms in the VRM must be understood and modeled. Each of these mechanisms consists of an excitation force of either magnetic or mechanical origin that acts on some part of the motor, resulting in vibrations which produce the acoustic noise.

## 1.1 Background

Previous work on the acoustic noise problem of electric motors has focused almost exclusively on the task of determining the resonant frequencies and mode shapes of the stators of induction motors. Most authors have determined that the acoustic noise of these motors is caused by magnetic forces exciting the resonant frequencies of the stator [27,10], making the knowledge of these frequencies essential to the reduction of the acoustic noise. The modeling of the stator and the analytical method used, however, have varied widely. The earlier papers, cited in [27], treated the stator as a thin shell, which greatly simplified the analysis, but gave poor results.

Verma and Girgis [27] modeled the stator as a thick cylinder encased by a thin cylindrical frame. Unlike earlier papers, the analysis of the stator was based on the three-dimensional theory of elasticity. The exact equations of motion were used, and radial, torsional, and axial vibrations of the stator were taken into consideration. Flügge's theory of thin-shells [7] was used for the frame. The surfaces of the stator were assumed to be traction-free. This analysis yielded reasonably accurate resonant frequencies for the lower modes when a correction factor was used to account for the stator teeth and the windings. However, this correction factor was not based on rigorous considerations, and the authors concluded that the effects of the teeth and windings should be considered rigorously along with the stator in order to attain more accurate results.

Girgis and Verma [10] also extended their earlier modeling of the stator as a thick cylinder encased by a thin cylindrical frame. In the extension, their analysis treated the stator teeth, windings, and cooling ribs as thin cantilevers attached to the stator cylinder.

In [27], the equations of motion were derived by satisfying the elastic equilibrium and compatibility conditions. In this case, however, the presence of the teeth, windings, and cooling ribs made it difficult to satisfy the boundary conditions along with the equilibrium and compatibility conditions. Therefore, the authors resorted to using the Rayleigh-Ritz method [13]. For complete confidence in the results, however, this method also requires that attention be paid to satisfying the boundary conditions. This is due to the fact that the resonant frequencies of a given structure under one set of boundary conditions can be entirely different from those of the same structure under a different set of conditions. The authors, however, did not address the issue of the boundary conditions. In spite of this, the resonant frequencies obtained from their analysis showed satisfactory agreement with those obtained from the experimental measurements made on an induction motor.

## 1.2 Thesis Scope

This thesis addresses several major issues concerning the origin and reduction of acoustic noise in VRMs. First, the relative importance of the acoustic noise generating mechanisms is examined. Although it is often stated that vibrations of the stator induced by radial magnetic forces is the dominant mechanism in induction motors, this need not be the case for all VRMs. An experimental procedure is presented for determining the relative importance of several hypothetical noise mechanisms. This procedure is performed on an experimental 1/2-hp VRM provided by the General Electric Company of Schenectady, NY. It shows that the dominant mechanism is indeed radial vibrations of the stator induced by pulsating magnetic forces. Specifically, the magnitudes of the vibrations are largest when harmonics of the radial force pulses coincide with the resonant frequencies of the stator;

the radial force pulses occur at the same frequencies as the current pulses applied to the phases of a VRM. The acoustic noise emitted by the stator is found to be proportional to the amplitude of these vibrations at each resonant frequency. The experimental procedure also shows, however, that there are significant vibrations of other parts of the VRM at each resonant frequency. Depending on how the VRM is mounted and loaded, these vibrations could potentially lead to additional noise problems. The main conclusion drawn from the experiments is that the shape of the radial force pulses, which is determined in part by the shape of the current pulses, has a direct effect on the amount of acoustic noise produced.

The second issue addressed by this thesis concerns the determination of the vibrational behavior of the stator as a function of the stator geometry and material properties. As stated above, the acoustic noise of the experimental VRM results from the interaction of harmonic components of the radial forces and the resonant frequencies of the stator. In order to predict the acoustic noise, therefore, a method is needed for determining the resonant frequencies and mode shapes of the stator. Knowledge of the mode shapes allows the elimination from consideration of those resonant frequencies whose corresponding mode shapes do not couple to the magnetic forces in the VRM.

The modeling performed and analysis used in this thesis is based on that in [10], but much closer attention is paid to satisfying the boundary conditions. Because frictional losses are not modeled in the analysis, only the resonant frequencies and shapes of the stator vibration modes can be predicted. Neither the amplitudes of the vibrations nor the width of the resonant peaks can be predicted. Because of these limitations, the analysis is mainly useful for determining whether the acoustic noise is in the audible range and the effect of changes in the geometry of the stator on its frequencies.

When applied to the experimental VRM, the analysis predicts the three major resonant frequencies along with their corresponding mode shapes as observed in the experiments. The values of the resonant frequencies, however, are significantly in error. The errors are attributed to the simplifications made in the analysis. The analysis also shows that attempting to raise the resonant frequencies above the audible range by changing the length or thickness of the stator cylinder is not very cost effective. Doubling the thickness of the stator cylinder only increases the lowest resonant frequency by a factor of 1.5, meaning that the lowest resonant frequency of the experimental VRM would increase from its original value of 2604 Hz to 3906 Hz. Changing the length of the stator has a much smaller effect than this on the resonant frequencies. Consequently, alternative methods must be used to achieve acoustic noise reduction.

The third issue addressed by this thesis concerns the determination of the radial magnetic forces acting on the stator. In the context of acoustic noise prediction/reduction, this issue has not received as much attention as the previous one even though, as the experiments performed on the experimental VRM show, it is clearly as important; this issue is addressed by Yang in [29] where he determined the magnetic forces acting on the stator of an induction motor. In fact, due to the difficulty in raising the resonant frequencies above the audible range by changing the geometry of the stator, this issue takes on even greater importance, since it offers an alternative method of reducing the acoustic noise, which involves neither mechanical nor acoustic dampers, but rather shaping of the radial force pulses so that none of the harmonics coincide with any of the resonant frequencies of the stator.

In this thesis, a model for the VRM is developed which predicts its currents and fluxes based on the inverter topology, the nonlinear current-flux-position relationship, and the control scheme. The current-flux-position relationship is modeled by an analytic expression derived from the actual current-flux-position curves of the experimental VRM. The magnetic fields in the air gap are approximated as being radially directed and the gap length is assumed to be independent of the rotor position. Simulations of the model show that while the currents, fluxes, and torque can be predicted to a sufficient degree of accuracy, the radial forces acting on the stator cannot. The main reason for this is that harmonic components of the radial force pulses, as high as the seventh in the case of the experimental VRM, must be accurately predicted. To be able to do this, a more detailed model for calculating the magnetic fields in the air gap at all angular positions of the rotor is needed.

## 1.3 Thesis Outline

The outline for the remainder of this thesis is as follows. Chapter 2 discusses the VRM drive which includes the motor, inverter, and controller. Chapter 3 describes the experimental procedure used to determine the relative importance of the acoustic noise generating mechanisms in the experimental VRM. Chapter 4 describes the analysis used to determine the resonant frequencies and modes of the stator. The effect of variations in the geometry of the stator on the resonant frequencies is illustrated. Chapter 5 presents an operational model of the VRM which allows the magnetic forces to be predicted based on parameters which define the control scheme used to operate the VRM. Finally, Chapter 6 contains the summary, conclusion, and suggestions for future work.

# Chapter 2

# VRM Drive Principles

## 2.1 Introduction

In order to establish a foundation for examining the issues addressed in this thesis, this chapter discusses the VRM drive which consists of a motor, inverter, and controller. The first section describes the construction of the experimental 0.5-hp VRM used in this thesis. The second section describes the basic operating principles of the VRM drive, and details the parameters which define its operation.

## 2.2 VRM Construction

As noted in Chapter 1, one of the advantages of a VRM is its simple construction. In Figure 2.1, the cross section of a typical four-phase VRM is shown. The rotor is constructed from simple iron laminations having six salient poles. It has neither windings nor permanent magnets. The stator is comprised of iron laminations having eight salient poles. Each of the poles has a concentrated copper winding. Windings on diametrically opposite poles are connected in series to form a phase winding; only one phase winding is shown in Figure 2.1 for clarity. When current is applied to a phase winding, the stator and rotor are both magnetized with apposing polarity across the air gap. This produces a force having a radial and a tangential component. The radial component of the force causes the radial deflections of the stator which, as Chapter 3 shows, is the main source of acoustic noise of the experimental VRM. The tangential component of the force acting on the rotor produces a torque which causes the rotor to align itself with the magnetized stator pole pair, regardless of the polarity of the current. Continuous rotation of the rotor is achieved by sequentially exciting the phases with pulses of current. The pulses must be timed so that the torque always acts to pull the rotor in the same direction. Consequentially, closed-loop control of the motor is necessary for good performance.

The actual rotor and stator laminations for the experimental VRM are shown in Figure 2.2. The stator laminations are supported by two endbells. The endbells are held together by four bolts. The rotor laminations are mounted on a steel shaft. Also mounted on the shaft on either side of the rotor stack are bearing assemblies. These assemblies are designed to fit into holes bored in the endbells. Ideally, when the motor is assembled, the rotor and stator lamination stacks should be aligned and the gap separation between all rotor and stator poles should be the same.

## 2.3 Principles of Operation

### 2.3.1 VRM Dynamics

The analysis in this section assumes that the phases are identical and magnetically independent. With these assumptions, the general operation of a VRM can be described by the following equations:

$$\frac{d\lambda_n}{dt} = -R_n i_n + v_n, \qquad n = 1, \ldots, N_p \tag{2.1}$$

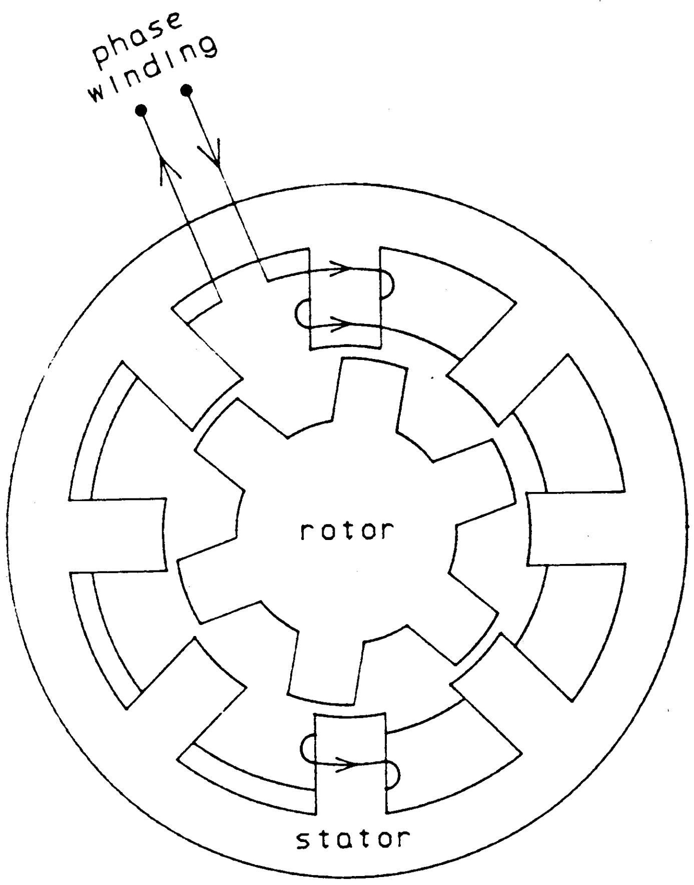

Figure 2.1: Cross section of a 4-phase VRM.

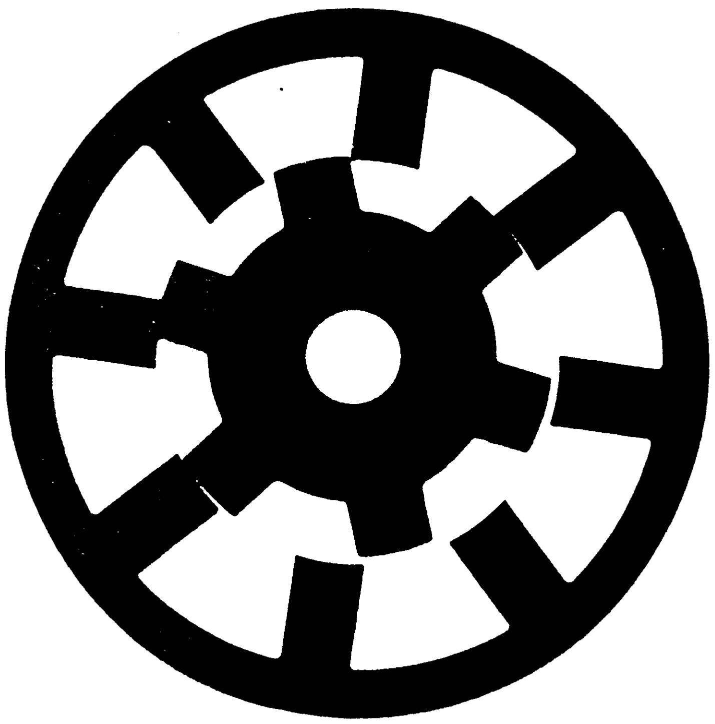

Figure 2.2: Experimental VRM laminations. The depth of the lamination stack is 2 inches.

$$\frac{d\theta}{dt} = \omega_r \tag{2.2}$$

$$J\frac{d\omega_r}{dt} = \tau_m - B_r\omega_r - \tau_f\frac{\omega_r}{|\omega_r|} - \tau_l \tag{2.3}$$

where $\lambda_n$, $i_n$, and $v_n$ are the flux linkage, current, and applied voltage of the nth phase winding, respectively, $R_n$ is the total resistance of the circuit that includes the nth phase winding, $\theta$ is the rotor position, $\omega_r$ is the rotor speed, $J$ is the total rotor and load inertia, $N_p$ is the number of phases, $\tau_m$ is the magnetic torque, $B_r$ and $\tau_f$ are the coefficients of viscous and coulomb friction, respectively, and $\tau_l$ is the load torque [6,14]. Under the assumptions that the phases are symmetrically located and the rotor and stator have very high permeabilities, the magnetic coupling between the phases can be neglected so that

$$\lambda_n = \lambda_n(\theta, i), \qquad n = 1, \ldots, N_p \tag{2.4}$$

In general, $\lambda_n$ is a nonlinear function of $i_n$ due to magnetic saturation. In addition, $\lambda_n$ varies periodically with the rotor position $\theta$. The period is equal to $2\pi/N_r$ where $N_r$ is the number of rotor poles. The flux-current relationship is identical for all phases except for a constant offset angle which takes into account the spatial location of the phases. The flux-current relationship for the nth phase can therefore be written as

$$\lambda_n(\theta, i_n) = \lambda_1(\theta_n, i_n) \qquad n = 1, \ldots, N_p \tag{2.5}$$

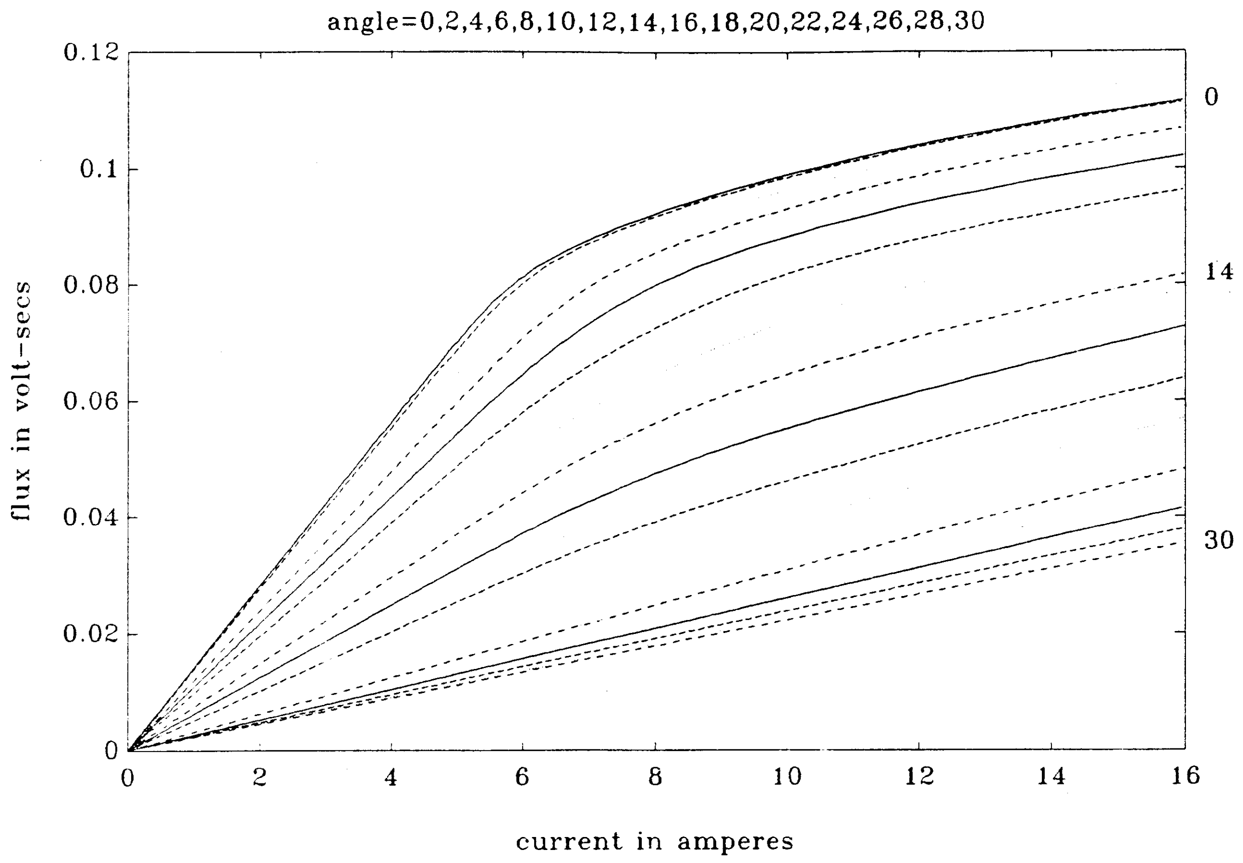

Figure 2.3: Flux versus current curves for the experimental VRM.

where the rotor position for the nth phase $\theta_n$ is given by

$$\theta_n = \theta + (n-1)\frac{2\pi}{N_r N_p} \tag{2.6}$$

The flux linkage curves for Phase 1 of the experimental VRM are shown in Figure 2.3.

Based upon the assumption that there is no magnetic coupling between the phases, the torque $\tau_n$ produced by phase n can be determined from the coenergy $W_n'$ using the relationship

$$\tau_n = \left. \frac{\partial W_n'(\theta, i_n)}{\partial \theta} \right|_{i_n} \tag{2.7}$$

[6] where

$$W_n'(\theta, i_n) = \int_0^{i_n} \lambda_n(\theta, i_n') di_n' \tag{2.8}$$

The total torque is the sum of the individual phase torques so that

$$\tau_m = \sum_{n=1}^{N_p} \tau_n \tag{2.9}$$

In order to evaluate Equations 2.7 and 2.8, the form of Equation 2.5 must be known explicitly. However, in order to simplify the discussion of torque production, it is assumed for now that the VRM is operated with its magnetic material unsaturated. At sufficiently low currents, this is a good assumption for all VRMs. Referring to Figure 2.3, it is seen that the experimental motor is magnetically linear for all rotor positions at currents below about 5 Amperes.

Under the assumption of magnetic linearity, Equation 2.5 can be written as

$$\lambda_n(\theta, i_n) = L_1(\theta_n) i_n \tag{2.10}$$

where $L_n$ is the inductance of the nth phase. The expression for torque in Equation 2.7

then simplifies to

$$\tau_n = \frac{1}{2} i_n^2 \frac{dL_1(\theta_n)}{d\theta_n} \tag{2.11}$$

Equation 2.11 illustrates a point mentioned earlier. Namely, that the direction of torque in a VRM is independent of the polarity of the phase current. Instead, it depends only on the sign of $dL/d\theta$. At sufficiently low currents, the magnitude of the torque in a given VRM is proportional to the current squared. At higher current levels where the magnetic material is in saturation, increases in current still lead to increases in torque, but the increase in torque is no longer a quadratic function of the current magnitude.

### 2.3.2 VRM Control

The control of a VRM can be achieved by varying four parameters: the turn-on angle $\theta_{on}$, the turn-off angle $\theta_{off}$, the chopping current level $i_{chop}$, and the supply voltage $V_{supply}$. The angles $\theta_{on}$ and $\theta_{off}$ define the positioning of the current excitation within a cycle of inductance variation. As illustrated in Figure 2.4, positive or negative torque can be produced depending on whether the phase is excited during a region of rising or falling inductance, respectively. At low to moderate speeds, the magnitude of the torque is controlled mainly by regulating the phase currents using a technique known as current chopping. Chopping is the process of applying a positive voltage $V_{supply}$ to the phase winding whenever the current is below $i_{chop}$ and zero or a negative voltage (depending of the inverter topology) whenever the current is above $i_{chop}$. This causes the peak current to be essentially equal to $i_{chop}$. However, due to nonzero switching times of the applied voltage, there is a small

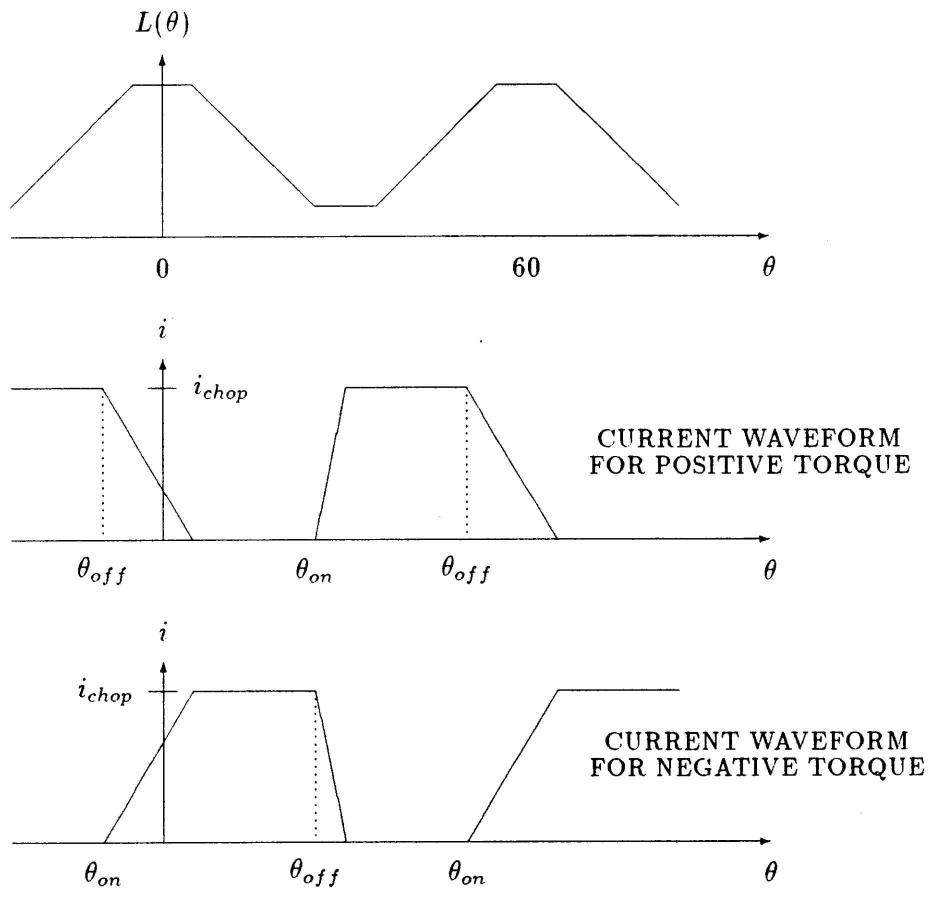

Figure 2.4: Positioning of the current pulse for positive and negative torque production.

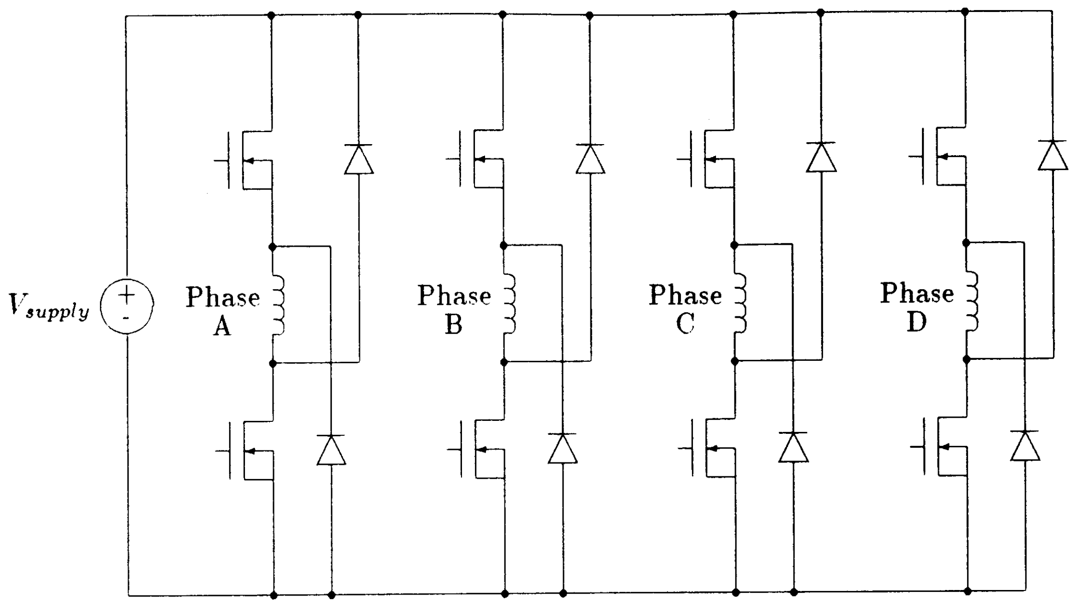

Figure 2.5: Four-phase inverter.

ripple in the current waveform around $i_{chop}$. The supply voltage $V_{supply}$ indirectly controls torque production through the the current. The level of $V_{supply}$ determines how fast the phase current can rise and fall, thereby placing constraints on the values of $\theta_{on}$ and $\theta_{off}$ can have (for positive torque production), the peak value that the current can reach during the conduction interval defined by $\theta_{on}$ and $\theta_{off}$, and the top speed that the VRM can attain.

### 2.3.3 VRM Inverter

The role of the inverter is to provide an interface between the low-level signals generated by the controller and the high-level voltages and currents required by the motor. A simplified schematic of the inverter used in this thesis is shown in Figure 2.5. A detailed schematic is given in Appendix B. At the beginning of the conduction interval denoted by $\theta_{on}$ in Figure 2.4, both the upper and lower transistor switches are turned on. This places the supply voltage $V_{supply}$ across the phase winding causing the current to build up. At the end of the conduction interval denoted by $\theta_{off}$ in Figure 2.4, both switches are turned off. This forces the phase current to flow from the negative terminal of $V_{supply}$, through the lower diode, the phase winding, and upper diode, and to the positive terminal of $V_{supply}$. While the current is flowing through this path, a voltage equal to minus $V_{supply}$ is placed across the phase winding causing the current to be reset to zero.

Chopping is performed by repeated turning the upper switch on and off while keeping the lower one on. When the upper switch is off, the current is forced to flow up from the negative rail of $V_{supply}$, through the lower diode, the phase winding, and the lower switch, and back to the negative rail. This places a small negative voltage (approximately equal to the sum of the voltage drops across the diode and lower switch) across the winding causing a slow decay of the current. Both switches could be used for chopping, but this would cause the current ripple to be greater for a fixed chopping frequency.

# Chapter 3

# Experimental Investigation of the Noise Sources

This chapter describes the experiments performed on the experimental VRM. The purpose of the experiments is first to determine the relative importance of several hypothetical noise sources in the motor. The focus then turns to developing an understanding of the excitation forces and response mechanisms for the sources that are found to contribute the most to the acoustic noise. This should allow appropriate models to be developed for the dominant sources.

## 3.1 Hypothetical Noise Sources

A brief description of several hypothetical noise sources in VRMs is given below.

1. Radial forces are developed by the tendency for the magnetic circuit to adopt a configuration of minimum reluctance. In particular, theses forces act to decrease the gap separation between the stator pole pair of the excited phase and a rotor pole pair, particularly as these poles approach alignment. This results in radial vibrations of the stator which, in turn, produce sound waves. The radial vibrations are largest when the radial forces excite the resonant frequencies of the stator. In addition to the radial

vibrations, lateral vibrations of the stator and rotor teeth are induced by tangential magnetic forces, which produce the torque in a VRM.

2. Lorentz forces act on the phase windings as they carry current. If $\vec{I}$ is the winding current vector and $\vec{B}$ is the slot leakage flux vector produced by the winding currents, then the force per unit length of the wire is $\vec{I} \times \vec{B}$. Since the phase currents vary with time, the forces on the windings also vary, resulting in the winding vibrations.

3. Magnetostrictive forces are present in all compressible magnetic materials under the influence of a magnetic field. Like the radial force discussed above, these forces are developed by the tendency of the magnetic circuit to adopt a configuration of maximum energy. Specifically, these forces act to compress the material in order to increase its effective permeability. In small transformers, these forces are responsible for the buzzing sound.

4. Bearing imperfections can develop either from the manufacturing process or from normal wear. These imperfections can lead to vibrations of the rotor shaft, which in turn, can be coupled to the motor's endbells through the bearing housing or to the motor's load through the shaft.

5. Other nonuniformities introduced in the manufacturing process can cause unbalanced radial or axial forces to be produced leading to undesirable bending or wobbling of the rotor shaft.

6. Windage noise produced by the motion of the rotor. This can cause the motor to act as a siren.

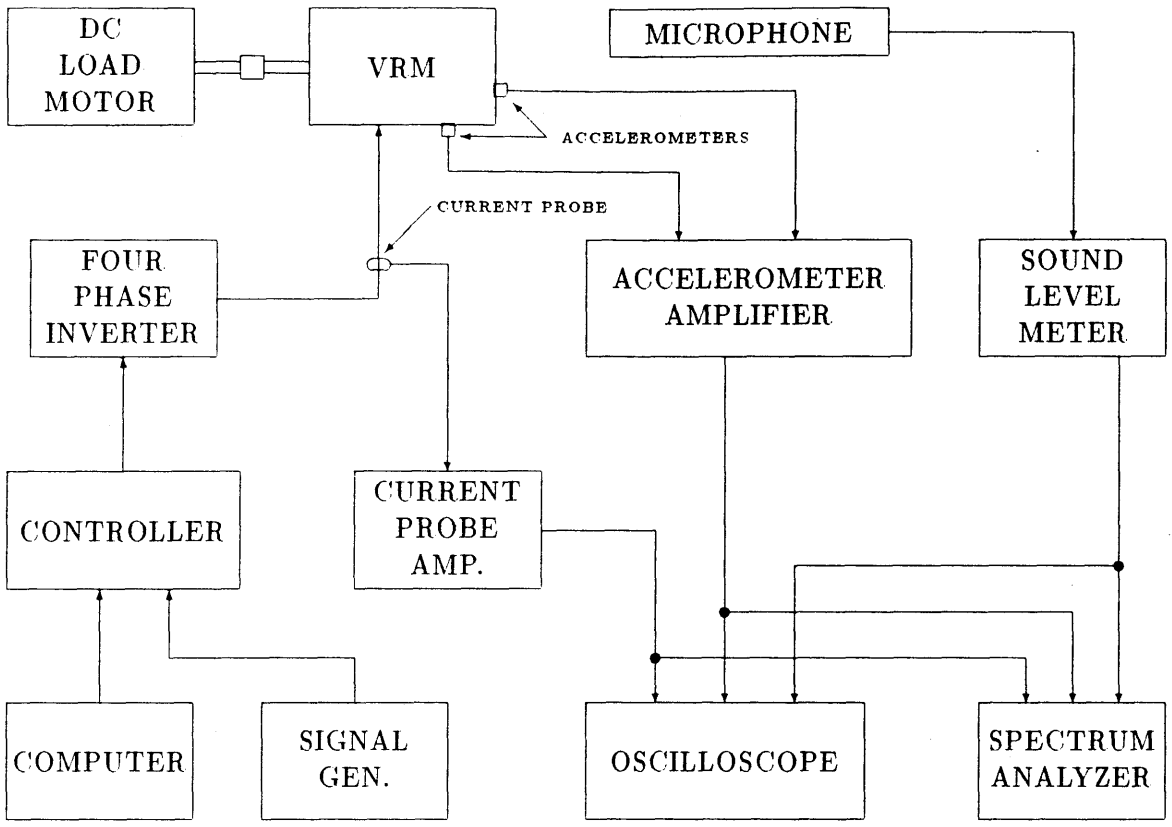

Figure 3.1: Schematic diagram of the experimental setup.

The acoustic noise sources in a VRM can be classified as being either of magnetic origin or mechanical origin. Of the noise sources listed above, those of magnetic origin include the first three items and part of the fifth while those of mechanical origin include the fourth, part of the fifth, and the sixth item.

## 3.2 Experimental Setup

The schematic diagram of the experimental setup is shown in Figure 3.1; more details are given in Appendix B. The variable-reluctance motor used in testing has four phases and a rating of 1/2-hp at about 7000 rpm. The excitation for the motor is provided by a four-phase inverter (see Figure 2.5). The controller is implemented using an Intel 8032 microcontroller. The computer is used to download various control programs to the controller and to monitor the operation of the motor. The signal generator provides an accurate frequency signal necessary for some of the experiments. Measurements are made of the VRM current, acceleration, and sound output. In all experiments, current measurements are made on Phase A of the VRM using a current probe; Phase A serves as the reference for all measurements. Acceleration measurements are made using two moveable accelerometers that are attached to the motor using beeswax. Sound measurements are made in an anechoic box using a single microphone located 20 cm from the surface of the motor. The signal from the microphone undergoes A-weighting by the sound level meter [29]. Noise level readings from the meter are in decibels where 0 dB = 0.0002 $\mu$bar. In the rotational experiments, a DC motor is used as a load for the VRM. The terminals of the DC motor are attached to an adjustable current source, so that various loads could be presented to the VRM.

## 3.3 Determination of the Dominant Noise Source

Due to the complexity of the noise sources listed in Section 3.1, the first goal of the experiments is to determine the relative importance of the noise sources of magnetic origin and those of mechanical origin. In the first experiment, a DC motor is attached to the shaft of the VRM using a rubber coupling in order to minimize the transmission of vibrations between the two motors. The total load torque due to brushes of the DC motor and the

| speed (rpm) | noise level (dB) | speed (rpm) | noise level (dB) |
|---|---|---|---|
| 500 | 68 | 4500 | 79 |
| 1000 | 74 | 5000 | 80 |
| 1500 | 76 | 5220 | 92 |
| 2000 | 76 | 5500 | 83 |
| 2500 | 79 | 6000 | 79 |
| 3000 | 80 | 6500 | 76 |
| 3500 | 80 | 7000 | 76 |
| 3700 | 86 | 7500 | 77 |
| 4000 | 80 | 8000 | 92 |

Table 3.1: Measured noise level of the experimental VRM.

bearings of both motors is given approximately by the following equation

$$\text{load torque} = 0.04 + (5 \times 10^{-5})\omega \text{ Nm} \tag{3.1}$$

where $\omega$ is the rotor speed in radians per second.

The VRM is then operated at speeds up to 8000 rpm while noise measurements are made with the microphone at various positions 20 cm away from the surface. The measurements reveal three narrow "noisy" speed ranges where the noise level is over 90 dB and "quiet" speed ranges where the noise level is below 75 dB. A noise level of 90 dB is considered to be at the unsafe threshold (see Table 1.1), while an average factory produces noise at a level of 75 dB. The measured noise level at several discrete speeds is given in Table 3.1.

Next, the power to the VRM is cut while it is operating at 8000 rpm. Noise measurements taken as it coasts down reveal that, after an initial transient, the noise level remains essentially constant at a level below 72 dB. This result shows that the noise sources of magnetic origin are responsible for most of the acoustic noise, since the noise is highest only when there is magnetic flux in the VRM. This conclusion is supported by a second experiment, where the load motor is used to drive the VRM up to 8000 rpm while noise measurements are taken. The phases of the VRM are unconnected so that no current flows. This time the maximum noise level is about 75 dB, again showing that the noise sources of magnetic origin in the VRM are more important than those of mechanical origin; i.e., windage noise or noise caused by the bearings or mechanical imbalances.

Having determined that the noise sources of magnetic origin are responsible for the acoustic noise, the next series of experiments is aimed at determining which of these mechanisms is dominant at the "noisy" speeds. At these speeds, it appears that the magnetic forces excite mechanical resonances of the VRM since at these speeds, there is a dramatic increase in the noise level. Therefore, the next series of experiments is performed to determine the resonant frequencies of the VRM and at those frequencies, which parts of the VRM are vibrating. To simplify matters, these experiments are performed using a relatively simple excitation waveform (described below) on a single phase of the VRM, which means that the VRM is not rotating. The results of these experiments are later verified by rotational experiments in which all four phases of the VRM are excited in the conventional manner.

The resonant frequencies of the experimental VRM are determined by exciting Phase A with a current waveform of varying frequency content and identifying peaks in the acoustic noise level. The load motor is disconnected so that the shaft of the VRM is free to vibrate. The VRM is resiliently mounted to prevent its vibrational behavior from being affected by the mount. The inverter drives Phase A with a current waveform consisting

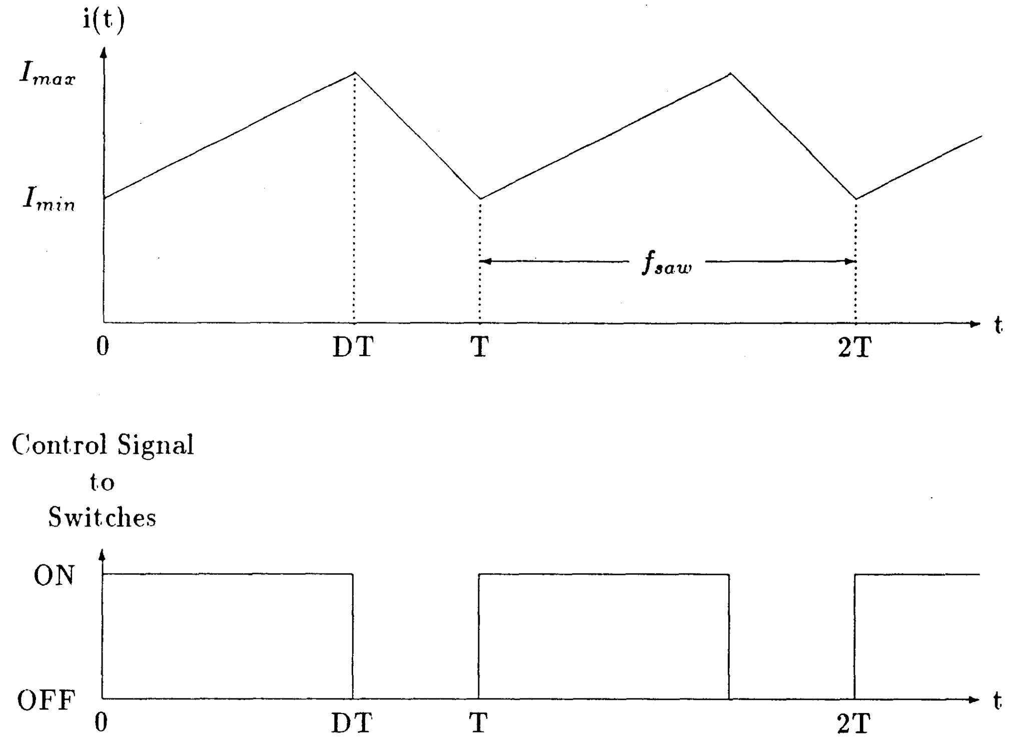

Figure 3.2: Sawtoothed current waveform used to drive Phase A.

of a sawtoothed AC component on top of a larger DC component as shown in Figure 3.2. The inverter generates this waveform by repeatedly turning both Phase A switches on and off (see Figure 2.5). The frequency of the switching $f_{saw}$ is varied from 100 Hz to about 15 kHz, while the duty cycle $D$ and the supply voltage $V_{supply}$ are adjusted so that $I_{min}$ and $I_{max}$ remain constant. The values of $I_{min}$ and $I_{max}$ are chosen to be 1.75 A and 2 A, respectively, so that the magnetic material remains unsaturated and so that measurable acoustic noise is produced at all frequencies. The current $i(t)$ can be written as

$$i(t) = a_0 + a_1 \cos(\omega t - b_1) + \text{h. o. t.} \tag{3.2}$$

where $a_0$, $a_1$, and $b_1$ are the Fourier series expansion coefficients and $\omega = 2\pi f_{saw}$. Since magnetic force in linear magneto-quasistatic systems is proportional to the current squared [6,28], the force which results from the current $i(t)$ in Equation 3.2 has components not only at the fundamental frequency $\omega$, but also at the harmonics of $\omega$. It is necessary, therefore, to check that the frequency of the dominant acoustic noise component and $\omega$ coincide before declaring $\omega$ to be a resonant frequency. This experiment reveals that the VRM has resonant frequencies at 2604 Hz, 9200 Hz, and 14.2 kHz. At all three frequencies, the signal picked up by the microphone is almost purely sinusoidal.

At all three resonant frequencies, the following experiment is performed to determine if vibrations the phase windings are causing the noise. Since the slot leakage flux is maximum when Phase A's stator poles are not aligned with any rotor poles, the magnetic force acting on the windings and therefore, the acoustic noise, should be greatest in this configuration. However, by turning the rotor while applying the current excitation to Phase A, the exact opposite is found to be true at all three resonant frequencies. At 2604 Hz, the noise level drops from 94 dB to 67 dB when the rotor is turned from an aligned position to an unaligned position. Similar results are found at the other two resonant frequencies, thus ruling out vibrations of the phase windings as a source of the acoustic noise. This experiment also rules out lateral vibrations of the rotor or stator teeth as a source of the acoustic noise since the torque in a VRM is zero in the aligned position and maximum at a position between maximum and minimum alignment.

The next two experiments are performed to determine the relative importance of the radial magnetic forces, the magnetostrictive forces, and magnetic forces due to imperfections in the VRM. In the first experiment, acceleration and noise measurements are taken at each resonant frequency in order to identify the parts of the VRM producing the acoustic noise. The current waveform of Figure 3.2 is used again. The sound measurements are made by placing the microphone at various positions on a hemisphere 10 cm from the motor. They show that much more noise emanates from the outer surface of the stator than from the endbells. These measurements are repeated at a distance of 20 cm from the motor. At all three resonant frequencies, they show that about 10 dB more noise emanates from the stator than from the endbells of the VRM.

In the second experiment, acceleration measurements are made at various points on the stator, endbells, and rotor shaft. The acceleration measurements are always made in symmetric pairs. For example, if one accelerometer is placed at a particular point on the stator, the other is placed at a diametrically opposing point, or if one accelerometer is placed on one endbell, the other is placed at a symmetric location on the other endbell, etc. The dual acceleration measurements allow distinctions to be made between compressive/expansive motion and translative motion. These measurements reveal the following types of motion:

1. Compressive radial vibrations of the stator caused by the radial magnetic forces acting to reduce the gap separation between the stator and rotor poles. Magnetostriction is ruled out in this case, because measurements made around the circumference of the stator reveal that there are peaks and nodes in the acceleration magnitude. Magnetostriction would tend to cause uniform radial vibrations unless the magnetic material

were nonuniform in the circumferential direction.

2. Compressive axial vibrations of the two endbells caused by magnetostrictive forces. The magnitude of the vibrations is maximum where the stator contacts the endbells, and drops off sharply as the accelerometers are moved towards the center of the endbells.

3. Translative axial vibrations of the shaft caused by the misalignment of the rotor lamination stack with respect to the stator lamination stack. Magnetic forces are generated which tend to pull the rotor laminations into alignment with the stator laminations.

4. Bending of the shaft due to nonuniform gap lengths between the rotor and stator poles. Since radial magnetic forces are inversely proportional to the gap length, the stator/rotor pole pair having the shorter gap length would generate greater forces than the apposing pair causing the rotor to bend.

The maximum acceleration for these four types of motion at the three resonant frequencies is shown in Table 3.2. The maximum radial acceleration (and hence displacement) of the stator is much larger than that of the other three types of motion. This, coupled with the result of the sound measurements which reveal that the most noise is coming from the stator, leads to the conclusion that the majority of the acoustic noise in the experimental VRM is due to radial vibrations of the stator caused by radial magnetic forces in the motor.

Before proceeding further, two important qualifications must be made about the experimental results presented thus far. First, although the vibrations of the endbells and rotor shaft don't produce significant noise, they are large enough to be considered potential noise

| motion | maximum acceleration in g's | | |
|---|---|---|---|
| type | 2604 Hz | 9200 Hz | 14.2 kHz |
| 1 | 18.30 | 21.29 | 28.90 |
| 2 | 3.05 | 2.29 | 9.25 |
| 3 | 0.24 | 0.72 | 1.54 |
| 4 | 3.91 | 1.10 | 2.19 |

Table 3.2: Maximum acceleration for the four types of motion at the three resonant frequencies.

sources. In the above experiments, the experimental VRM is essentially an isolated structure. In a real application, vibrations of the endbells could excite mechanical resonances of the support structure, while vibrations of the shaft could do the same to the load attached to the VRM. Second, since the experiments are performed only on the experimental VRM, the results are not necessarily valid for all VRMs. However, experience shows that they are generally true for VRMs having similar constructions as the experimental VRM.

## 3.4 Characterization of the Dominant Noise Source

Having identified radial vibrations of the stator as being the dominant noise source, the focus of the experiments turns to investigating in greater detail the the vibrational behavior of the stator at the resonant frequencies and the radial magnetic forces responsible for inducing the vibrations so that appropriate models can be developed. To simplify matters, stationary experiments are performed first. Using knowledge gained from these experiments, rotational experiments are performed to complete the investigation.

### 3.4.1 Stationary Experiments

In the first series of stationary experiments, the excitation applied to Phase A again is the current waveform of Figure 3.2. Detailed acceleration measurements are made on the stator at the three resonant frequencies in order to determine the pattern of vibration around the circumference of the stator and along its length (henceforth referred to as the circumferential and longitudinal mode shapes, respectively). Acceleration measurements are made at 16 equally spaced points around the circumference of the stator and at 5 equally spaced points along its length as shown in Figure 3.3. One accelerometer is placed above Phase A at axial Point (I) to serve as a reference point. The other accelerometer is used to make measurements at the 15 other points around the circumference of the stator. This procedure is repeated at the other four axial measuring points so that acceleration amplitudes and phases, relative to the reference point can be made.

At 2604 Hz, the acceleration measurements reveal that the stator is exhibiting single-ovalization, which is also referred to as the 2nd order circumferential vibration mode [23]. Figure 3.4 shows the (highly exaggerated) deflection of the stator at two instances in a cycle of vibration when the deflection of the stator is at a maximum. Typically, the maximum deflection is on the order of a micron. The orientation of the stator in Figure 3.4 is the same as that in Figure 3.3. The vibrations of the stator above Phase A are 180 degrees out of phase with the vibrations above Phase C, while there are nodes located above Phases B and D. The amplitude and phase of the vibrations are essentially uniform in the axial direction; the vibrations at axial Points (I) and (V) are slightly larger than those at Point (III).

At 9200 Hz, the measurements reveal that the stator is exhibiting double-ovalization as

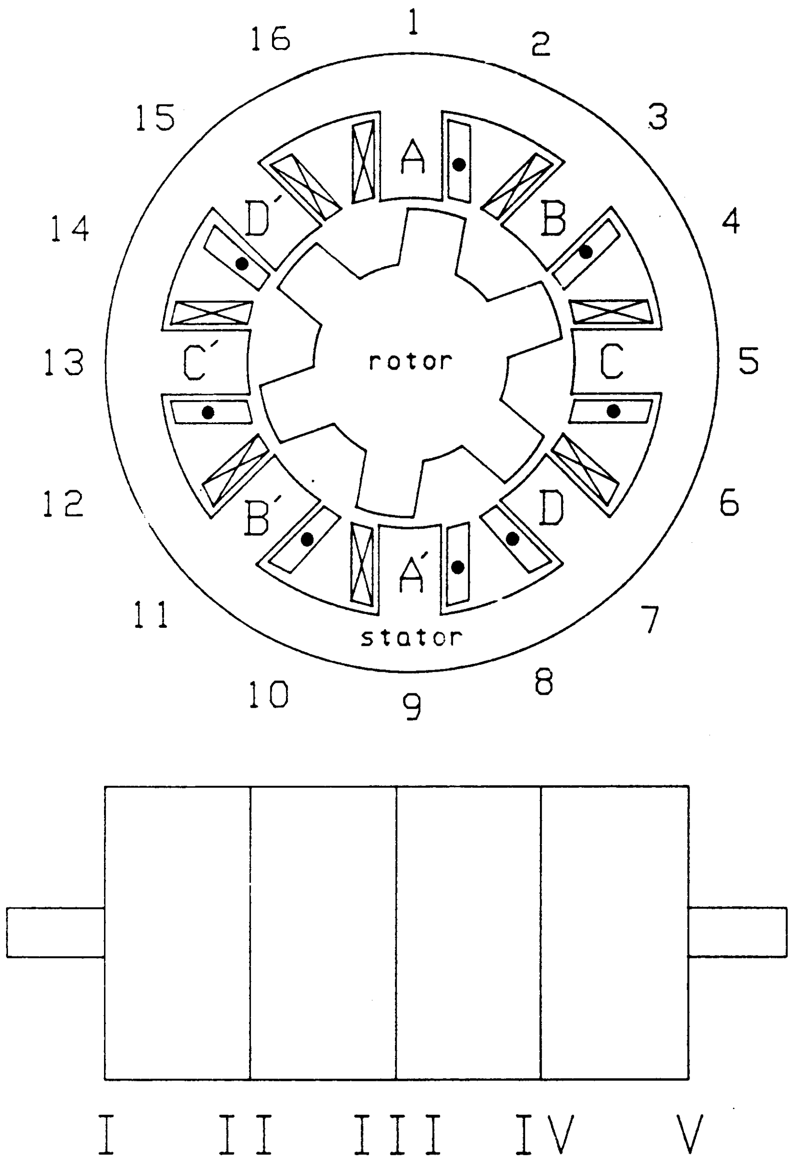

Figure 3.3: Measurement points on the experimental VRM.

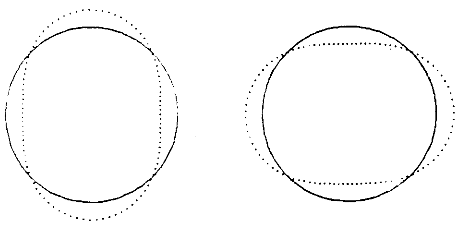

Figure 3.4: Single-ovalization of the stator at 2604 Hz.

shown in Figure 3.5; double-ovalization is also referred to as the 4th order circumferential vibration mode. In this case, the vibrations above Phase C are in phase with those above Phase A, while the vibrations above Phases B and D are 180 degrees out of phase with those above Phase A. As in the previous case, there is little variation in the amplitude or phase of the vibrations along the length of the stator.

At 14200 Hz, the measurements reveal the zeroth order mode of vibration as shown in Figure 3.6; the zeroth order mode is also referred to as the breathing mode. In this case, the vibrations around the circumference of the stator are all in phase. The magnitude, however, shows some variation; the amplitude of the vibrations above Phase A are larger than any other point. As in the previous two cases, there is little variation in the amplitude or phase along the length of the stator.

These three mode shapes are also found in sound measurements when the microphone is located at the same points around the stator as the accelerometers, except at a distance

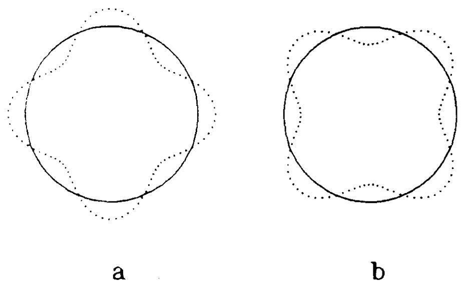
Figure 3.5: Double-ovalization of the stator at 9200 Hz.

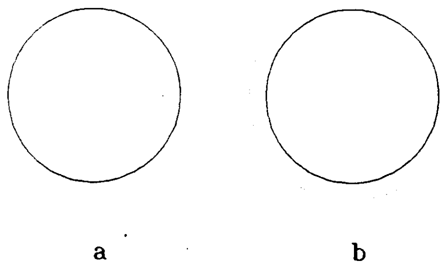
Figure 3.6: Zeroth order vibration mode at 14200 Hz.

of 20 cm. The magnitude and phase patterns are the same as those seen in the acceleration measurements for all three resonant frequencies.

In the second series of stationary experiments, the excitation $i(t)$ applied to Phase A consists of unidirectional current pulses. The duty cycle of the pulses is set to 25%. The frequency of the pulses is varied from 10 Hz to 800 Hz, corresponding to a rotational speed ranging from 100 rpm to 8000 rpm, respectively, if the motor were actually rotating. The amplitude of the pulses is set to 2 A, again to prevent saturation from occurring. The same measurements of acceleration and sound are made for this series of tests as those for the first series.

The experiments revealed that the acceleration and acoustic noise of the motor have responses at the same frequencies as the frequency components of $i^2(t)$, and that the two responses are largest when a harmonic of $i^2(t)$ coincides with the resonant frequency at 2604 Hz; the harmonics of $i^2(t)$ near 9200 Hz and 14200 Hz are too small to cause any measurable response. Figure 3.7 shows the frequency spectra of $i^2(t)$, the acceleration, and the noise waveforms at a pulse frequency of 372 Hz. The acceleration is measured at Point (1,III) on the stator, while the noise is measured above Phase A. The presence of the 7th harmonic of $i^2(t)$ at 2604 Hz, which is more than 15 dB lower than the 1st harmonic, results in acceleration and acoustic noise waveforms whose 7th harmonics are over 30 dB larger than their 1st harmonics. The noise level recorded by the sound level meter set for A-weighting and averaged response is 89 dB. Acceleration measurements show that the stator is again exhibiting single-ovalization. This finding is consistent with the results of the first series of stationary experiments which show that single-ovalization is the mode shape associated with the resonant frequency at 2604 Hz. Since, as stated above, the harmonics of $i^2(t)$ near

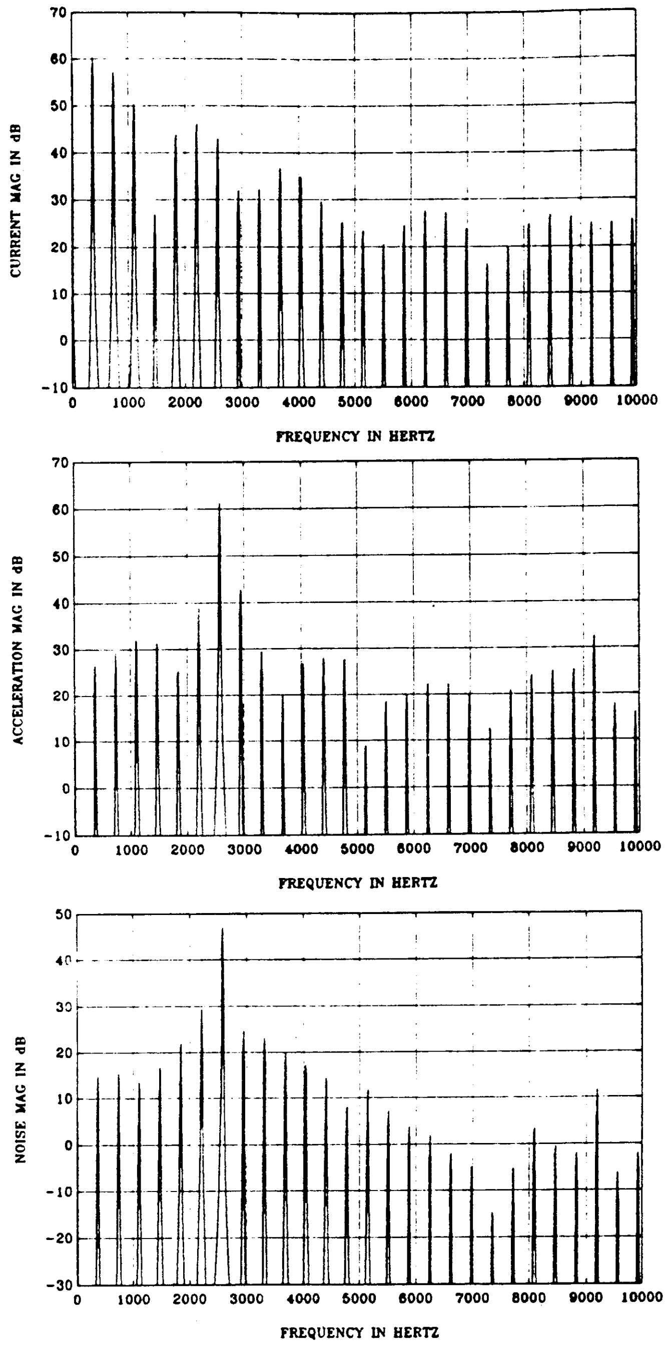

Figure 3.7: Frequency spectra of $i^2(t)$, acceleration, and acoustic noise. The fundamental frequency of the current pulses is 372 Hz.

9200 Hz and 14200 Hz are too small to excite those resonances, the fourth and zeroth order mode shapes are not present in the measurements.

The same type of results are seen at pulse frequencies of 289 Hz and 521 Hz, where the 9th and 5th harmonics, respectively, excite the resonance at 2604 Hz, but are noticeably absent at 325 Hz and 651 Hz. This is due to the fact that a 25% duty cycle square wave doesn't have 4th or 8th harmonics. Since the current pulses are not exactly square, there are nonzero 4th and 8th harmonics. However, these harmonics are too small to measurably excite the resonance at 2604 Hz.

The results of the two series of stationary experiments show that the magnitude of the acceleration and noise responses are dependent on the harmonic content of $i^2(t)$. Since the harmonic content of $i^2(t)$ depends not only on the frequency of the current pulses, but also their shape, it seems reasonable to expect changes in the noise level when the duty cycle of the pulses is changed. To test this, an experiment is performed, which again uses unidirectional current pulses. This time, the duty cycle of the pulses is varied while the fundamental frequency is held constant at 372 Hz. The noise level measured at several different duty cycles is given in Table 3.3. As evident from this table, variations in the duty cycle of the current pulses cause the noise level to change significantly.

The results of the stationary experiments show that for the experimental VRM, single-ovalization of the stator is responsible for producing the acoustic noise. The ovalization is due to the coincidence of a harmonic component of $i^2(t)$ and the resonant frequency at 2604 Hz; the harmonics of $i^2(t)$ near the other two resonant frequencies are too small to cause any measurable response. This result illustrates that not only the frequency of the current pulses, but their shape determine the amount of acoustic noise produced. As discussed in

| duty cycle | noise level | duty cycle | noise level |
|---|---|---|---|
| (percent) | (dB) | (percent) | (dB) |
| 17 | 88 | 24 | 90 |
| 18 | 79 | 26 | 88 |
| 19 | 84 | 27 | 87 |
| 20 | 86 | 28 | 85 |
| 21 | 88 | 29 | 81 |
| 22 | 89 | 30 | 76 |
| 23 | 89 | 31 | 72 |

Table 3.3: Variation of the noise level with the duty cycle

Chapter 2, the shape of the current waveform depends on, among other things, the voltage excitation applied to the phase windings. This fact raises the possibility of reducing the acoustic noise by changing the voltage excitation so that the harmonics of $i^2(t)$ at the resonant frequencies of the motor are reduced. The results of the stationary experiments also offer an explanation for the "noisy" and "quiet" speeds that are observed in the initial rotational experiments, since the current pulse frequency of the phase windings in a VRM is directly related to the rotational speed of the motor.

### 3.4.2 Rotational Experiments

In the first rotational experiment, the VRM is operated at a speed of 3720 rpm so that the fundamental frequency of the current pulses is again 372 Hz. The total load torque seen by the VRM is 0.106 Nm. As in the case of the stationary experiments using unidirectional current pulses, the acceleration and acoustic noise of the VRM are found to have responses at the same frequencies as the frequency components of $i^2(t)$. Likewise, the presence of the 7th harmonic of $i^2(t)$ at 2604 Hz results in significant acceleration and noise components at that frequency. This is illustrated in Figure 3.8. The $i^2(t)$ spectrum is derived from Phase A's current, while the acceleration is measured at Point (1,III) on the stator. The noise is measured above Phase A. The turn on and turn off angles, $\theta_{on}$ and $\theta_{off}$, are 34 and 49 degrees, respectively, and $V_{supply}$ is 40 volts. The current chopping level $i_{chop}$ is set high enough so that chopping doesn't occur. The noise level recorded at this operating point is 92 dB. The current, acceleration, and noise spectra are the same for the other three phases.

Unlike the stationary experiments, the acceleration pattern around the circumference of the stator is not stationary. At 3720 rpm, the stator is ovalizing, but the oval is rotating at 1/3 the speed of the rotor in the opposite direction as shown in Figure 3.9. Again, the orientation of the stator is the same as that in Figure 3.3. The time it takes for the oval to go from the Orientation A in Figure 3.9, through Orientations B through B, and back to Orientation A equals 1/(372Hz), the period of the current pulses.

In order to illustrate the effect that the shape of the current pulses has on the noise level in the rotational case, the following experiment is performed. While the VRM is still operating at a speed of 3720 rpm and a load of 0.106 Nm, $\theta_{on}$, $\theta_{off}$, and $V_{supply}$ are adjusted to reduce the noise level while, at the same time, keeping the speed constant at 3720 rpm. The new values of $\theta_{on}$, $\theta_{off}$, and $V_{supply}$ which define the second operating point are shown in Table 3.4. The values for the first operating point are shown for comparison. As a result of the adjustments, the noise level decreases by 6 dB to 86 dB. The difference in the shape of the current pulses for the two operating points is shown in Figures 3.10 and 3.11. The new $i^2(t)$, acceleration, and noise spectra are shown in Figure 3.12. The acceleration and noise spectra show a reduction in the frequency components at 2604 Hz as compared to the corresponding spectra in Figure 3.8 (note the scale change). The exact opposite is true,

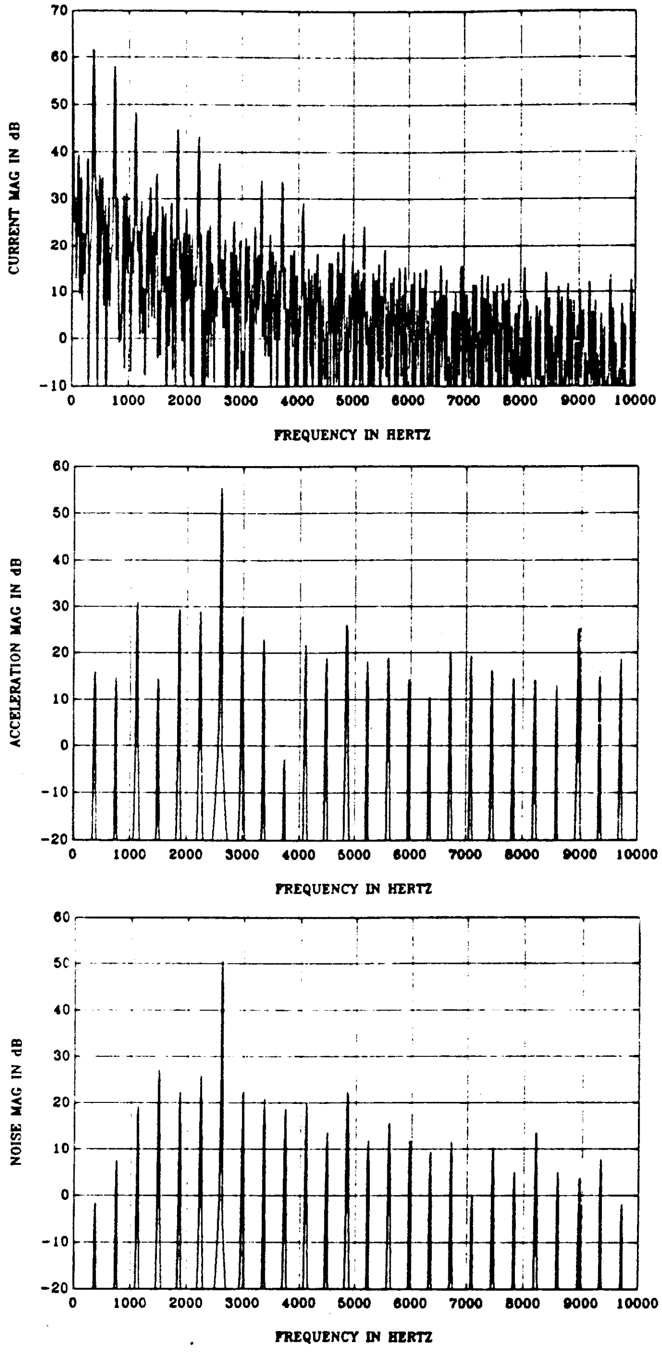

Figure 3.8: Frequency spectra of $i^2(t)$, acceleration, and acoustic noise at the operating point defined by $\theta_{on} = 34°$, $\theta_{off} = 49°$, $V_{supply} = 40$ V, and a motor speed of 3720 rpm.

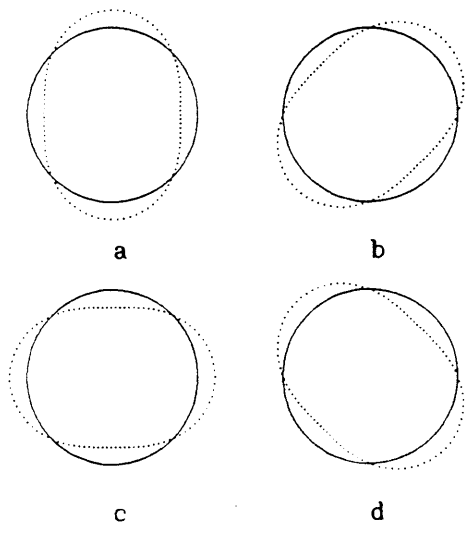

Figure 3.9: Single-ovalization of the stator at 3720 rpm.

| operating point | $\theta_{on}$ (degrees) | $\theta_{off}$ (degrees) | $V_{supply}$ (volts) | speed (rpm) | $i_{chop}$ (amps) | load torque ( $\times 10^{-3}$ Nm) |
|---|---|---|---|---|---|---|
| 1 | 34 | 49 | 40.0 | 3720 | – | 106 |
| 2 | 30 | 45 | 30.0 | 3720 | – | 106 |

Table 3.4: Definition of the two operating points used to evaluate the performance of the constant speed VRM model.

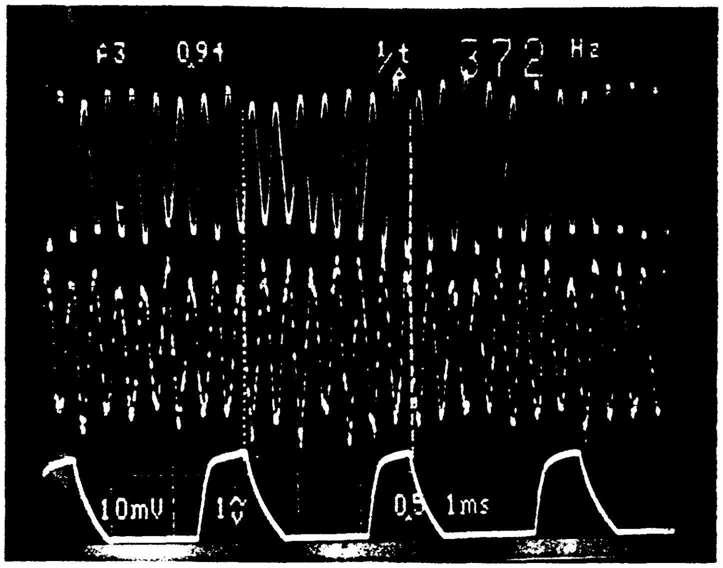

Figure 3.10: Top trace: output of sound level meter on 90 dB setting; mid: accelerometer output @ 9.6 g/div; bot: current @ 2 A/div for $\theta_{on} = 34°$, $\theta_{off} = 49°$, and $V_{supply} = 40$ V.

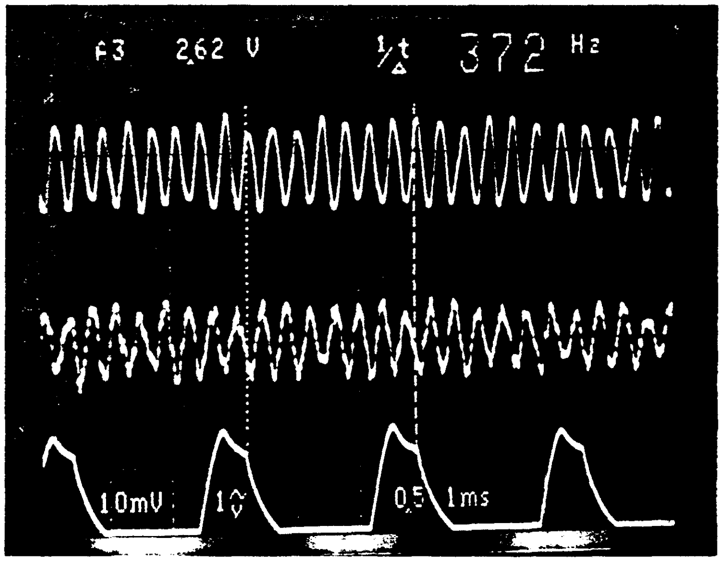

Figure 3.11: Top trace: output of sound level meter on 90 dB setting; mid: accelerometer output @ 9.6 g/div; bot: current @ 2 A/div for $\theta_{on} = 30°$, $\theta_{off} = 45°$, and $V_{supply} = 30$ V.

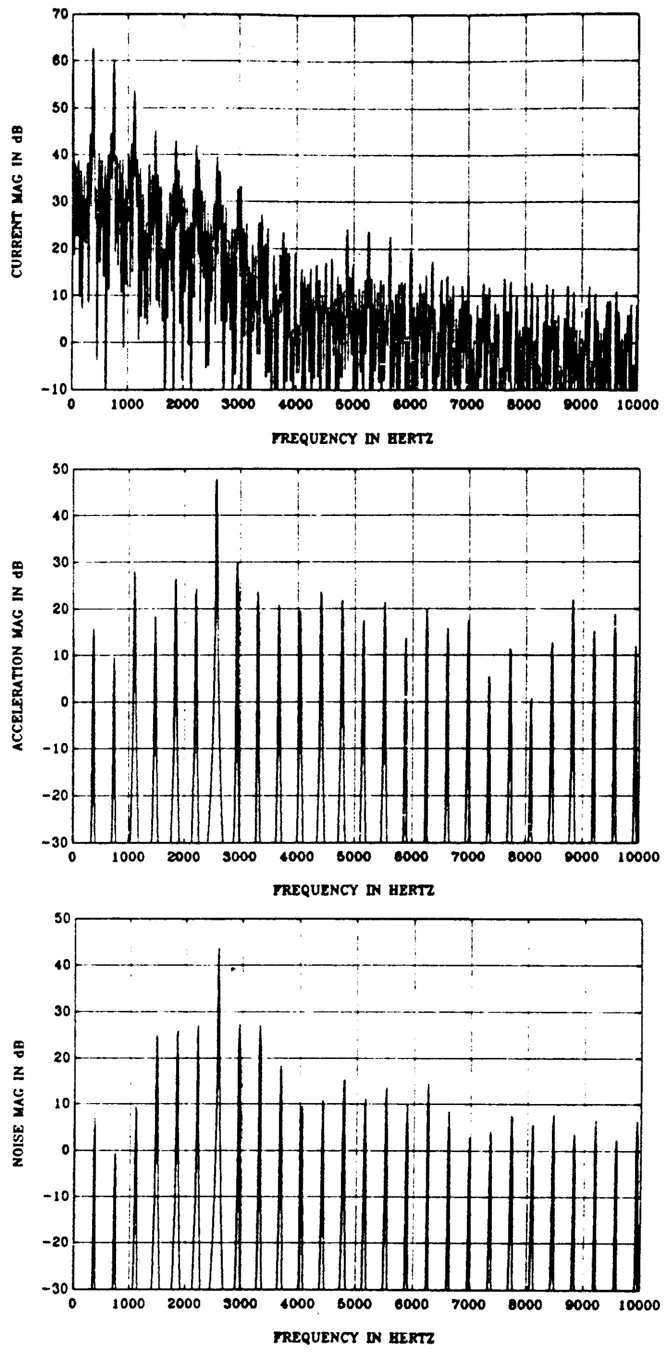

Figure 3.12: Frequency spectra of $i^2(t)$, acceleration, and acoustic noise at the operating point defined by $\theta_{on} = 30°$, $\theta_{off} = 45°$, $V_{supply} = 30$ V, and a motor speed of 3720 rpm.

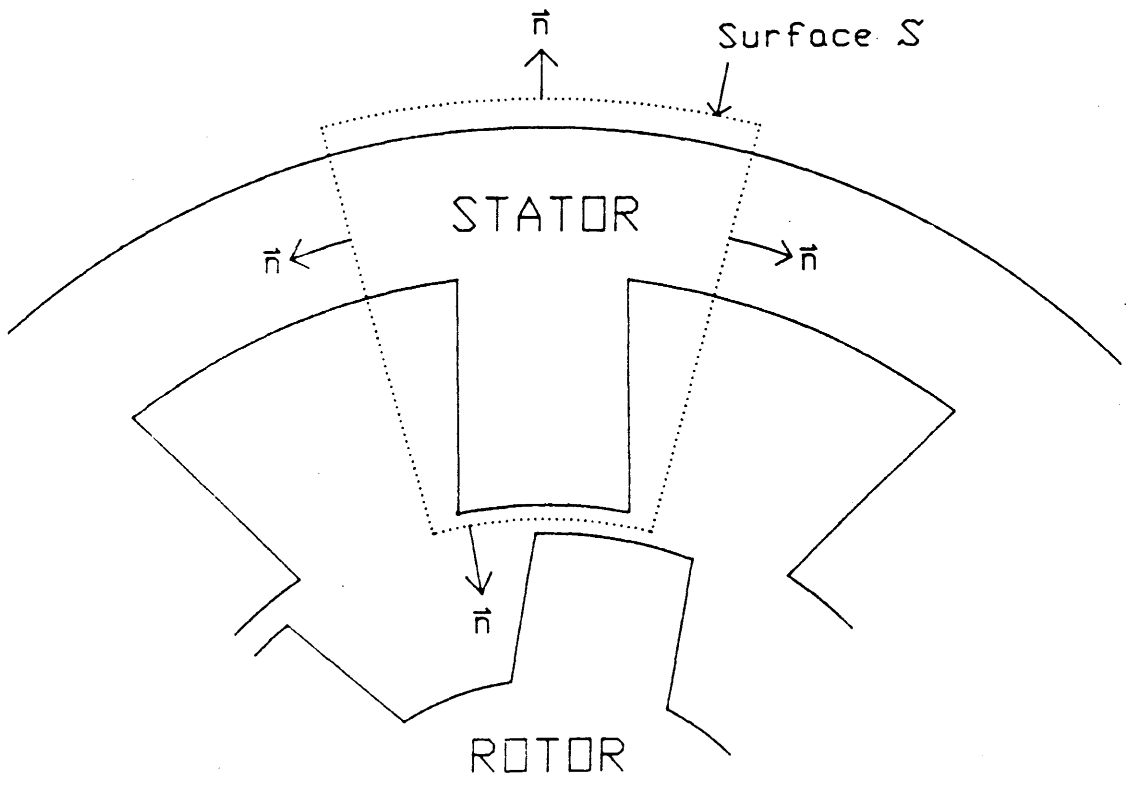

Figure 3.13: Surface used to determine the radial force acting on a stator pole.

however, for $i^2(t)$. The frequency component at 2604 Hz shows an increase as compared to the previous spectrum shown in Figure 3.8. The reason for this apparent discrepancy is that the radial force in the rotational case is not only a function of $i^2(t)$, but also of $\theta(t)$ as shown below.

In deriving an expression for the radial force $F_r$, it is assumed that the flux in the air gap between the stator and rotor poles is radially directed and uniform in the tangential direction. Under these assumptions, $F_r$ can be determined from the following equation [28]:

$$F_r = \oint_S \frac{\mu_o H_r^2}{2} \hat{\imath}_r \cdot \vec{n}\, da \tag{3.3}$$

where $\mathcal{S}$ is a surface which encloses a stator pole as shown in Figure 3.13, $\vec{n}$ is the outwardly-directed unit vector normal to this surface, $H_r$ is the radially directed magnetic field intensity which passes through $\mathcal{S}$, $\hat{\imath}_r$ is the unit vector in the radial direction, and $\mu_o$ is the permeability of free space. Integration of Equation 3.3 yields

$$F_r = -\frac{\mu_o\, A_{gap}(\theta)\, H_r^2}{2} \tag{3.4}$$

where $A_{gap}(\theta)$ is the effective cross-sectional overlap area between the stator and rotor poles. The word "effective" is used here, because the concept of an air gap or overlap area becomes obscured as the rotor and stator poles move away from alignment. The magnetic field intensity $H_r$ can be written in terms of the phase current $i$, the air gap length $g$, and the number of winding turns $N$ per phase as follows

$$H_r = \frac{Ni}{2g} \tag{3.5}$$

Substituting Equation 3.5 into Equation 3.4 yields

$$F_r = -\frac{\mu_o\, A_{gap}(\theta)\, N^2\, i^2}{8g^2} \tag{3.6}$$

In the stationary experiments, $\theta$ is constant so that $F_r$ depends only on $i^2(t)$. In the rotational case, however, $i^2(t)$ is modulated by the time-varying function $A_{gap}(\theta)$, an effect not shown in the spectrum plots. Therefore, in order to predict the acceleration and acoustic noise in the rotational case, the function $A_{gap}(\theta)$ along with $i^2$ need to be known.

## 3.5 Summary of the Experimental Results

In this chapter, an experimental procedure for determining the cause of the acoustic noise of the experimental VRM is described. The procedure was designed to investigate several hypothetical noise, which are listed in Section 3.1, to determine which ones, if any, were dominant. The first series of experiments focused on determining the relative importance of the noise sources of magnetic origin and those of mechanical origin. These experiments showed that there were several "noisy" and "quiet" speed ranges. At the "noisy" speed ranges, the noise sources of magnetic origin were found to be dominant. In fact, when the magnetic excitation was removed from the motor, the noise remained constant at a low level for all speeds between zero and 8000 rpm. The second series of experiments were performed to determine which of the noise sources of magnetic origin was dominant at the "noisy" speeds. These experiments revealed that the experimental VRM had resonant frequencies at 2604 Hz, 9200 Hz, and 14200 Hz, and that at those frequencies, the acoustic noise was caused by radial vibrations of the stator induced by radial magnetic forces in the motor.

Having reached this conclusion, a series of experiments using single-phase excitation were then performed to determine exactly how the acoustic noise was being produced. These experiments showed that radial vibrations of the stator were set up when harmonics of the current squared waveform coincided with a resonant frequency of the stator. The pattern of the radial vibrations and acoustic noise around the circumference of the stator depended on which resonant frequency was being excited; at 2604 Hz, 9200 Hz, and 14.2 kHz, the second, fourth, and zeroth order circumferential modes were observed, respectively. However, when the motor phase was excited with current pulses whose frequency fell within the range of frequencies that would be seen during normal operation of the VRM, the harmonics of the current squared waveform around 9200 Hz and 14.2 kHz were too small to produce any measurable vibration or noise response; the experimental VRM was designed to operate at speeds up to 8000 rpm which corresponds to a pulse frequency of 800 Hz for a single phase.

To verify the results of the single-phase stationary experiments, rotational experiments were performed using conventional excitation on all four phases of the experimental VRM. These experiments showed that the magnitude of the acoustic noise and radial acceleration of the stator depended on the shape of the radial force pulses. Specifically, the noise and acceleration were largest when harmonics of the radial force pulses coincided with the resonant frequency at 2604 Hz; the harmonics near the other two resonant frequencies were again too small to produce any response. The experiments also showed that by changing some of the operating parameters, it was possible to make the VRM run quieter at a given speed and load torque. The reason for this was that the shape of the radial force pulses depended on the operating parameters. This dependence was illustrated through Equation 3.6 which showed that the shape of the radial force pulses depended on the current squared waveform and the air gap function $A_{gap}(\theta)$. The shape of the current squared waveform, as discussed in Chapter 2, depended on the turn-on and turn-off angles, the supply voltage, and the current chopping level. By changing these operating parameters so that the harmonics of the radial force pulses near 2604 Hz decreased in magnitude, the acoustic noise could be reduced.

This result lead to the conclusion that methods were needed to calculate the vibrational behavior of the stator based on parameters defining its geometry and the radial forces of the motor based on parameters defining its operation. These two issues are examined in Chapters 4 and 5, respectively. Once these were known, a model could be developed that took as inputs the geometric and operational parameters and that gave as an output the magnitude of the stator acceleration. The acoustic noise at a given frequency, as the experiments have shown, is proportional to the acceleration of the stator. This model could be used to predict not only the acoustic noise of a particular design, but the sensitivities of the noise to variations in the parameters. This would allow the operating parameters which minimized the acoustic noise for a particular VRM operating at a given speed and load to be determined. As noted earlier in the chapter, the results and conclusions of this chapter are based on tests performed on the experimental VRM, and are not necessarily valid for all VRMs. It would seem reasonable, however, to expect that they would hold for VRMs having similar structures.

# Chapter 4

# Determination of the Resonant Frequencies and Modes of the Stator

## 4.1 Introduction

In the previous chapter, the experiments showed that the acoustic noise problem of the VRM was due to the coincidence of a harmonic component of the radial force and a resonant frequency of the stator. This result showed that accurate knowledge of the vibrational behavior of the stator was essential to the prediction of the acoustic noise. Specifically, the transfer function between the radial forces acting on the stator poles and the resulting radial vibrations of the stator needed to be known at all frequencies in the audible range.

Finding this transfer function is an extremely difficult problem for several reasons. First, the radial forces acting on the stator are not concentrated at a single point. The exact areas of the stator poles that the forces act on depends on the position of the rotor. Second, it is difficult to derive equations of motion for the stator which satisfy not only the equilibrium and compatibility conditions, but also the complex set of boundary conditions presented by the stator and the parts of the motor which support it [25,27]. Third, adequately modeling losses in the stator poses a major problem, since some of the damping forces acting on the stator depend on the parts supporting it.

Therefore, instead of determining the transfer function outright, the analysis presented in this chapter determines only the resonant frequencies and modes of the stator assuming that the motor behaves as an elastically conservative system. Due to the assumption of a lossless system, the analysis will not be able to determine the width of the resonant peaks or the absolute amplitude of the vibrations. Approximations of the widths of the peaks could be obtained by experimentally determining an effective damping factor for the stator material. However, this still would not allow the absolute vibration amplitudes to be determined. The main usefulness of this analysis, therefore, will be to tell the motor designer the frequencies where the acoustic noise will occur and how these frequencies change when the size of the stator is changed.

The problem of predicting resonant frequencies of stators has been addressed by several authors using varying methods of analysis. The analysis presented in this chapter uses the method described in [10] and [23] with some modifications. It is based on the three dimensional theory of elasticity found in [25]. The well-known Rayleigh-Ritz method [13] along with Lagrange's equation are used to derive the frequency equation of the stator. The frequency equation is derived as a function of parameters which describe the stator's geometry, material composition, and elastic properties. This allows the variation of the resonant frequencies as a function of each parameter to be determined. Due to its transcendental form, the frequency equation is solved numerically using a computer.

## 4.2 The Rayleigh-Ritz Method

The Rayleigh-Ritz method is based on the premise that a finite number of independent position-dependent deflection functions $\vec{\phi}^0(\vec{p}), \vec{\phi}^1(\vec{p}), \ldots, \vec{\phi}^{N-1}(\vec{p})$ may be used to provide an approximation to the exact natural modes of a conservative system which are usually difficult or impossible to find. Although the number of functions used is arbitrary, it is agreed that, in a manner similar to the Fourier series synthesis of a function, a larger number of functions leads to more accurate results at the expense of greater computational effort.

In choosing deflection functions for use in the Rayleigh-Ritz method, there is one important requirement that should be taken into consideration. If the natural modes determined by this method are to satisfy the prescribed boundary conditions, then all of the chosen deflection functions must individually satisfy them. Having the natural modes satisfy the boundary conditions is important, because the natural frequencies of a given structure under one set of boundary conditions can be entirely different from those of the same structure under a different set of conditions. Usually, it is inconvenient to satisfy all of the boundary conditions, and it is common practice to ignore some of the constraints. However, for complete confidence in the results, the Rayleigh-Ritz method requires that all boundary conditions be met.

The application of Lagrange's equation to conservative systems involves the use of time-dependent energy functions. This means that the displacement function $\vec{\mathcal{U}}$ in general will be a function of position and time. In the Rayleigh-Ritz method, $\vec{\mathcal{U}}$ is approximated as

$$\vec{\mathcal{U}}(\vec{p},t) = \sum_{g=0}^{N-1} \vec{\phi}^{g}(\vec{p}) q_g(t) \tag{4.1}$$

where $N$ is the number of deflection functions, $\vec{p}$ is the position vector, and $q_g(t)$ are the time-dependent generalized displacement coordinates. The following form is assumed for the generalized coordinates:

$$q_g(t) = Q_g \cos(\omega t) \tag{4.2}$$

where $Q_g$ are constant amplitudes and $\omega$ is the frequency of vibration. Equation 4.1 is inserted into expressions for the kinetic and potential energies of the system. The resulting expressions are substituted into the following form of Lagrange's equation [19]:

$$\frac{d}{dt}\frac{\partial(KE)}{\partial \dot{q}_g} + \frac{\partial(PE)}{\partial q_g} = 0 \tag{4.3}$$

where KE and PE are expressions, in terms of Equations 4.1 and 4.2, for the kinetic and potential energies respectively. Equation 4.3 is evaluated for each of the N generalized coordinates $\{q_0, \ldots, q_{N-1}\}$ yielding N linear equations which can be written in matrix form as

$$\mathbf{A}\mathbf{x} = \omega^2 \mathbf{B}\mathbf{x} \tag{4.4}$$

where **A** is a symmetric N by N matrix, **B** is a symmetric positive definite N by N matrix and

$$\mathbf{x} = [Q_0 \, Q_1 \, \cdots \, Q_{N-1}]^T \tag{4.5}$$

Equation 4.4 is referred to as the generalized eigenvalue problem, and numerical solutions to it can be found using either MATLAB [18], an interactive program that performs matrix computations or the EISPACK [8,24] and LINPACK routines [3]. Solutions of Equation 4.4 yield a set of approximations to the natural frequencies of the system $\omega_0, \omega_1, \ldots, \omega_{N-1}$ and a set of eigenvectors $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{N-1}$. An approximation to the g'th natural mode of the system can be obtained using the following equation:

$$\vec{U}_g(\vec{p}) = \mathbf{x}_g^T \mathbf{y} \tag{4.6}$$

where

$$\mathbf{y} = [\vec{\phi}^0(\vec{p}) \, \vec{\phi}^1(\vec{p}) \, \cdots \, \vec{\phi}^{N-1}(\vec{p})]^T \tag{4.7}$$

## 4.3 Determination of the Deflection Functions

In applying the Rayleigh-Ritz method to the experimental VRM, several simplifying assumptions are made in the analysis:

- The stator is treated as a thick cylinder. The presence of the stator poles is ignored.

- All displacements are infinitesimal in comparison to the dimensions of the stator.
- The stator lamination stack is homogeneous, isotropic, linearly elastic, and compressible.
- There are no surface forces acting on the stator; the endbells and windings impose negligible constraints.

Figure 4.1 shows the dimensions of the stator and the coordinates used to describe it. In the analysis, all lengths are made dimensionless by dividing them by the outer radius $r_2$.

The components of the displacement distribution function $\vec{\mathcal{U}}$ have the following form in cylindrical coordinates:

$$\begin{aligned}
\mathcal{U}_r(r,\theta,z,t) &= \sum_{g=0}^{N-1} \phi_r^g(r,\theta,z) q_g(t) \\
\mathcal{U}_\theta(r,\theta,z,t) &= \sum_{g=0}^{N-1} \phi_\theta^g(r,\theta,z) q_g(t) \qquad (4.8) \\
\mathcal{U}_z(r,\theta,z,t) &= \sum_{g=0}^{N-1} \phi_z^g(r,\theta,z) q_g(t)
\end{aligned}$$

where $\mathcal{U}_r$, $\mathcal{U}_\theta$, and $\mathcal{U}_z$ are the radial, tangential, and axial components of displacement, respectively. The deflection functions are approximated by a double power series having the following form:

$$\phi_r(r,\theta,z) = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} a_{ij}\, r^j z^i \cos(\eta\theta)$$

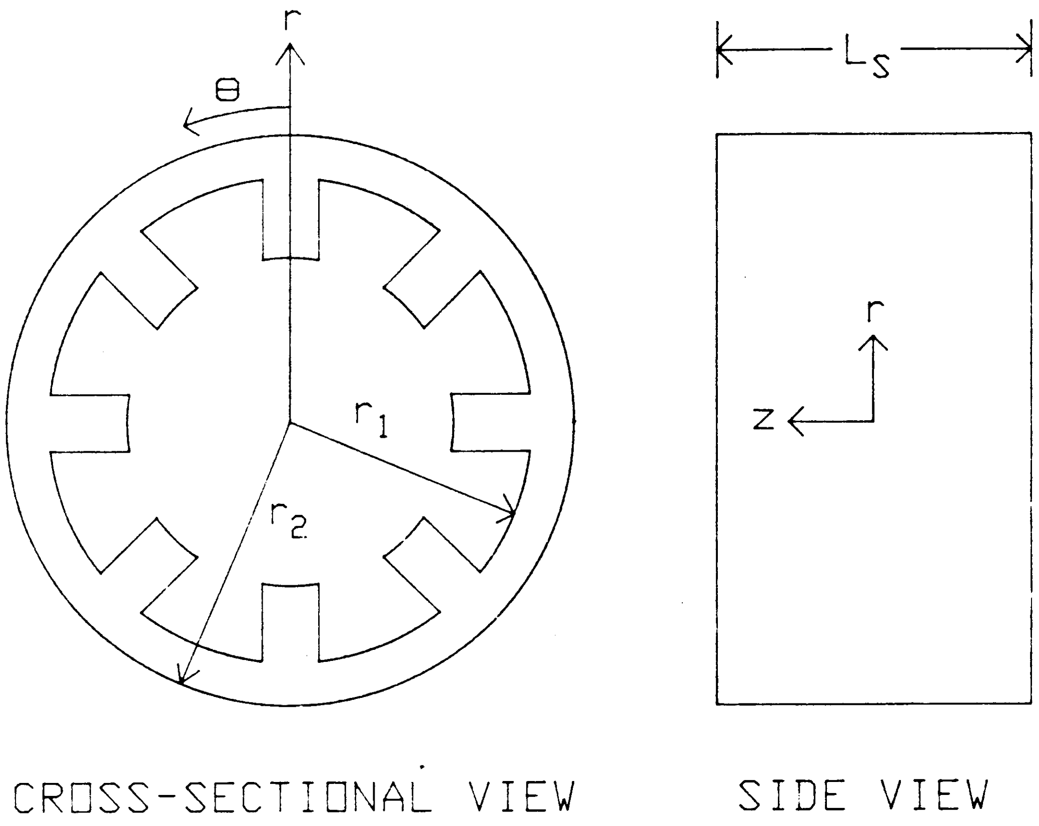

Figure 4.1: Dimensions and coordinates for the stator.

$$\phi_\theta(r,\theta,z) = \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} b_{ij}\, r^j\, z^i \sin(\eta\theta) \tag{4.9}$$

$$\phi_z(r,\theta,z) = \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} c_{ij}\, r^j\, z^i \cos(\eta\theta)$$

where $\eta$ is the circumferential mode number, $M$ is an integer which determines the size and accuracy of the model, and $a_{ij}$, $b_{ij}$, and $c_{ij}$ are expansion coefficients. As stated earlier, in order for the natural modes to satisfy the boundary conditions of the problem, all of the deflection functions must satisfy them. The boundary conditions of the stator under the simplifying assumptions made above are

$$\sigma_r(r,\theta,z) = \frac{2\bar{G}}{1-2\mu}\left[(1-\mu)\frac{\partial\phi_r}{\partial r} + \mu\left(\frac{\phi_r}{r} + \frac{\partial\phi_\theta}{r\partial\theta} + \frac{\partial\phi_z}{\partial z}\right)\right] = 0, \qquad \text{at } r = r_n,\ 1 \tag{4.10}$$

and

$$\sigma_z(r,\theta,z) = \frac{2\bar{G}}{1-2\mu}\left[(1-\mu)\frac{\partial\phi_z}{\partial z} + \mu\left(\frac{\partial\phi_r}{\partial r} + \frac{\phi_r}{r} + \frac{\partial\phi_\theta}{r\partial\theta}\right)\right] = 0, \qquad \text{at } z = \pm l_n \tag{4.11}$$

where $\sigma_r$ and $\sigma_z$ are the components of normal stress in the radial and axial directions, respectively, $r_n = r_1/r_2$ is the normalized length of the inner radius of the stator, $l_n = L_s/2r_2$ is half the normalized length of the stator, $\bar{G} = r_2 G$ is the normalized shear modulus of elasticity, and $\mu$ is Poisson's ratio. These two equations simply state that there are no forces acting on any surfaces of the stator cylinder. Substituting Equation 4.9 into

Equation 4.10 yields

$$\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}\left[a_{ij}\,(j+\kappa)\,r_n^{j-1}\,z^i + b_{ij}\,\kappa\,\eta\,r_n^{j-1}\,z^i + c_{ij}\,\kappa\,i\,r_n^j\,z^{i-1}\right]\cos(\eta\theta) = 0 \tag{4.12}$$

and

$$\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}\left[a_{ij}\,(j+\kappa)\,z^i + b_{ij}\,\kappa\,\eta\,z^i + c_{ij}\,\kappa\,i\,z^{i-1}\right]\cos(\eta\theta) = 0 \tag{4.13}$$

where $\kappa = \mu/(1-\mu)$. Equations 4.12 and 4.13 are homogeneous polynomials in z; the theta dependence cancels out. Since these polynomial must be satisfied for $-l_n \le z \le l_n$, their $M$ coefficients, which are linear functions of the expansion coefficients $a_{ij}$, $b_{ij}$, and $c_{ij}$, must be equal to zero. This leads to $2M$ homogeneous equations.

In a similar manner, substituting Equation 4.9 into Equation 4.11 yields

$$\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}\left[a_{ij}\,\kappa\,(j+1)\,r^{j-1}\,l_n^i + b_{ij}\,\kappa\,\eta\,r^{j-1}\,l_n^i + c_{ij}\,i\,r^j\,l_n^{i-1}\right]\cos(\eta\theta) = 0 \tag{4.14}$$

and

$$\sum_{i=0}^{M-1}\sum_{j=0}^{M-1}\left[a_{ij}\,\kappa\,(j+1)\,r^{j-1}\,(-l_n)^i + b_{ij}\,\kappa\,\eta\,r^{j-1}\,(-l_n)^i + c_{ij}\,i\,r^j\,(-l_n)^{i-1}\right]\cos(\eta\theta) = 0 \tag{4.15}$$

Equations 4.14 and 4.15 are homogeneous polynomials in r which must be satisfied for $r_n \le r \le 1$. Again, this means that the coefficients of both polynomials are zero, leading to $2M+2$ homogeneous equations. These equations along with the $2M$ equations obtained from Equations 4.12 and 4.13 can be put into the following form:

$$\mathbf{Ax} = \mathbf{0} \tag{4.16}$$

where $\mathbf{A}$ is a $(4M+2)$ by $(3M^2)$ matrix and

$$\mathbf{x} = [\leftarrow a_{ij} \rightarrow \leftarrow b_{ij} \rightarrow \leftarrow c_{ij} \rightarrow]^T \tag{4.17}$$

is a $3M^2$ element column vector of the expansion coefficients of the deflection functions. Thus the problem of determining the deflection functions of Equation 4.9 which satisfy the boundary conditions reduces to finding vectors in the null space of $\mathbf{A}$. This is easily accomplished using MATLAB, which can find the complete set of independent vectors that satisfy Equation 4.16. Assuming that $\mathbf{A}$ has full rank, the maximum number of independent vectors that can be obtained for a given value of $M \geq 2$ is $3M^2 - 4M - 2$. The number of vectors in the null space of A specifies $N$ in Equation 4.8. The $N$ deflection functions can now be expressed as

$$\begin{aligned}
\phi_r^g(r,\theta,z) &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} a_{ij}^g \, r^j z^i \cos(\eta\theta) \\
\phi_\theta^g(r,\theta,z) &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} b_{ij}^g \, r^j z^i \sin(\eta\theta) \\
\phi_z^g(r,\theta,z) &= \sum_{i=0}^{M-1}\sum_{j=0}^{M-1} c_{ij}^g \, r^j z^i \cos(\eta\theta)
\end{aligned} \tag{4.18}$$

where $\overset{g}{a}_{ij}$, $\overset{g}{b}_{ij}$, and $\overset{g}{c}_{ij}$ are the expansion coefficients taken from the g'th null space vector. These functions form the set of all possible deflection functions of the form shown in Equation 4.9 that satisfy the boundary conditions given in Equations 4.10 and 4.11. Because of this, the displacement distribution function in Equation 4.8 will be able to provide the best approximation to the exact natural modes of the stator for a given value of $M$.

## 4.4 Determination of the Kinetic and Potential Energies of the Stator

The kinetic energy of the stator cylinder can be obtained from the following equation [10,13]:

$$KE = \frac{\bar{\rho}_s}{2} \int_{-l_n}^{l_n} \int_0^{2\pi} \int_{r_n}^{1} \left[ \left( \frac{\partial \mathcal{U}_r}{\partial t} \right)^2 + \left( \frac{\partial \mathcal{U}_\theta}{\partial t} \right)^2 + \left( \frac{\partial \mathcal{U}_z}{\partial t} \right)^2 \right] r dr d\theta dz \tag{4.19}$$

where $\bar{\rho}_s = r_2^3 \rho_s$ is the normalized mass density of the stator. Substituting Equation 4.8 into Equation 4.19 and changing the order of summation and integration yields

$$KE = \frac{\bar{\rho}_s}{2} \sum_{g=0}^{N-1} \sum_{h=0}^{N-1} \dot{q}_g \dot{q}_h \int_{-l_n}^{l_n} \int_0^{2\pi} \int_{r_n}^{1} \left( \phi_r^g \phi_r^h + \phi_\theta^g \phi_\theta^h + \phi_z^g \phi_z^h \right) r dr d\theta dz \tag{4.20}$$

Using the integrals given in Appendix A, this equation simplifies to

$$KE = \frac{\bar{\rho}_s}{2} \sum_{gh}^{N-1} \dot{q}_g \dot{q}_h \sum_{ijkl}^{M-1} \left( \overset{g}{a}_{ij} \overset{h}{a}_{kl} \ \mathcal{R}_1 \mathcal{P}_1 \mathcal{Z}_1 + \overset{g}{b}_{ij} \overset{h}{b}_{kl} \ \mathcal{R}_1 \mathcal{P}_2 \mathcal{Z}_1 + \overset{g}{c}_{ij} \overset{h}{c}_{kl} \ \mathcal{R}_1 \mathcal{P}_1 \mathcal{Z}_1 \right) \tag{4.21}$$

The potential or strain energy of the stator cylinder can be determined from the following equation [10,25]:

$$PE = \frac{1}{2}\int_{-l_n}^{l_n}\int_{0}^{2\pi}\int_{r_n}^{1}(\sigma_r\epsilon_r + \sigma_\theta\epsilon_\theta + \sigma_z\epsilon_z + \tau_{r\theta}\gamma_{r\theta} + \tau_{rz}\gamma_{rz} + \tau_{\theta z}\gamma_{\theta z})\, r dr d\theta dz \tag{4.22}$$

where the strains are given by

$$\begin{array}{lll} \epsilon_r = \frac{\partial \mathcal{U}_r}{\partial r} & \epsilon_\theta = \frac{\mathcal{U}_r}{r} + \frac{\partial \mathcal{U}_\theta}{r\partial\theta} & \epsilon_z = \frac{\partial \mathcal{U}_z}{\partial z} \\ \gamma_{r\theta} = \frac{\partial \mathcal{U}_r}{r\partial\theta} + \frac{\partial \mathcal{U}_\theta}{\partial r} - \frac{\mathcal{U}_\theta}{r} & \gamma_{rz} = \frac{\partial \mathcal{U}_r}{\partial z} + \frac{\partial \mathcal{U}_z}{\partial r} & \gamma_{\theta z} = \frac{\partial \mathcal{U}_\theta}{\partial z} + \frac{\partial \mathcal{U}_z}{r\partial\theta} \end{array} \tag{4.23}$$

and the stresses are given by

$$\begin{array}{ll} \sigma_r = \frac{2\bar{G}}{1-2\mu}\left[(1-\mu)\epsilon_r + \mu(\epsilon_\theta + \epsilon_z)\right] & \tau_{r\theta} = \bar{G}\gamma_{r\theta} \\ \sigma_\theta = \frac{2\bar{G}}{1-2\mu}\left[(1-\mu)\epsilon_\theta + \mu(\epsilon_r + \epsilon_z)\right] & \tau_{rz} = \bar{G}\gamma_{rz} \\ \sigma_z = \frac{2\bar{G}}{1-2\mu}\left[(1-\mu)\epsilon_z + \mu(\epsilon_r + \epsilon_\theta)\right] & \tau_{\theta z} = \bar{G}\gamma_{\theta z} \end{array} \tag{4.24}$$

The term-by-term integration of Equation 4.22 is found in Appendix A. The result of that integration yields:

$$PE = \bar{G}\sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1} \Bigg[$$

$$\overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\left\{\left[\eta^2\mathcal{P}_2 + 2\mathcal{P}_1\frac{\mu(j+l)+(1-\mu)(1+jl)}{1-2\mu}\right]\mathcal{R}_3\frac{\mathcal{Z}_1}{2} + ik\,\mathcal{R}_1\mathcal{P}_1\frac{\mathcal{Z}_3}{2}\right\}$$

$$+\ \overset{g}{a}_{ij}\,\overset{h}{b}_{kl}\left\{2\eta\left[(1-l)\,\mathcal{P}_2 + 2\mathcal{P}_1\frac{\mu j+(1-\mu)}{1-2\mu}\right]\mathcal{R}_3\frac{\mathcal{Z}_1}{2}\right\}$$

$$+ \overset{g}{a}_{ij} \overset{h}{c}_{kl} \left\{ 2 \left[ il + \frac{2\mu k(1+j)}{1-2\mu} \right] \mathcal{R}_2 \mathcal{P}_1 \frac{\mathcal{Z}_2}{2} \right\}$$

$$+ \overset{g}{b}_{ij} \overset{h}{b}_{kl} \left\{ \left[ \frac{2\eta^2(1-\mu)}{1-2\mu} \mathcal{P}_1 + (j-1)(l-1)\mathcal{P}_2 \right] \mathcal{R}_3 \frac{\mathcal{Z}_1}{2} + ik\,\mathcal{R}_1 \mathcal{P}_2 \frac{\mathcal{Z}_3}{2} \right\}$$

$$+ \overset{g}{b}_{ij} \overset{h}{c}_{kl} \left\{ 2\eta \left[ \frac{2\mu k}{1-2\mu} \mathcal{P}_1 - i\,\mathcal{P}_2 \right] \mathcal{R}_2 \frac{\mathcal{Z}_2}{2} \right\}$$

$$+ \overset{g}{c}_{ij} \overset{h}{c}_{kl} \left\{ \left[ \frac{2(1-\mu)ik}{1-2\mu} \right] \mathcal{R}_1 \mathcal{P}_1 \frac{\mathcal{Z}_3}{2} + \left[ jl\,\mathcal{P}_1 + \eta^2 \mathcal{P}_2 \right] \mathcal{R}_3 \frac{\mathcal{Z}_1}{2} \right\} \Bigg] \tag{4.25}$$

Equations 4.21 and 4.25 can be written in a more concise form as follows:

$$KE = \bar{\rho}_s \sum_{gh}^{N-1} \dot{q}_g \dot{q}_h \sum_{ijkl}^{M-1} \left[ \overset{g}{a}_{ij} \overset{h}{a}_{kl} \overset{7}{\mathcal{D}}_{ijkl} + \overset{g}{b}_{ij} \overset{h}{b}_{kl} \overset{8}{\mathcal{D}}_{ijkl} + \overset{g}{c}_{ij} \overset{h}{c}_{kl} \overset{7}{\mathcal{D}}_{ijkl} \right] \tag{4.26}$$

$$PE = \bar{G} \sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1} \Bigg[ \overset{g}{a}_{ij} \overset{h}{a}_{kl} \overset{1}{\mathcal{D}}_{ijkl} + \overset{g}{a}_{ij} \overset{h}{b}_{kl} \overset{2}{\mathcal{D}}_{ijkl} + \overset{g}{a}_{ij} \overset{h}{c}_{kl} \overset{3}{\mathcal{D}}_{ijkl}$$

$$+ \overset{g}{b}_{ij} \overset{h}{b}_{kl} \overset{4}{\mathcal{D}}_{ijkl} + \overset{g}{b}_{ij} \overset{h}{c}_{kl} \overset{5}{\mathcal{D}}_{ijkl} + \overset{g}{c}_{ij} \overset{h}{c}_{kl} \overset{6}{\mathcal{D}}_{ijkl} \Bigg] \tag{4.27}$$

where

$$\overset{1}{\mathcal{D}}_{ijkl} = \left[ \eta^2 \mathcal{P}_2 + 2\mathcal{P}_1 \frac{\mu(j+l) + (1-\mu)(1+jl)}{1-2\mu} \right] \mathcal{R}_3 \frac{\mathcal{Z}_1}{2} + ik\,\mathcal{R}_1 \mathcal{P}_1 \frac{\mathcal{Z}_3}{2}$$

$$\overset{2}{\mathcal{D}}_{ijkl} = 2\eta\left[(1-l)\,\mathcal{P}_2 + 2\mathcal{P}_1\frac{\mu j + (1-\mu)}{1-2\mu}\right]\mathcal{R}_3\frac{\mathcal{Z}_1}{2}$$

$$\overset{3}{\mathcal{D}}_{ijkl} = 2\left[il + \frac{2\mu k(1+j)}{1-2\mu}\right]\mathcal{R}_2\mathcal{P}_1\frac{\mathcal{Z}_2}{2}$$

$$\overset{4}{\mathcal{D}}_{ijkl} = \left[\frac{2\eta^2(1-\mu)}{1-2\mu}\,\mathcal{P}_1 + (j-1)(l-1)\,\mathcal{P}_2\right]\mathcal{R}_3\frac{\mathcal{Z}_1}{2} + ik\,\mathcal{R}_1\mathcal{P}_2\frac{\mathcal{Z}_3}{2}$$

$$\overset{5}{\mathcal{D}}_{ijkl} = 2\eta\left[\frac{2\mu k}{1-2\mu}\,\mathcal{P}_1 - i\,\mathcal{P}_2\right]\mathcal{R}_2\frac{\mathcal{Z}_2}{2}$$

$$\overset{6}{\mathcal{D}}_{ijkl} = \left[\frac{2(1-\mu)ik}{1-2\mu}\right]\mathcal{R}_1\mathcal{P}_1\frac{\mathcal{Z}_3}{2} + \left[jl\,\mathcal{P}_1 + \eta^2\mathcal{P}_2\right]\mathcal{R}_3\frac{\mathcal{Z}_1}{2}$$

$$\overset{7}{\mathcal{D}}_{ijkl} = \mathcal{R}_1\mathcal{P}_1\frac{\mathcal{Z}_1}{2}$$

$$\overset{8}{\mathcal{D}}_{ijkl} = \mathcal{R}_1 \mathcal{P}_2 \frac{\mathcal{Z}_1}{2}$$

## 4.5 Determination of the Frequency Equation of the Stator

The frequency equation of the stator is obtained by substituting Equations 4.21 and 4.25 into Lagrange's equation for each of the $N$ generalized coordinates. Equation 4.3 evaluated for the $\bar{g}$'th coordinate yields

$$\sum_{g}^{N-1} q_g \sum_{ijkl}^{M-1} \Bigg[ 2\, \overset{g}{a}_{ij}\, \overset{\bar{g}}{a}_{kl}\, \overset{1}{\mathcal{D}}_{ijkl} + \left( \overset{g}{a}_{ij}\, \overset{\bar{g}}{b}_{kl} + \overset{\bar{g}}{a}_{ij}\, \overset{g}{b}_{kl} \right) \overset{2}{\mathcal{D}}_{ijkl} + \left( \overset{g}{a}_{ij}\, \overset{\bar{g}}{c}_{kl} + \overset{\bar{g}}{a}_{ij}\, \overset{g}{c}_{kl} \right) \overset{3}{\mathcal{D}}_{ijkl}$$

$$+ 2\, \overset{g}{b}_{ij}\, \overset{\bar{g}}{b}_{kl}\, \overset{4}{\mathcal{D}}_{ijkl} + \left( \overset{g}{b}_{ij}\, \overset{\bar{g}}{c}_{kl} + \overset{\bar{g}}{b}_{ij}\, \overset{g}{c}_{kl} \right) \overset{5}{\mathcal{D}}_{ijkl} + 2\, \overset{g}{c}_{ij}\, \overset{\bar{g}}{c}_{kl}\, \overset{6}{\mathcal{D}}_{ijkl} \Bigg] =$$

$$\sum_{g}^{N-1} q_g \sum_{ijkl}^{M-1} 2 \left[ \left( \overset{g}{a}_{ij}\, \overset{\bar{g}}{a}_{kl} + \overset{g}{c}_{ij}\, \overset{\bar{g}}{c}_{kl} \right) \overset{7}{\mathcal{D}}_{ijkl} + \overset{g}{b}_{ij}\, \overset{\bar{g}}{b}_{kl}\, \overset{8}{\mathcal{D}}_{ijkl} \right] \left( \frac{\omega}{\omega_o} \right)^2 \tag{4.28}$$

where $\omega_o = \sqrt{\bar{G}/\bar{\rho}_s}$. As stated earlier, the $N$ equations obtained from Equation 4.28 can be written in matrix form as

$$\mathbf{Ax} = \omega^2 \mathbf{Bx} \tag{4.29}$$

where the element in row $\alpha$ and column $\beta$ of $\mathbf{A}$ is given by

$$\mathbf{A}_{\alpha,\beta} = \sum_{ijkl}^{M-1} \Bigg[ 2\, \overset{\beta}{a}_{ij}\, \overset{\alpha}{a}_{kl}\, \overset{1}{\mathcal{D}}_{ijkl} + \left( \overset{\beta}{a}_{ij}\, \overset{\alpha}{b}_{kl} + \overset{\alpha}{a}_{ij}\, \overset{\beta}{b}_{kl} \right) \overset{2}{\mathcal{D}}_{ijkl} + \left( \overset{\beta}{a}_{ij}\, \overset{\alpha}{c}_{kl} + \overset{\alpha}{a}_{ij}\, \overset{\beta}{c}_{kl} \right) \overset{3}{\mathcal{D}}_{ijkl}$$

$$+ 2\, \overset{\beta}{b}_{ij}\, \overset{\alpha}{b}_{kl}\, \overset{4}{\mathcal{D}}_{ijkl} + \left( \overset{\beta}{b}_{ij}\, \overset{\alpha}{c}_{kl} + \overset{\alpha}{b}_{ij}\, \overset{\beta}{c}_{kl} \right) \overset{5}{\mathcal{D}}_{ijkl} + 2\, \overset{\beta}{c}_{ij}\, \overset{\alpha}{c}_{kl}\, \overset{6}{\mathcal{D}}_{ijkl} \Bigg] \tag{4.30}$$

and the element in row $\alpha$ and column $\beta$ of $\mathbf{B}$ is given by

$$\mathbf{B}_{\alpha,\beta} = \sum_{ijkl}^{M-1} 2 \left[ \left( \overset{\beta}{a}_{ij}\, \overset{\alpha}{a}_{kl} + \overset{\beta}{c}_{ij}\, \overset{\alpha}{c}_{kl} \right) \overset{7}{\mathcal{D}}_{ijkl} + \overset{\beta}{b}_{ij}\, \overset{\alpha}{b}_{kl}\, \overset{8}{\mathcal{D}}_{ijkl} \right] \tag{4.31}$$

As mentioned earlier, both $\mathbf{A}$ and $\mathbf{B}$ in Equation 4.29, which are the equivalent stiffness and inertia matrices of the stator cylinder, respectively, are symmetric and positive definite [19].

## 4.6 Results

### 4.6.1 Resonant Frequencies and Mode Shapes

The dimensions of the experimental VRM and the material parameters used to calculate the resonant frequencies are shown in Table 4.1. The values chosen for $G$, $\mu$, and $\rho$ are those of mild steel; the actual values depend on the annealing process and the properties of the insulation between the laminations.

The resonant frequencies and mode shapes are determined using a combination of MATLAB [18] and Microsoft C 5.0 [20] routines which are listed in Appendix 4. The main driver program for these routines is the MATLAB M-file RES.M which, in addition to the pa-

| parameter | value |
|---|---|
| $G$ | $8.09 \times 10^{10}$ Pa |
| $\mu$ | .28 |
| $\rho$ | $7860\,\mathrm{kg}/m^3$ |
| $r_1$ | 4.06 cm |
| $r_2$ | 4.69 cm |
| $L_s$ | 5.08 cm |

Table 4.1: Dimensions and material parameters of the experimental VRM.

rameters listed in Table 4.1, must be supplied with the model size parameter $M$ and the desired circumferential mode number $\eta$. Due to numerical round-off problems, the maximum model size parameter that can be used is $M = 4$. Increasing $M$ beyond this point causes the matrix $\mathbf{B}$ to no longer be positive definite.

Table 4.2 shows the resonant frequencies for the first six modes of the experimental VRM as determined by the analysis with the model size parameter $M = 4$; only the frequencies below 20 KHz are shown. The mode shapes associated with each of these frequencies can be determined by evaluating Equation 4.6 at several points on the surfaces of the stator cylinder. This allows a picture of the stator to be developed at an instant in time when all deflections (radial, tangential, and axial) are at a maximum. From this picture, the longitudinal symmetry of the mode can be determined. Longitudinal symmetry is defined in terms of the radial displacement of the outer surface $r = 1$ along the length of the stator. Even and odd symmetry are defined as $\mathcal{U}_r(1, \theta, z) = \mathcal{U}_r(1, \theta, -z)$ and $\mathcal{U}_r(1, \theta, z) =$

| circumferential mode number $\eta$ | frequency in Hertz | |
|---|---|---|
| | longitudinal symmetry | |
| | even | odd |
| 0 | 18603 | 18548 |
| 1 | – | 16902 |
| 2 | 2127 | 3520 |
| 3 | 5936 | 8665 |
| 4 | 11168 | 14663 |
| 5 | 17633 | – |

Table 4.2: Calculated resonant frequencies of the experimental VRM.

$-\mathcal{U}_r(1, \theta, -z)$, respectively.

Some examples for the calculated frequency of 2127 Hz are shown in Figures 4.2 and 4.3. Figure 4.2 shows the radial, tangential, and axial displacements of the outer radial surface of the stator as a function of axial position. The displacements are evaluated at Phase A's location $\theta = 0$ and have been normalized so that the maximum displacement is equal to one. Figure 4.2 shows that the radial vibration amplitude changes very little along the length of the stator. This is in agreement with the the acceleration measurements performed on the experimental VRM as discussed in Chapter 3. Figure 4.2 also shows that the outer surface is expanding and contracting in the axial direction, since for positive values of $z$ the axial displacement $\mathcal{U}_z$ is positive and for negative values of $z$ it is negative. A half of a cycle later, the opposite will be true. The tangential displacement $\mathcal{U}_\theta$ is constant over the length showing that there is no twisting motion of the stator.

Figure 4.3 shows the radial, tangential, and axial displacements of the end surface of the stator at $z = +l_n$ as a function of radial position. The radial displacement $\mathcal{U}_r$ is constant meaning that there is no compression of the stator in the radial direction. The

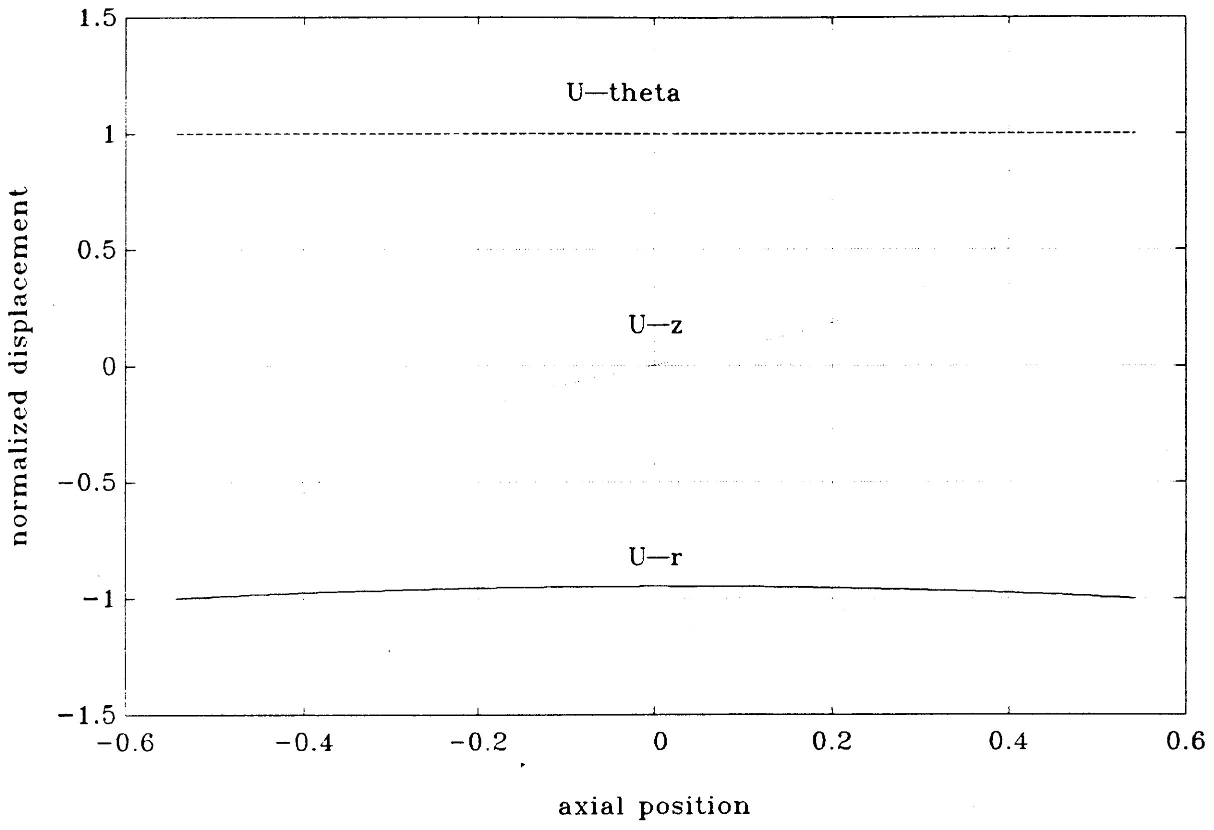

Figure 4.2: Normalized radial, tangential, and axial displacement of the outer radial surface of the stator as a function of axial position for the predicted resonant frequency at 2127 Hz.

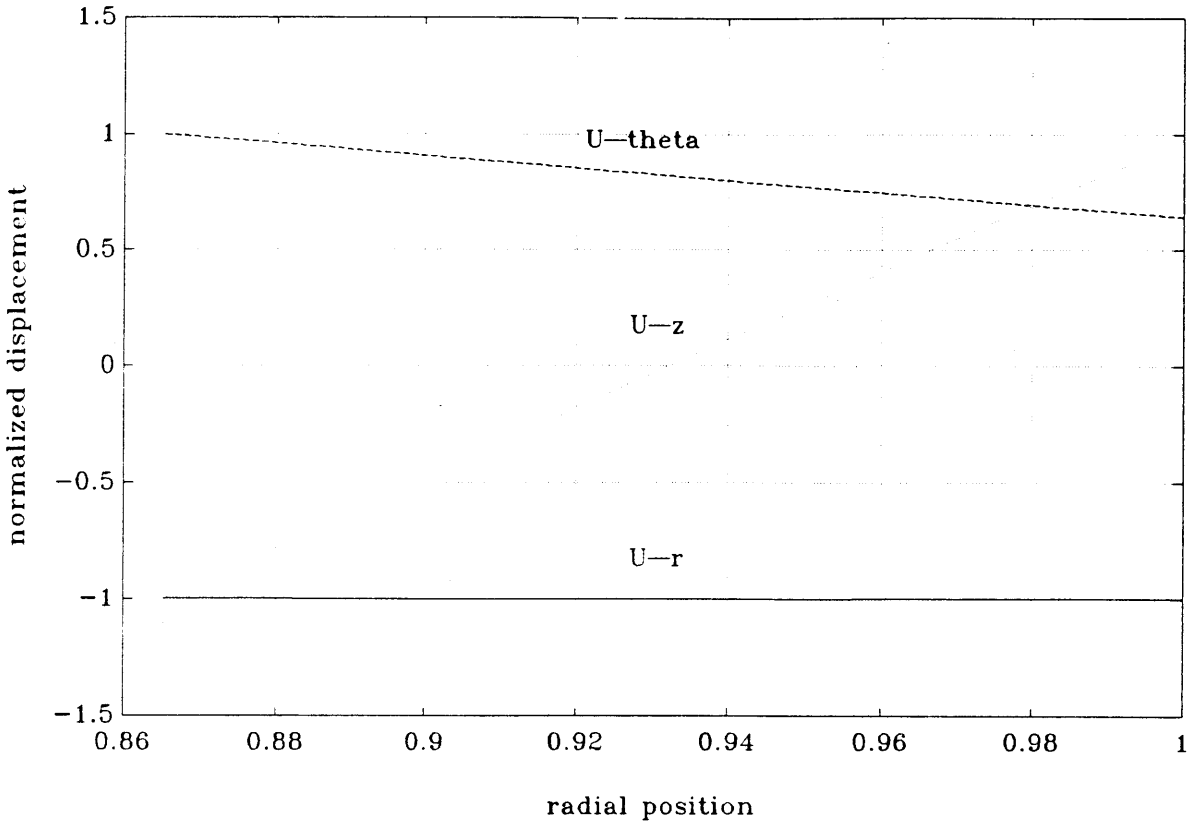

Figure 4.3: Normalized radial, tangential, and axial displacement of the axial surface of the stator at $z = +ln$ as a function of radial position for the predicted resonant frequency at 2127 Hz.

axial displacement shows that the ends of the stator, which would be in contact with the endbells in the case of the experimental VRM, are pivoting around the radial midpoint of the stator cylinder. This illustrates one of the problems of including the endbells in the model for the stator. Because of this rocking motion, the area of contact between the endbells and the ends of the stator would change with time, meaning that the boundary condition at the stator ends would vary with time and position. The tangential displacement shows that there is shearing between the inner and outer radial surfaces of the stator.

Using the mode shapes and symmetry arguments, most of the frequencies given in Table 4.2 can be ruled out as being resonant frequencies of the stator of the experimental VRM. First, since the radial forces acting on the stator are essentially uniform in the axial direction except near the ends of the stator, resonant frequencies whose radial displacement $\mathcal{U}_r$ is not an even function of the axial position, will not be excited. Second, since the radial forces act simultaneously at diametrically opposing points on the stator, resonant frequencies corresponding to odd circumferential modes will not be excited. This leaves only three resonant frequencies at 18603 Hz, 2127 Hz, and 11168 Hz corresponding to the zeroth, second, and fourth order circumferential modes, respectively. In Chapter 3, these same three mode shapes were observed for the actual resonant frequencies of the experimental VRM. A comparison between the respective frequencies is given in Table 4.3. As this table shows, there is considerable discrepancy between the calculated and measured resonant frequencies of the stator, especially at the higher frequencies. This can be attributed mainly to the absence of the poles, windings, and endbells from the model and the small value used for $M$; a similar method of analysis that didn't account for the boundary conditions has been shown to be accurate to within a couple of percent for thick cylinders [23]. As noted above,

| circumferential | frequency in Hertz | | |
|---|---|---|---|
| mode number $\eta$ | calculated | measured | % error |
| 0 | 18603 | 14200 | +31.0 |
| 2 | 2127 | 2604 | -18.2 |
| 4 | 11168 | 9200 | +21.4 |

Table 4.3: Comparison between the calculated and measured resonant frequencies of the experimental VRM.

the value of $M$ is limited by numerical round-off problems. The particular value used for $G$, $\rho$, and $\mu$ does not effect the magnitude of the errors significantly since changing the first two simply scales all three resonant frequencies up or down by the same amount, and changing the third over the range of values for all steels causes very little change in the frequencies.

Because of the size of the errors and the sharpness of the resonant peaks measured for the experimental VRM, it will be necessary to perform a more accurate analysis of the stator which takes into accounts the poles, windings, and endbells, if the magnitude of the stator acceleration is to be predicted from geometric parameters. However, as mentioned at the beginning of the chapter, this places many more constraints on the form that the displacement function can take which further complicates the solution. Alternatives would be either to use finite-element analysis to determine the resonant frequencies or to simply measure them after the machine is built.

Despite its shortcomings, the analysis presented in this chapter is useful for giving a yes/no answer to the question of whether a potential noise problem exists for a given design. An important question that could be answered is whether a design would lead to a motor having resonant frequencies low enough to be excited by the magnetic forces in that motor. As shown in the next section, the analysis is also useful for determining trends in the resonant frequencies as geometric parameters are changed.

### 4.6.2 Effect of Dimension Changes on the Resonant Frequencies

As shown in Chapter 3, the acoustic noise problem of the experimental VRM is due to the coincidence of a harmonic of the radial force and a resonant frequency of the stator. Consequentially, if the resonant frequencies can be raised high enough by changing the dimensions of the stator, so that they are not excited by the major harmonic components of the radial force, the acoustic noise can be reduced. This section, therefore, presents figures showing the effect of changes changes in the dimensions of the stator on the three calculated resonant frequencies of 18603 Hz, 2127 Hz, and 11168 Hz corresponding to circumferential modes 0, 2, and 4, respectively. In Figure 4.4, the effect of changing the stator length while holding the inner and outer radii constant is shown. This figure shows that increasing the length causes the two lowest frequencies to increase and the highest one to decrease. The changes, however, are very small indicating that this is not a very effective way of reducing the acoustic noise.

Larger changes, however, are obtained when either the outer or inner radius is changed. Figure 4.5 shows the effect of changing the outer radius while holding the inner radius and axial length constant. In this case, the two lowest frequencies more than double when the outer radius is increased by 50%, while the highest one decreases slightly. Figure 4.6 shows the effect of changing the inner radius while holding the outer radius and axial length constant. As in the previous case, increasing the thickness of the stator (by decreasing the inner radius) causes the two lowest resonant frequencies to increase. The highest frequency,

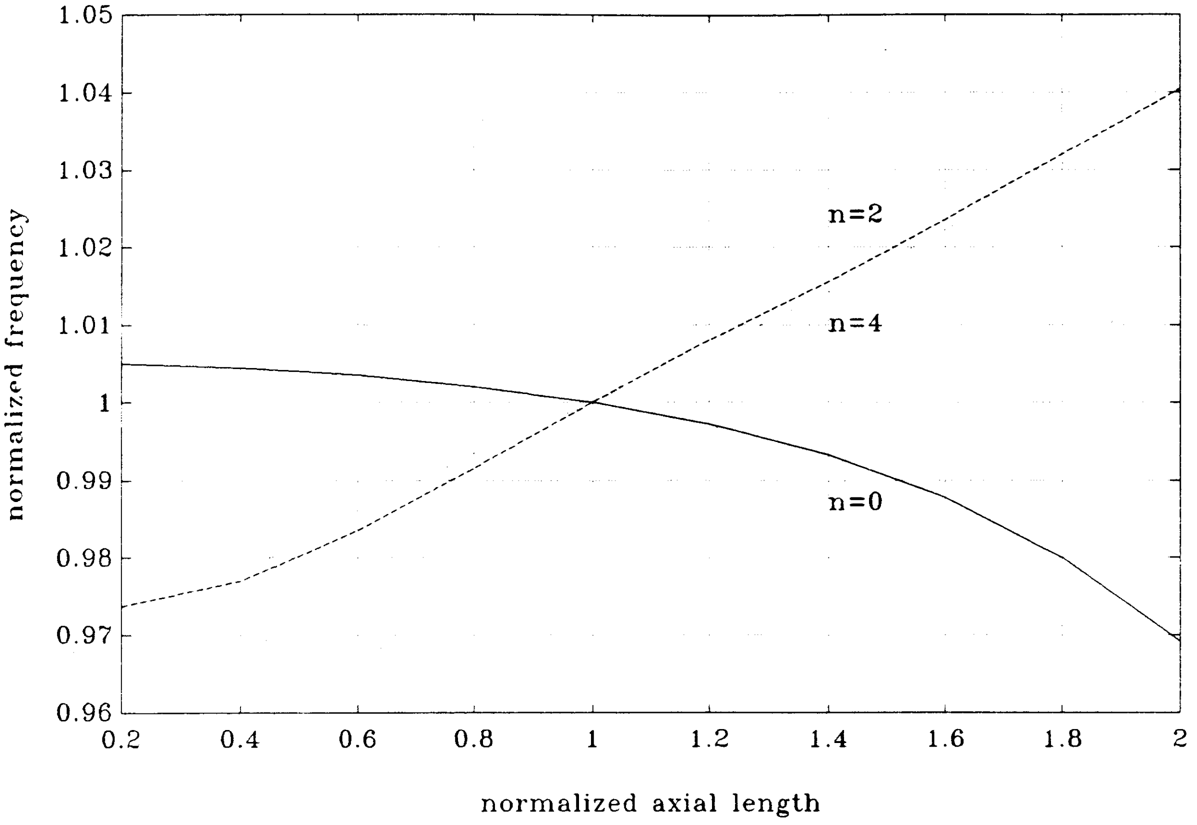

Figure 4.4: Effect of stator length on the resonant frequencies corresponding to circumferential modes 0, 2, and 4. The inner and outer radii are held constant.

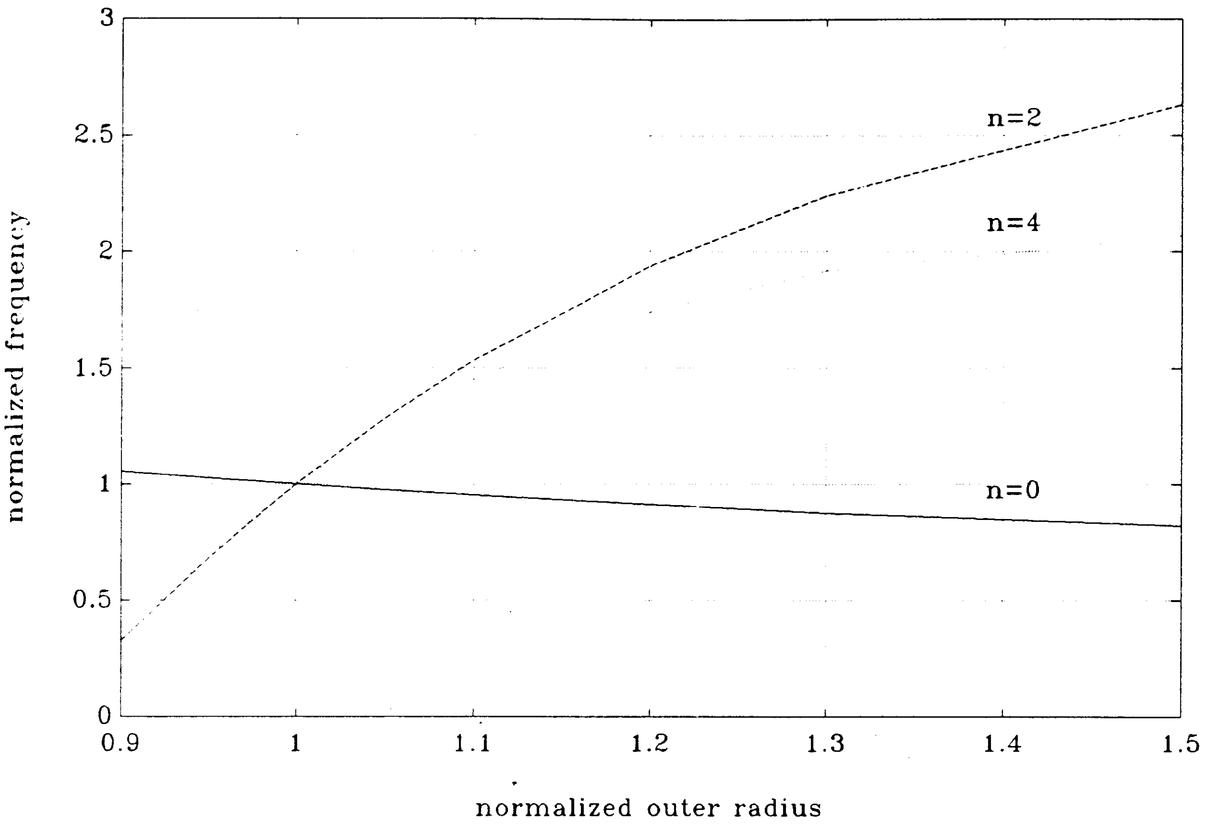

Figure 4.5: Effect of outer radius on the resonant frequencies corresponding to circumferential modes 0, 2, and 4. The inner radius and axial length are held constant.

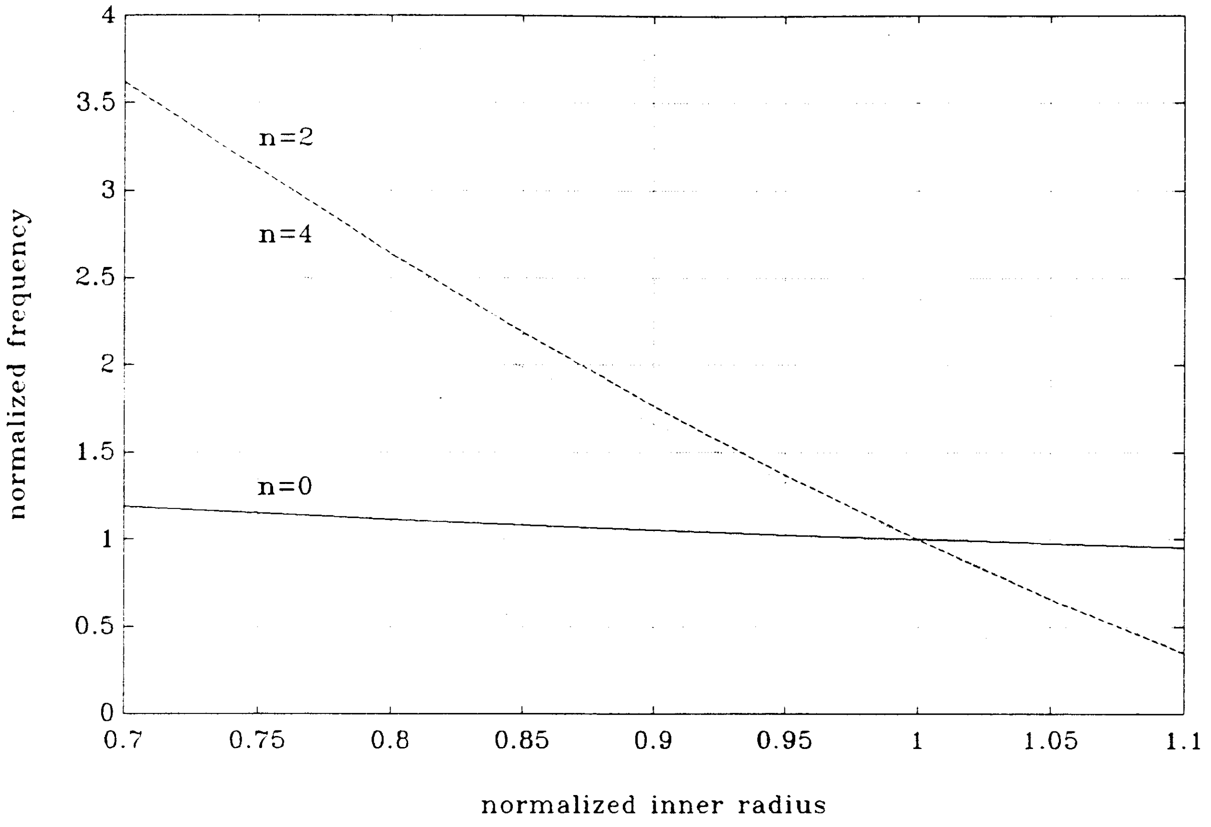

Figure 4.6: Effect of outer radius on the resonant frequencies corresponding to circumferential modes 0, 2, and 4. The outer radius and axial length are held constant.

however, increases as the thickness of the stator is increased, although the percent change is small as before. Both Figures 4.5 and 4.6 show that the most effective way to increase the resonant frequencies is to increase the thickness of the stator cylinder. However, this method might not be the most cost effective way of reducing the acoustic noise of a VRM, due to the amount of increase in thickness that might be needed. This point is illustrated below for the experimental VRM.

The experimental VRM was designed to operate at speeds up to 8000 rpm, which corresponds to a maximum pulse frequency of 800 Hz. In Chapter 3, the experiments revealed that harmonics of the radial force pulses as high as the seventh could excite the lowest resonant frequency of the stator and produce significant noise. Therefore, to prevent 7th harmonic excitation from occurring, the predicted resonant frequency at 2127 Hz would have to be raised above 5600 Hz, an increase by a factor of about 2.6. From Figure 4.5, such an increase would require that the outer radius, for example, be increased by 50increased by a factor of almost 5. Because such an increase would add significantly to the cost and size of the motor, a conclusion that can be drawn here is that changing the size of the stator is not a very practical means of reducing the acoustic noise of a VRM, and that other means such as current/radial force pulse shaping probably offer better possibilities.

# Chapter 5

# Prediction of Magnetic Forces in the VRM

As shown in Chapter 3, the two factors that determine the level of acoustic noise in the experimental VRM are the values of the resonant frequencies of the stator and the harmonic content of the radial forces. In Chapter 4, a model is presented for predicting the resonant frequencies and modes of the stator based on its length and its inner and outer radii. Using this model, the effect of changes in these three parameters on the resonant frequencies can be determined. However, once the geometry of the stator is chosen, the level of the acoustic noise depends on the shape of the radial force pulses. In this chapter, a model is presented that predicts the fluxes and currents in the VRM at a constant speed based on the electromechanical properties of the VRM, the inverter topology used to excite the phases of the VRM, and the control scheme. As will be shown in this chapter, by using the flux-current curves taken from the experimental VRM, the current and torque waveforms can be predicted in a straightforward manner allowing the peak current and average torque of the VRM to be accurately determined. The radial force waveform, however, can not be predicted accurately enough to determine the effect of changes in the operating parameters on the magnitude of the acoustic noise. The main reason for this is that the model of the

VRM used here is not detailed enough to allow the harmonic components of the radial force waveform as high as the seventh in the case of the experimental VRM, to be accurately predicted. As a result, more detailed modeling, such as would be available through finite-element analysis, appears necessary.

## 5.1 Constant-Speed Model of the VRM

As in Chapters 2 and 3, it is assumed that the phases of the VRM are identical and magnetically independent and that the fields in the air gap are radially oriented and uniform in the circumferential direction. The equations governing the operation of the experimental VRM at constant speed, taken from Chapters 2 and 3, are

$$\frac{d\lambda_n(t)}{dt} = -R_n i_n + v_n, \qquad n = 1,2,3,4 \tag{5.1}$$

$$\frac{d\theta(t)}{dt} = \omega_r \tag{5.2}$$

$$\tau_m(t) = \sum_{n=1}^{4} -\left.\frac{\partial W_n(\theta_n, \lambda_n)}{\partial \theta_n}\right|_{\lambda_n} \tag{5.3}$$

$$W_n(\theta_n, \lambda_n) = \int_0^{\lambda_n} i_n(\theta_n, \lambda_n')d\lambda_n' \tag{5.4}$$

$$\tau_{av} = \frac{\omega_r}{2\pi}\int_0^{2\pi/\omega_r} \tau_m dt \tag{5.5}$$

$$F_{r,n}(t) = -\frac{\mu_o\, A_{gap,n}(\theta_n)\, N^2\, i_n^2}{8g^2} \tag{5.6}$$

$$\theta_n = \theta + (n-1)\frac{2\pi}{N_r N_p} \tag{5.7}$$

where $W_n$ and $F_{r,n}$ are the energy and the radial force of the nth phase, respectively, $A_{gap,n}$ is the cross-sectional area of the gap between the stator poles of the nth phase and a pair of rotor poles, $\omega_r$ is the rotor speed which is assumed to be constant, and $\tau_{av}$ is the average torque.

### 5.1.1 Modeling the Flux-Current Relationship

Before Equations 5.1- ע.7 can be simulated, the current-versus-flux relationship needs to be known. In Chapters 2 and 3 it was assumed to be linear as in Equation 2.10, which is valid for currents below 5 Amperes in the experimental VRM. However, a VRM is normally operated with its magnetic material in saturation in order to minimize the power rating of its inverter [21]. In order to adequately model the effects of saturation and the dependence on the rotor position, the current-versus-flux relationship for the VRM is approximated by a polynomial in $\lambda$ whose coefficients are functions of the rotor angle $\theta$. The current verses flux relationship for the nth phase is given by

$$i_n = \sum_{k=0}^{N_1-1} \alpha_k(\theta_n)\,\lambda_n^{N_1-k} \tag{5.8}$$

where

$$\alpha_k(\theta_n) = \sum_{l=0}^{N_2} \beta_{kl}\,\cos[N_r\theta_n(N_2-l)] \tag{5.9}$$

The expansion coefficients $\beta_{kl}$ are determined in the following manner. First, an $N_1$th order polynomial of the form shown in Equation 5.8 is fit to each of the 16 curves shown in Figure 2.3 in a least-squares sense. This yields the value of $\alpha_k$ for $k = 0, 1, \ldots, N_1$ at the 16 discreet positions $\theta = 0^o, 2^o, \ldots, 30^o$ which can be put into matrix form as follows

$$\text{16 rows} \left\{ \underbrace{\begin{bmatrix} \alpha_0(0^o) & \alpha_1(0^o) & \cdots & \alpha_{N_1-1}(0^o) \\ \alpha_0(2^o) & \alpha_1(2^o) & \cdots & \alpha_{N_1-1}(2^o) \\ & \vdots & & \\ \alpha_0(30^o) & \alpha_1(30^o) & \cdots & \alpha_{N_1-1}(30^o) \end{bmatrix}}_{N_1 \text{ columns}} \right.$$

Next, an $N_2$th order cosine series of the form shown in Equation 5.9 is fit to the 16 points in column one again in a least-squares sense yielding the $N_2 + 1$ coefficients $\beta_{0l}$. This step is repeated for the other $N_1 - 1$ columns of the matrix yielding $\beta_{1l}, \beta_{2l}, \ldots, \beta_{N_1 l}$. The values of $\beta_{kl}$ for the experimental VRM determined using $N_1 = 5$ and $N_2 = 10$ are given in Table 5.1. The MATLAB M-files used to determine these coefficients are given in Appendix 5.

Figures 5.1–5.3 show plots of the flux verses current curves of the experimental VRM and the fitted curves at rotor angles of 0, 14, and 30 degrees. These plots show that there is good agreement between the actual and fitted curves at all current levels including those in saturation. As expected, the largest errors occur at the aligned position where the current-flux curve bends the most. All of the curves, however, have slight errors around 16 Amperes. This is due to the fact that polynomial fits are valid only over the domain of points used in the actual fitting. When determining the coefficients for the experimental VRM, the maximum value of the current was approximately 15.9 Amperes. At the origin, this problem is avoided by not having any constant term in the polynomial of Equation 5.8.

| $l$ | $\beta_{0l}$ | $\beta_{1l}$ | $\beta_{2l}$ | $\beta_{3l}$ | $\beta_{4l}$ |
|---|---|---|---|---|---|
| 0 | $1.3205 \times 10^5$ | $6.1552 \times 10^3$ | $-6.0646 \times 10^2$ | $7.6789 \times 10^{-1}$ | $1.2506$ |
| 1 | $-5.5807 \times 10^5$ | $-1.3583 \times 10^4$ | $4.3489 \times 10^3$ | $-8.8548 \times 10^1$ | $-2.4099$ |
| 2 | $1.1096 \times 10^6$ | $-3.3699 \times 10^4$ | $-5.8892 \times 10^3$ | $2.2368 \times 10^2$ | $4.8191 \times 10^{-1}$ |
| 3 | $-1.1908 \times 10^6$ | $1.8456 \times 10^5$ | $-7.1263 \times 10^3$ | $4.4564 \times 10^1$ | $9.1909 \times 10^{-1}$ |
| 4 | $-9.8087 \times 10^5$ | $-9.6229 \times 10^4$ | $1.8146 \times 10^4$ | $-5.5584 \times 10^2$ | $1.0212 \times 10^{-1}$ |
| 5 | $5.3986 \times 10^6$ | $-6.5613 \times 10^5$ | $2.1212 \times 10^4$ | $-2.0606 \times 10^2$ | $6.9344$ |
| 6 | $-3.1644 \times 10^6$ | $6.4900 \times 10^5$ | $-4.0461 \times 10^4$ | $9.1372 \times 10^2$ | $-1.1264 \times 10^1$ |
| 7 | $-6.9785 \times 10^6$ | $8.8913 \times 10^5$ | $-3.1092 \times 10^4$ | $4.0449 \times 10^2$ | $-1.5473 \times 10^1$ |
| 8 | $7.6580 \times 10^6$ | $-1.2448 \times 10^6$ | $5.9599 \times 10^4$ | $-1.0849 \times 10^3$ | $6.6049 \times 10^1$ |
| 9 | $4.9605 \times 10^6$ | $-7.0111 \times 10^5$ | $2.9456 \times 10^4$ | $-4.6172 \times 10^2$ | $-1.7971 \times 10^2$ |
| 10 | $-2.8802 \times 10^6$ | $4.6541 \times 10^5$ | $-1.9495 \times 10^4$ | $3.2080 \times 10^2$ | $2.0577 \times 10^2$ |

Table 5.1: Expansion coefficients of the flux-current relationship for the experimental VRM.

This forces the current to be zero when the flux is zero.

### 5.1.2 Determination of the Torque

The total torque $\tau_m$ can be written in terms of the flux variables $\lambda_n$ and $\theta$ using the current-flux relationship determined in the previous section as follows. Substituting Equations 5.8 and 5.9 into Equation 5.4 and simplifying the result gives

$$
\begin{aligned}
W_n(\theta_n, \lambda_n) &= \int_0^{\lambda_n} \sum_{k=0}^{N_1-1} \alpha_k(\theta_n) \lambda_n'^{N_1-k} d\lambda_n' \\
&= \sum_{k=0}^{N_1-1} \alpha_k(\theta_n) \int_0^{\lambda_n} \lambda_n'^{N_1-k} d\lambda_n'
\end{aligned}
$$

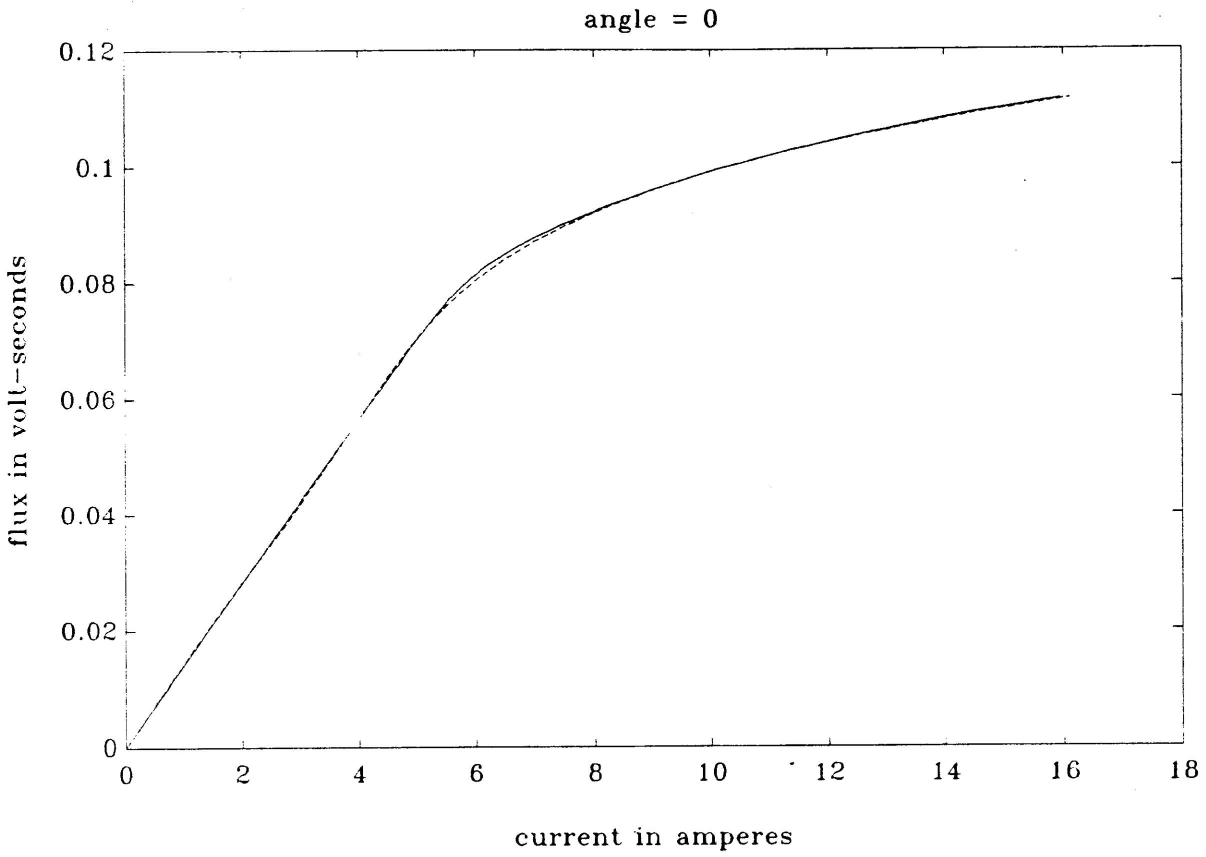

Figure 5.1: Comparison between the flux vs. current curve of the experimental VRM and the fitted curve at $\theta = 0°$. The dashed line is the fitted curve.

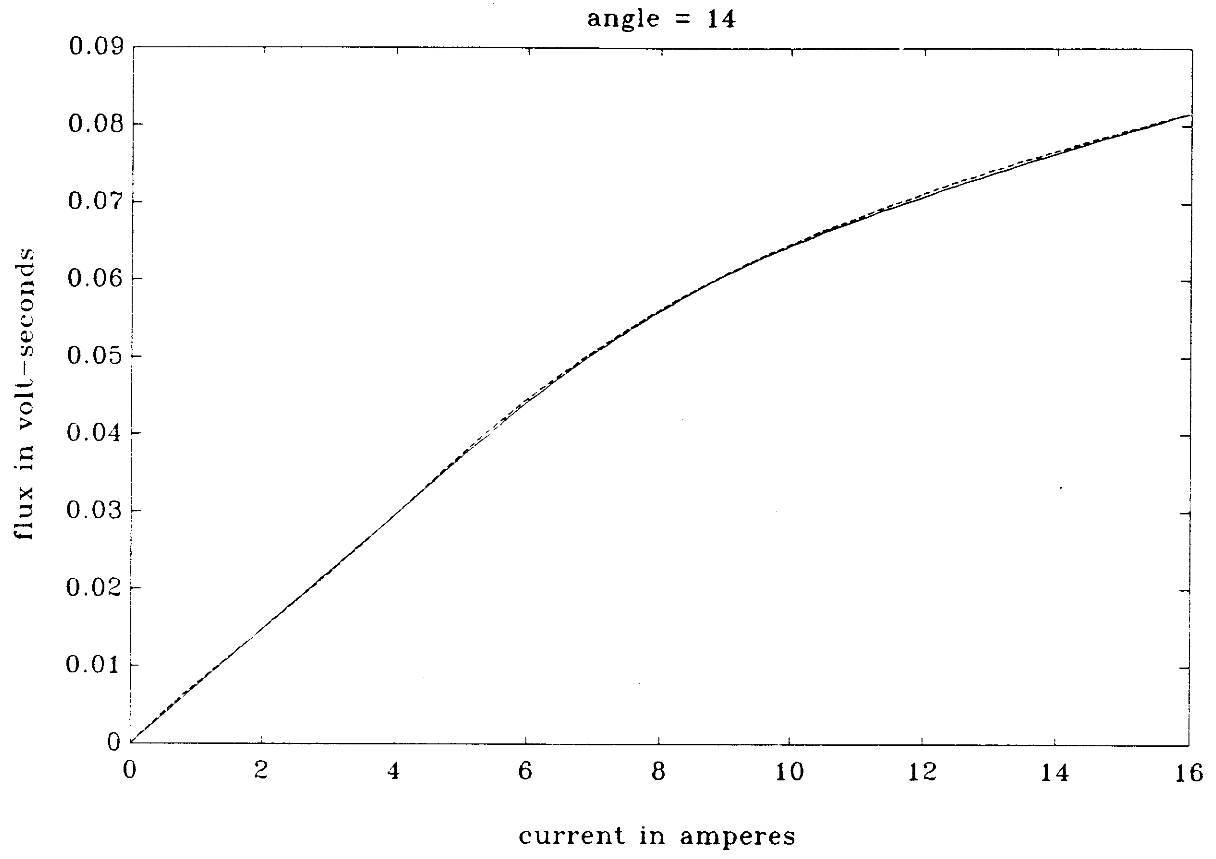

Figure 5.2: Comparison between the flux vs. current curve of the experimental VRM and the fitted curve at $\theta = 14^o$. The dashed line is the fitted curve.

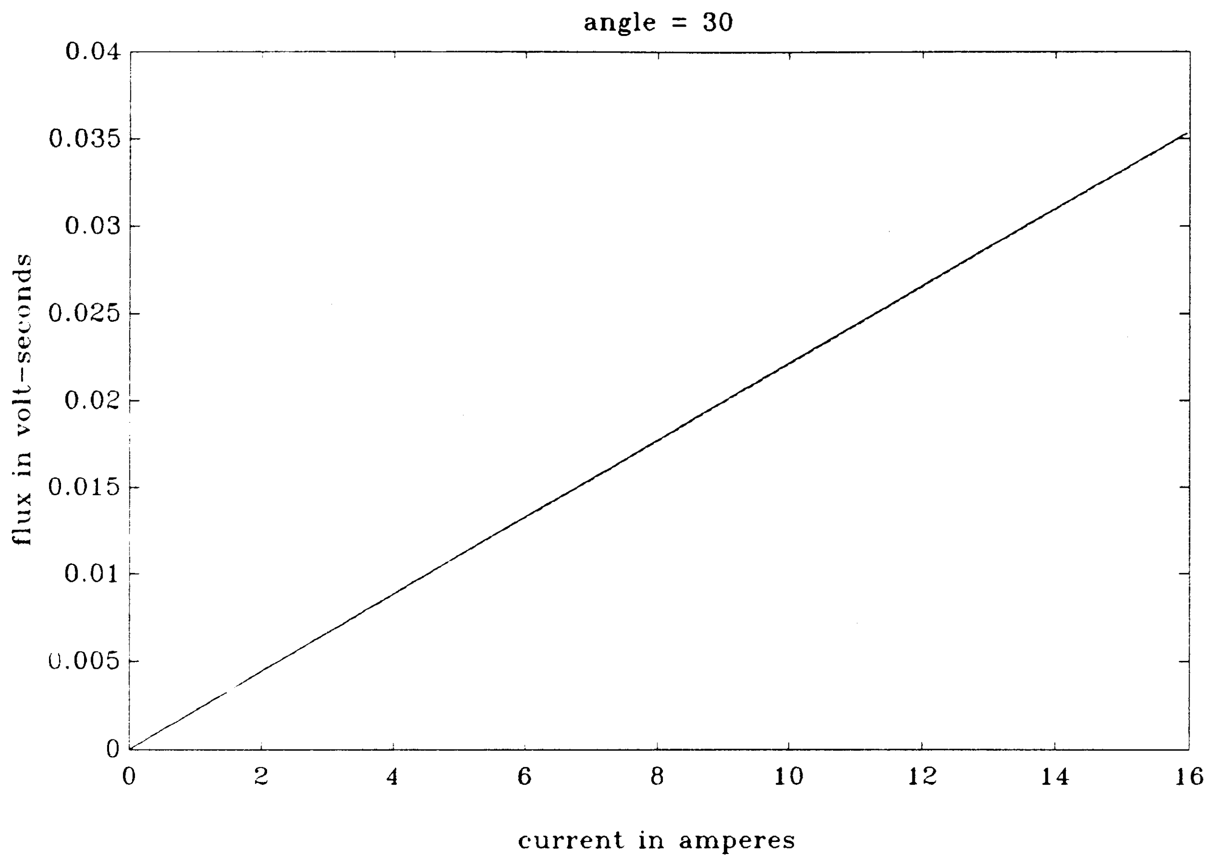

Figure 5.3: Comparison between the flux vs. current curve of the experimental VRM and the fitted curve at $\theta = 30^o$. The dashed line is the fitted curve.

$$= \sum_{k=0}^{N_1-1} \alpha_k(\theta_n) \frac{\lambda_n^{N_1-k+1}}{N_1-k+1} \tag{5.10}$$

Combining Equations 5.10 and 5.3 yields

$$\tau_m = \sum_{n=1}^{4} -\frac{\partial}{\partial \theta_n} \sum_{k=0}^{N_1-1} \alpha_k(\theta_n) \frac{\lambda_n^{N_1-k+1}}{N_1-k+1} \Bigg|_{\lambda_n}$$

$$= -\sum_{n=1}^{4} \sum_{k=0}^{N_1-1} \frac{\lambda_n^{N_1-k+1}}{N_1-k+1} \frac{\partial \alpha_k(\theta_n)}{\partial \theta_n} \tag{5.11}$$

where

$$\frac{\partial \alpha_k(\theta_n)}{\partial \theta_n} = -N_r \sum_{l=0}^{N_2} (N_2-l)\beta_{kl} \sin[N_r\theta_n(N_2-l)] \tag{5.12}$$

### 5.1.3 Determination of the Radial Forces

In order to determine the radial forces using Equation 5.6, the air gap length $g$, the current $i_n$, and the air gap $A_{gap}(\theta_n)$ need to be known at all times. In the simulations, the air gap length is approximated as being constant since the experiments of Chapter 3 showed that the maximum radial deflection of the stator was on the order of a micron, which is much less than the nomial length of 0.0254 cm. The current $i_n$ is available at each step in the simulation since it is directly related to the state variable $\lambda_n$ through Equation 5.8. The air gap area $A_{gap}(\theta_n)$ is determined from known quantities as follows. The air gap area is related to the flux through the following equation [6]

$$\lambda = \mu_o N H_r A_{gap,n} \tag{5.13}$$

where $H_r$ is given by Equation 3.5. Combining these two equations and solving for the air gap area yields

$$A_{gap,n} = \frac{2g\lambda_n}{\mu_o N^2 i_n} \tag{5.14}$$

The only independent variable in this equation is the flux $\lambda_n$. The $\theta$-dependence of $A_{gap,n}$ is contained in the current $i_n$ through Equation 5.8. By fixing $\lambda_n$ and varying $\theta$, a picture of $A_{gap}(\theta)$ can be obtained. Figure 5.4 shows a plot of $A_{gap,n}$ verses $\theta$ for one phase at the flux level $\lambda_n = 0.02$ Wb. Substituting Equation 5.14 into Equation 5.6 yields the following equation for the radial force

$$F_{r,n} = -\frac{\lambda_n i_n}{4g} \tag{5.15}$$

### 5.1.4 Modeling of the Controller and Inverter

The role of the controller is to generate the low-level signals that tell the inverter when to turn phases on and off and at what level to chop the current. It uses the rotor position $\theta$ obtained from a shaft-encoder ($\theta$ can also be determined from the terminal characteristics of the VRM [11,17]) along with the turn-on angle $\theta_{on}$ and turn-off angle $\theta_{off}$ to generate

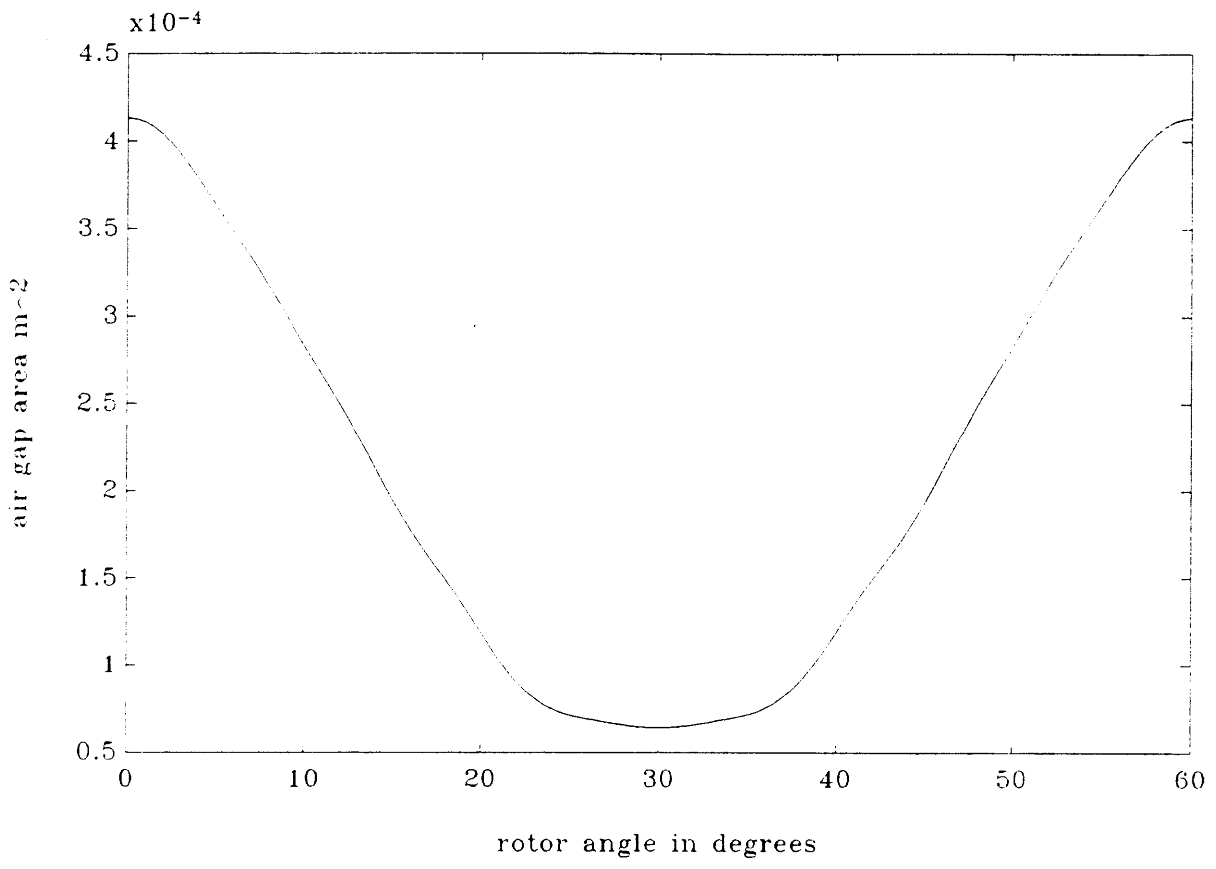

Figure 5.4: Plot of the air gap area $A_{gap}(\theta)$ at the flux level $\lambda = 0.02$ Wb.

the phase-on/off signal $\Gamma_n$ based on the following law. If $\theta_{off} \geq \theta_{on}$ then

$$\Gamma_n(\theta_n) = \begin{cases} \text{on} & \text{if } \theta_{on} \leq \theta_n \leq \theta_{off} \\ \text{off} & \text{otherwise} \end{cases} \tag{5.16}$$

or if $\theta_{off} < \theta_{on}$ then

$$\Gamma_n(\theta_n) = \begin{cases} \text{on} & \text{if } \theta_{on} \leq \theta_n \leq \frac{2\pi}{N_r} \text{ or } 0 \leq \theta_n \leq \theta_{off} \\ \text{off} & \text{otherwise} \end{cases} \tag{5.17}$$

All angles are assumed to be modulo $2\pi/N_r$. The current chopping level $i_{chop}$, $\theta_{on}$, and $\theta_{off}$ are usually set by the controller according to some closed-loop speed and/or torque algorithm. In the simulations that follow, however, these parameters are set at the start and remain constant throughout the duration of each simulation.

The inverter (see Figure 2.5) uses the phase-on/off signal $\Gamma_n$ and $i_{chop}$ to generate the gate signals for the FET's. Depending on the levels of these signals the inverter resides in one of the following four states:

| | |
|---|---|
| state 1: | both FET's are on and both diodes are off; |
| state 2: | upper FET and diode are off and lower FET and diode are on; |
| state 3: | both FET's are off and both diodes are on; |
| state 4: | both FET's and both diodes are off. |

The voltage $v_n$ applied to the VRM phases during each of these states is given by:

$$v_n = \begin{cases} +V_{supply} & \text{state 1} \\ -V_{diode} & \text{state 2} \\ -(V_{supply} + 2V_{diode}) & \text{state 3} \\ 0 & \text{state 4} \end{cases} \tag{5.18}$$

where the forward voltage drop across a diode $V_{diode} = 0.7$ Volts. The total circuit resistance for the nth phase is

$$R_n = \begin{cases} R_{phase} + 2R_{FET} & \text{state 1} \\ R_{phase} + R_{FET} & \text{state 2} \\ R_{phase} & \text{state 3} \\ R_{phase} & \text{state 4} \end{cases} \tag{5.19}$$

where the resistance of a phase is $R_{phase} = 0.8\Omega$, and the on resistance of a FET is $R_{FET} = 0.18\Omega$.

Current regulation during the conduction interval, defined by $\theta_{on}$ and $\theta_{off}$ is performed by the inverter using the following chopping algorithm. Like the algorithm described in Chapter 2, the upper FET is always switched off whenever the phase current rises above $i_{chop}$. The decision whether or not to turn on the upper FET is made every $t_{chop}$ seconds. If the phase current is below $i_{chop}$ at a decision point then the upper FET is turned on, otherwise, it remains off until the next decision point. This algorithm restricts the values that the chopping frequency can take on to $f_{chop} = 1/t_{chop}$ and its submultiples. As a result, $t_{chop}$ can be chosen to maximize the distance between the discrete values that the chopping frequency can take on and the resonant frequencies of the stator. In the simulations, $f_{chop}$ is set to 10 kHz, the same value used by the actual inverter.

## 5.2 Evaluation of the Constant-Speed VRM Model

The constant-speed VRM model is numerically simulated using a fixed step, fourth-order Runge-Kutta integration routine, which is listed in Appendix 5. A discussion of this method is given in [9]. Due to the symmetry of the phases and the periodic nature of the flux-current relationship, the simulations need to be carried out for only one phase of the VRM over a period of an electrical cycle. The total torque is determined by adding shifted versions of the torque due to a single phase to itself.

The main driver program is the MATLAB M-file MOTOR.M which prompts the user for the operating point parameters. It passes these parameters to the C routine MOTOR.EXE which is the actual simulation program. The output of MOTOR.EXE includes the current, torque, and radial force waveforms as well as the computed average torque. Both of these programs are also listed in Appendix 5.

In order to evaluate the performance of the constant-speed VRM model, simulations are performed at two new operating points (operating points 3 and 4) as well as at the two described in Chapter 3, and the results are compared with data taken from the experimental VRM running at the same operating points. Operating points 3 and 4 are chosen so that comparisons of the currents and torques for the model and the experimental VRM can be made at both low and high speeds. In order to check the validity of the radial force, operating points 1 and 2 are used for reasons explained later. The parameters defining all four operating points are given in Table 5.2. Figures 5.5 and 5.6 show comparisons

| operating point | $\theta_{on}$ (degrees) | $\theta_{off}$ (degrees) | $V_{supply}$ (volts) | speed (rpm) | $i_{chop}$ (amps) | load torque ( $\times 10^{-3}$ Nm) |
|---|---|---|---|---|---|---|
| 1 | 34 | 49 | 40.0 | 3720 | – | 106 |
| 2 | 30 | 45 | 30.0 | 3720 | – | 106 |
| 3 | 34 | 54 | 20.0 | 951 | 1.7 | 55.0 |
| 4 | 25 | 45 | 31.2 | 6538 | – | 84.2 |

Table 5.2: Definition of the four operating points used to evaluate the performance of the constant-speed VRM model.

between the actual and simulated phase current waveforms at the third and fourth operating points, respectively. The differences of the current waveforms at the third operating point are mainly due to hysteresis in the current regulating circuitry of the inverter which isn't included in the VRM model. The combined effects of the hysteresis and noise results in the switching period multiplication as shown in Figure 5.5. At the fourth operating point, the differences can be attributed to the finite resolution of the rotor position that is available to the controller and the finite processing capabilities of the controller. As a result of these two limitations, $\theta_{on}$ and $\theta_{off}$ can vary from their commanded values by $\pm 3$ mechanical degrees, resulting in sizable changes in the current waveform. This is illustrated in Figure 5.7 which shows the current waveform of one phase of the experimental VRM at the fourth operating point. The simulated torque waveforms for one phase of the VRM at the third and fourth operating points are shown in Figures 5.8 and 5.9, respectively. The average torques corresponding to these two simulated waveforms are $54.6 \times 10^{-3}$ Nm and $81.8 \times 10^{-3}$ Nm, respectively.

The simulated radial force waveforms at the third and fourth operating points are shown in Figures 5.10 and 5.11, respectively. Since the radial force of the experimental VRM can-

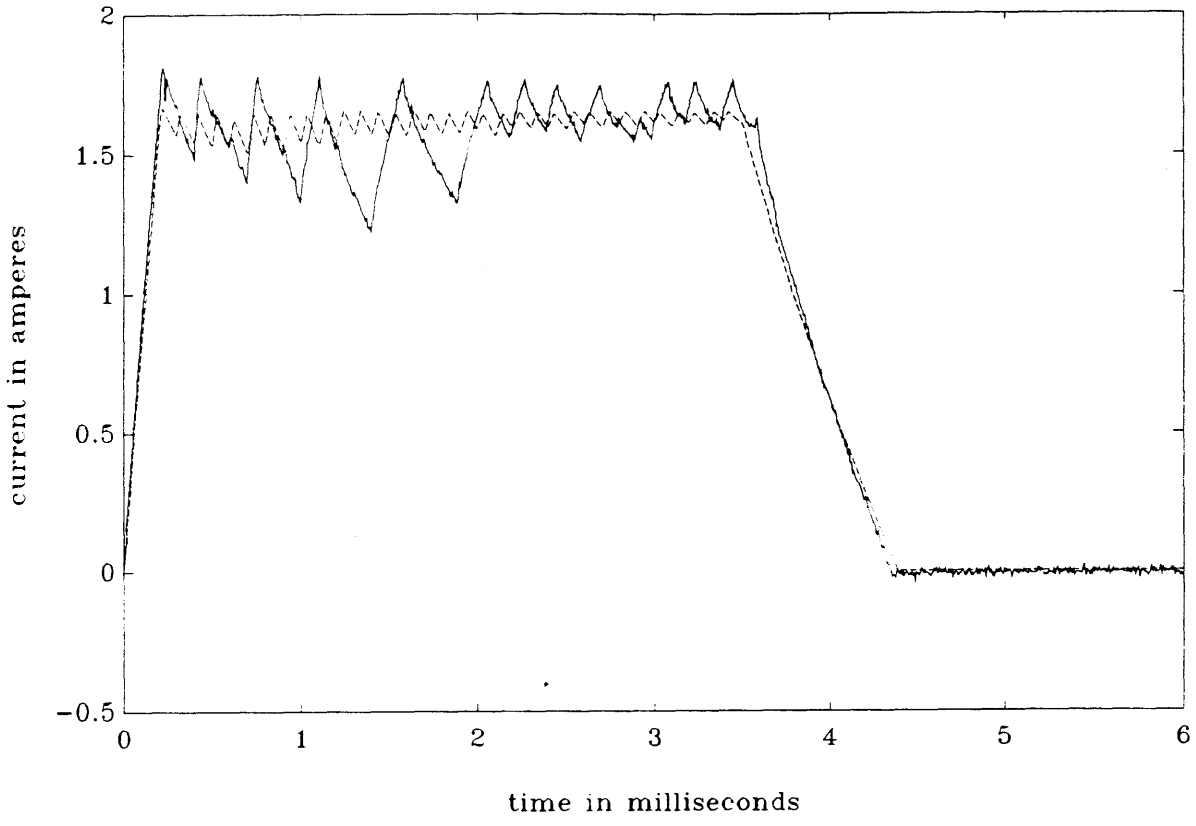

Figure 5.5: Comparison between the actual and simulated current waveform at the third operating point defined in the text. The solid line is the actual current waveform.

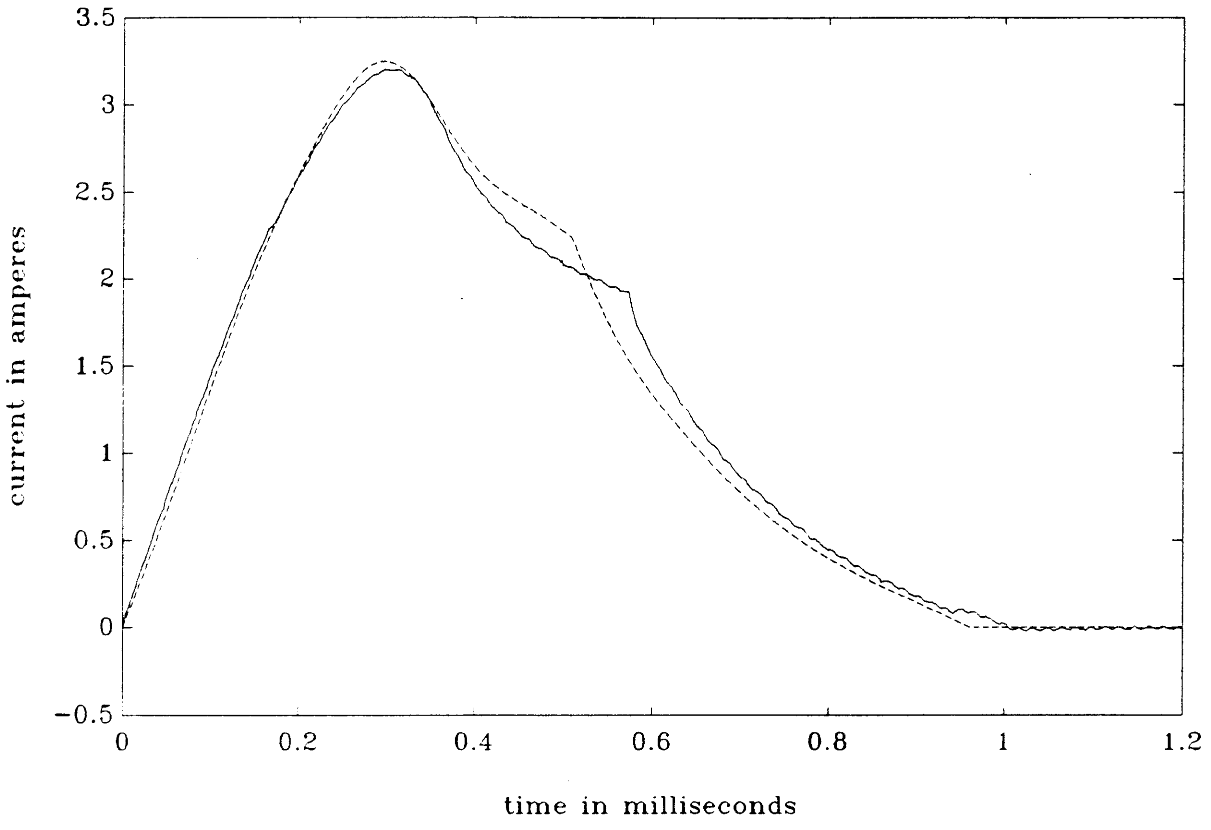

Figure 5.6: Comparison between the actual and simulated current waveform at the fourth operating point defined in the text. The solid line is the actual current waveform.

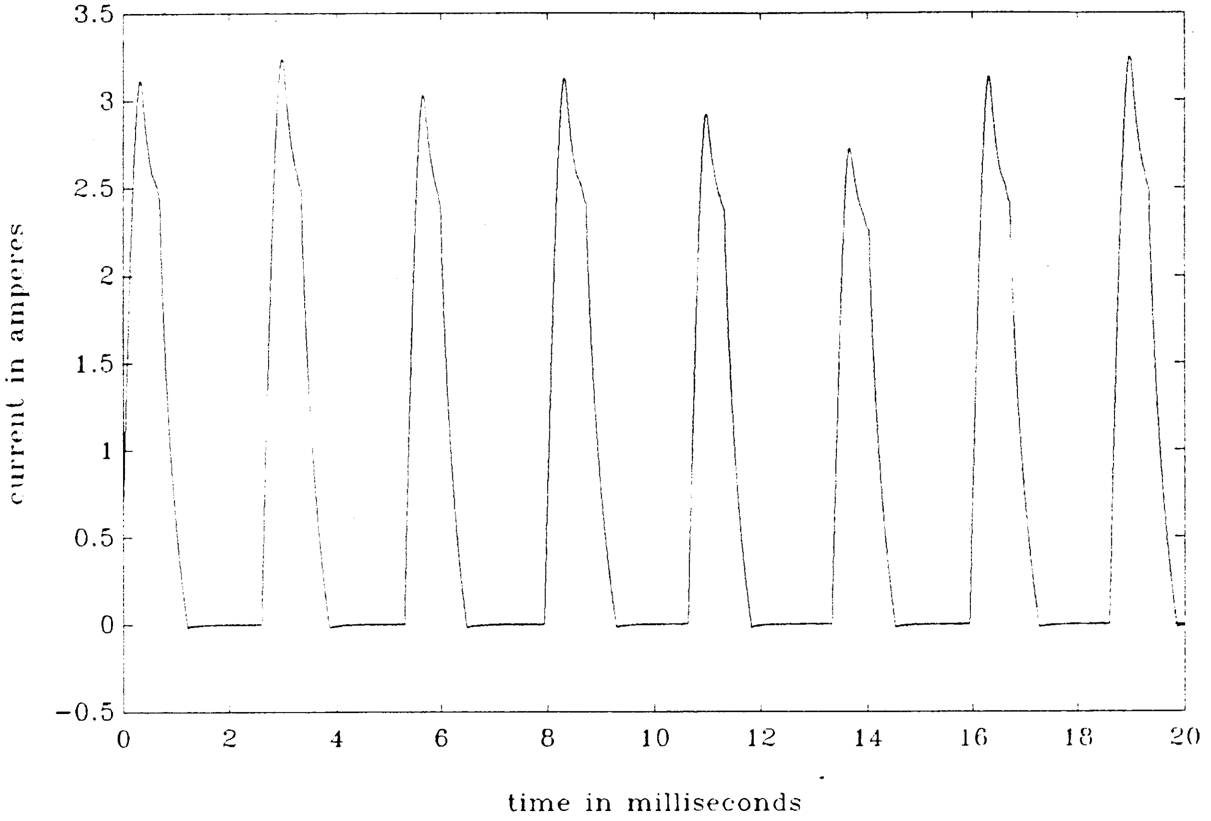

Figure 5.7: Current waveform of one phase of the experimental VRM showing the variation of the shape of the current pulses.

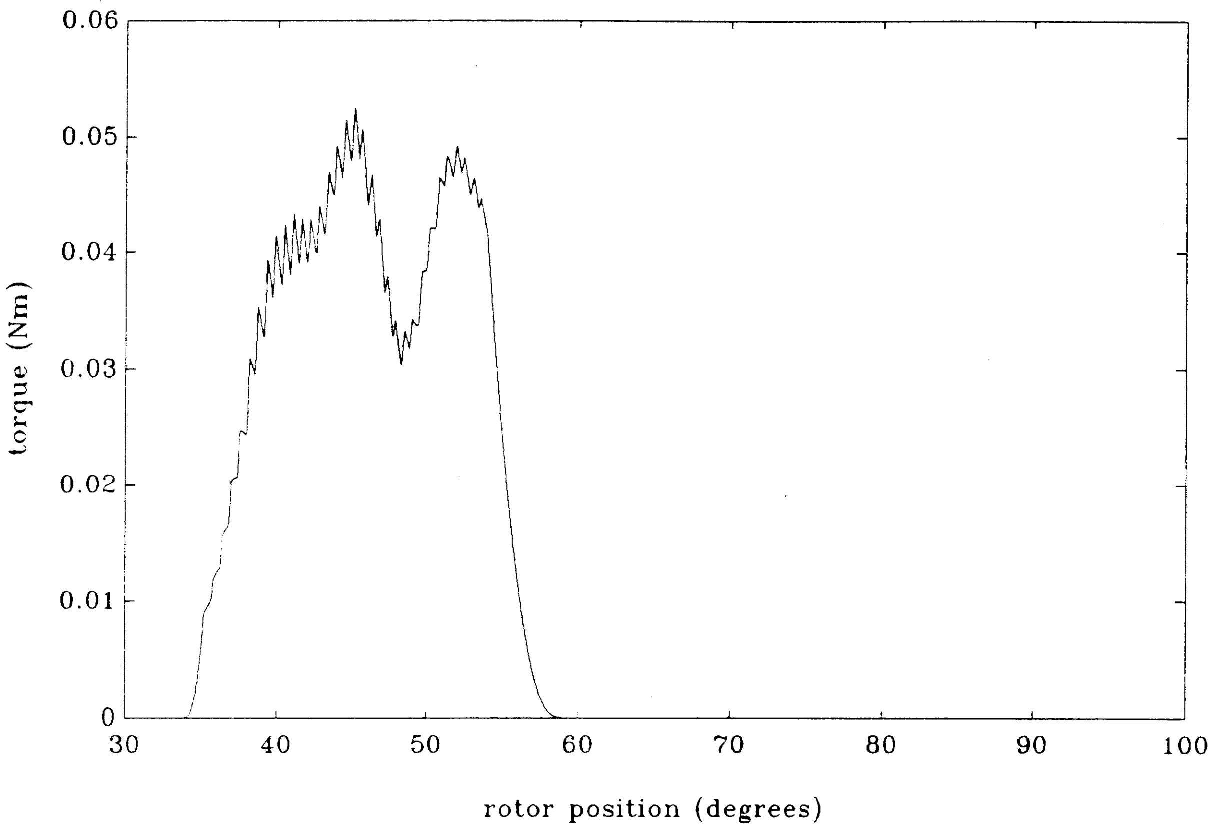

Figure 5.8: Simulated torque waveform at the third operating point.

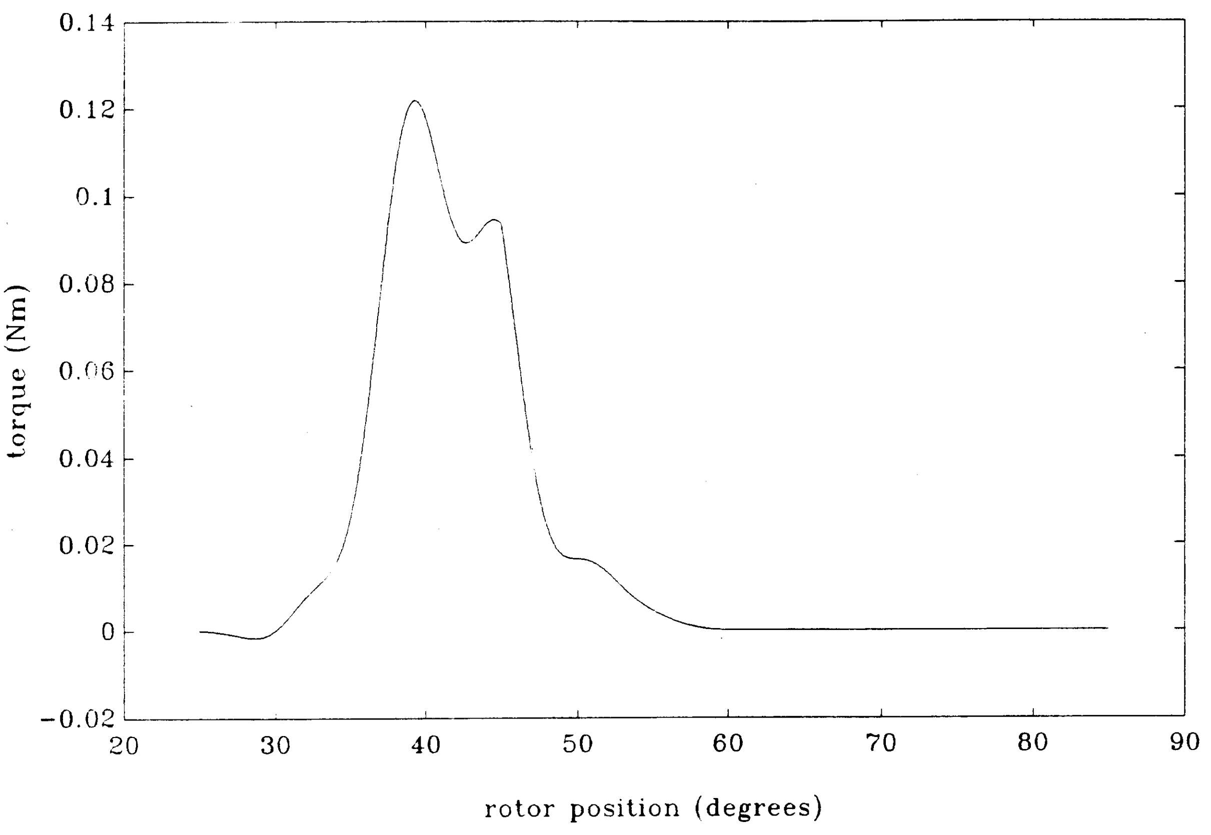

Figure 5.9: Simulated torque waveform at the fourth operating point.

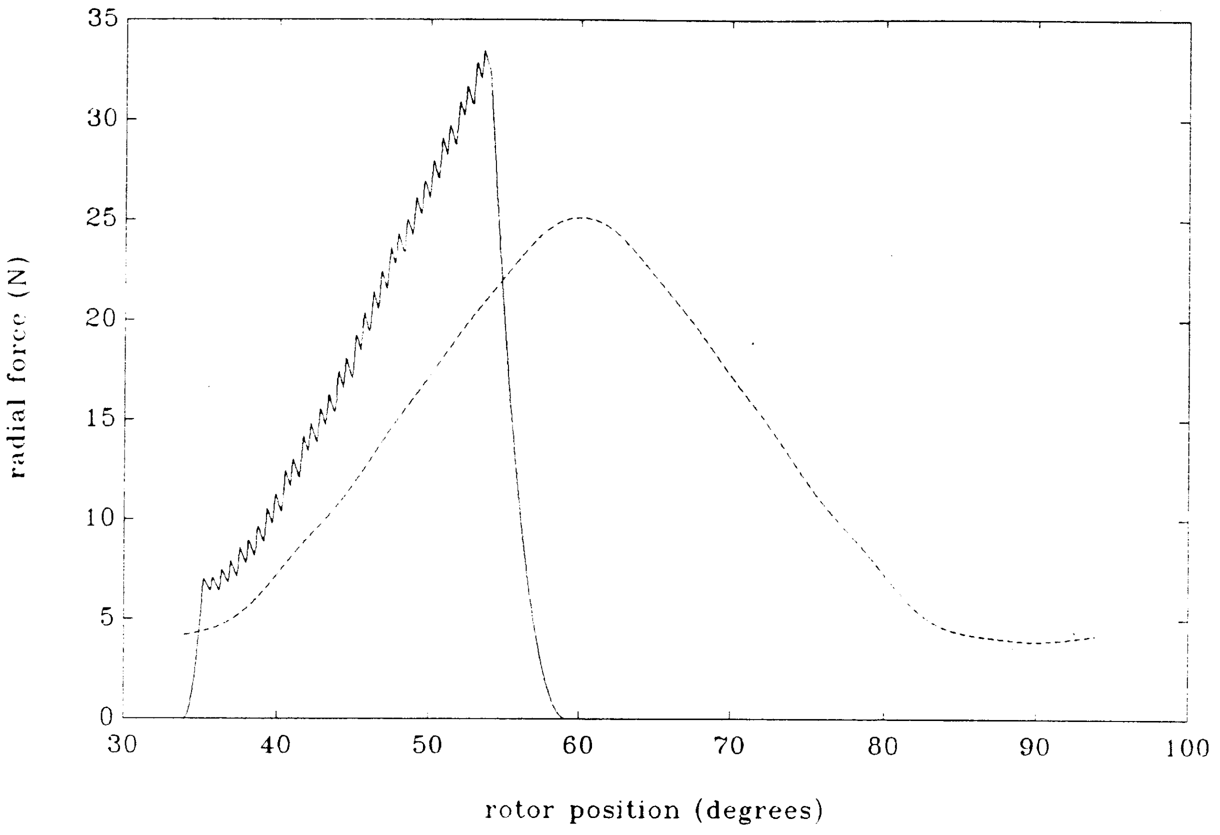

Figure 5.10: Simulated radial force waveform at the third operating point. The dashed line is the air gap area which has been scaled up for clarity.

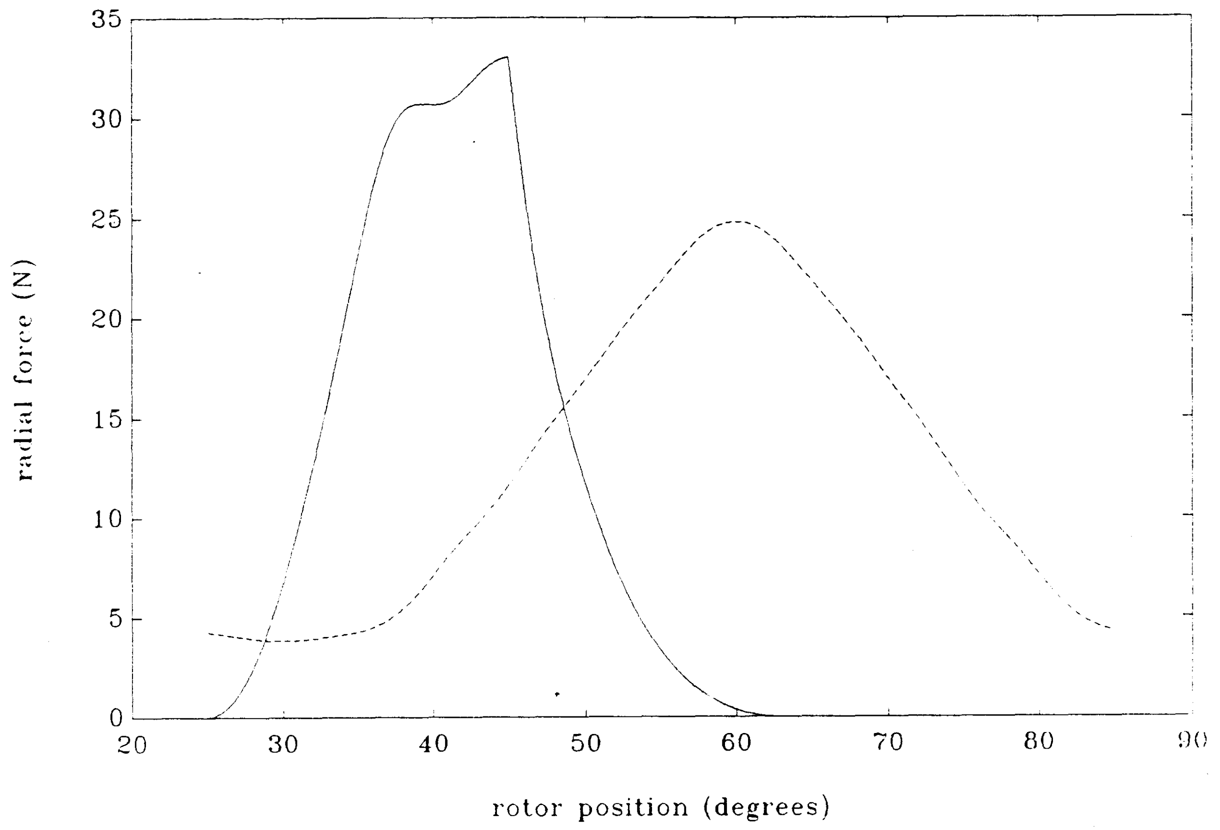

Figure 5.11: Simulated radial force waveform at the fourth operating point. The dashed line is the air gap area which has been scaled up for clarity.

not be measured, the accuracy of the simulated radial force waveforms cannot be directly checked. However, from the experiments of Chapter 3, we know that at a given frequency, the stator acceleration measured above the excited phase is proportional to the radial force component at that frequency. Therefore, the accuracy of the simulated radial force waveforms is checked here by comparing the difference of a specific harmonic component of two radial force waveforms to the difference of the same harmonic component of the resulting acceleration waveform. Since this comparison is most easily done when there is significant acceleration of the stator, operating points 1 and 2 are used. Recall that at both of these operating points, the 7th harmonic of the radial force excites the resonant frequency at 2604 Hz.

Plots of the simulated current and radial force waveforms for operating points 1 and 2 are shown in Figures 5.12 and 5.13, respectively. The acceleration waveforms for the two operating points, whose spectra are shown in Figures 3.8 and 3.12, respectively, are taken from an accelerometer above Phase A of the motor. The comparison is made only for the seventh harmonic component of the waveforms at the resonant frequency of 2604 Hz, since it is this harmonic which produces the acoustic noise. Recall from Chapter 3 that the second operating point was chosen so that the acoustic noise was reduced while the speed remained the same. The seventh harmonic components of the simulated and experimental waveforms is shown in Table 5.3. As noted in Chapter 3, the magnitude of the seventh harmonic of $i^2$ at the second operating point is larger than that at the first in the experimental case. Table 5.3 shows that this is also true for the simulated waveforms, although the difference is slightly larger. A comparison between the acceleration harmonics and simulated radial force harmonics, however, shows that while the acceleration magnitude decreases by 7.7

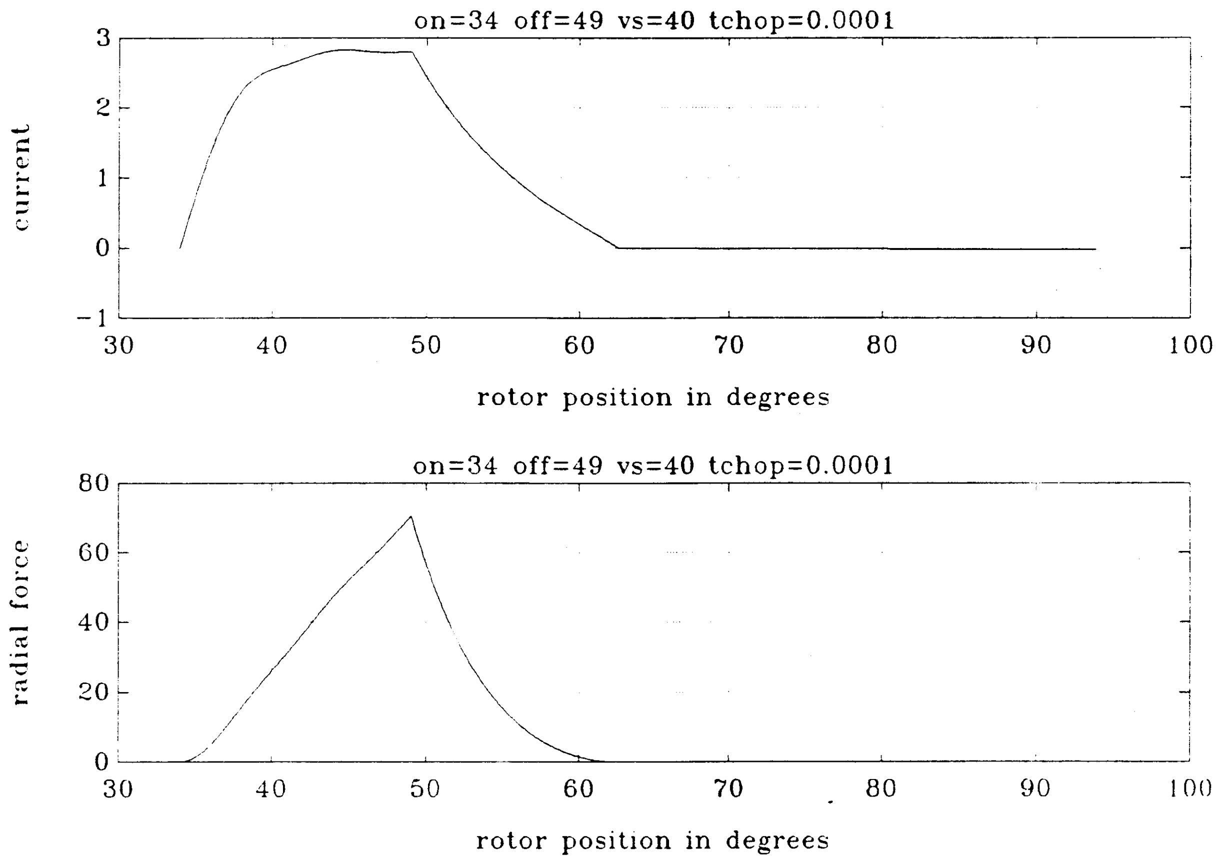

Figure 5.12: Simulated current and radial force waveforms for the first operating point.

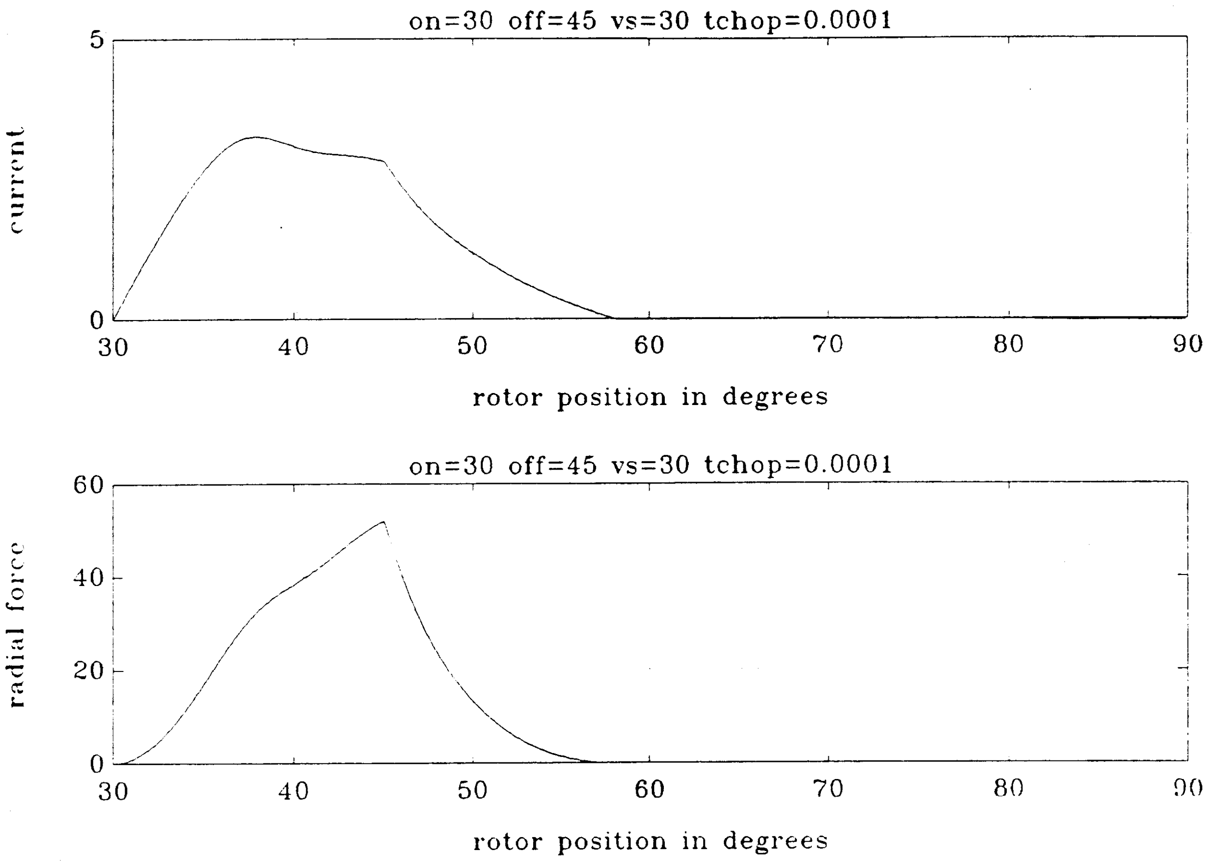

Figure 5.13: Simulated current and radial force waveforms for the second operating point.

| operating | experimental | | simulated | |
|---|---|---|---|---|
| point | $i^2$ | acceleration | $i^2$ | radial force |
| 1 | 37.5 | 55.5 | 35.4 | 51.2 |
| 2 | 39.5 | 47.8 | 39.1 | 50.2 |

Table 5.3: Magnitudes of the 7th harmonic components of the simulated and experimental waveforms.

dB, the radial force magnitude decreases only by 1.0 dB. It is difficult to specify the exact source of this error since there are many possibilities. For example, in deriving the equation for the radial force (Equation 5.15), the assumption is made that the flux in the air gap is radially directed and uniform across the cross-sectional overlap area of the rotor and stator poles. Additional sources could include errors in measuring the flux-current curves of the experimental VRM and fitting polynomials to these measured curves, errors in measuring the acceleration of the stator, numerical round off errors, etc.

Compounding the difficulty presented by all of these possible error sources is the fact that harmonic components of the radial force waveform as high as the seventh have to be accurately predicted; the magnitudes of the seventh harmonic component of the radial force waveforms shown in Figures 5.12 and 5.13 are over 20 dB smaller than the fundamental. It seems clear, therefore, that if the acoustic noise problem of the VRM is to be solved using current (radial force) waveform shaping, further research is needed to determine the source(s) of the error and the model needed to account for all of the factor which influence the shape of the radial force waveform.

# Chapter 6

# Summary and Conclusions

## 6.1 Summary

This thesis investigated the origin of acoustic noise in VRMs and the issues involved in predicting and reducing it. In Chapter 2, a brief discussion was given of the experimental VRM used throughout this thesis. In particular, the construction of the experimental VRM was detailed and a discussion of the basic operating principles was given.

In Chapter 3, an experimental procedure for determining and characterizing the dominant noise source in the experimental VRM was presented. The experiments were designed to investigate several potential noise sources of magnetic and mechanical origin in order to determine their relative contribution to the total acoustic noise. From these experiments, the experimental VRM was found to have three resonant frequencies at 2604 Hz, 9200 Hz, and 14.2 kHz corresponding to circumferential modes zero, two, and four, respectively. At each of these frequencies, several parts of the motor were found to be vibrating. The vibrations that produced the most noise, however, were radial vibrations of the stator induced by radial magnetic forces in the motor. The magnitude of the vibrations were largest when harmonics of the radial forces having sufficiently large magnitudes (usually the harmonics up to and including the seventh) coincided with resonant frequencies of the stator. The experiments also revealed that at the resonant frequencies, the acoustic noise was proportional to the magnitude of the vibrations. Under normal operating conditions where the speed could range up to 8000 rpm, only the resonant frequency at 2604 Hz was excited, since the radial force harmonics near the other two resonant frequencies were too small to produce any measurable response.

An additional finding of the experiments was that at the speeds where radial force harmonics coincided with the resonant frequency at 2604 Hz, the acceleration and acoustic noise could be reduced through the correct choice of the four control parameters $\theta_{on}$, $\theta_{off}$, $i_{chop}$, and $V_{supply}$. This point was demonstrated for the experimental VRM operating at a speed of 3720 rpm where the 7th harmonic of the radial force waveform excited the resonant frequency at 2604 Hz. A 6 dB reduction of the acoustic noise level was achieved at this speed through trial and error. In order to determine the optimal values of the control parameters, it was concluded that a model was needed that could accurately predict the radial force waveform based on the control parameters and the topology of the inverter.

While radial vibrations of the stator produced most of the noise in the experiments performed in this thesis, this result is by no means universally applicable to all VRMs in all applications. The other vibrations that were observed at each resonant frequency (such as axial vibrations of the endbells and rotor shaft) could also produce acoustic noise depending on how the motor is mounted and loaded. Since the experimental VRM was elastically mounted and attached to the load motor using a rubber coupling, these vibrations did not produce significant noise. In other configurations, however, these vibrations could easily excite mechanical resonances of the structure used to support the VRM or the load attached to it. It must be emphasized, therefore, that in the context of acoustic noise reduction, the VRM cannot be treated separately from the structure in which it is to be used.

In Chapter 4, an analytical method for the determining the resonant frequencies and modes of the stator was presented. The analysis used the Rayleigh-Ritz method, which is based on energy conservation principles, to determine the frequency equation of the stator. Due to the difficulty in finding basis functions which satisfied all of the boundary, compatibility, and equilibrium conditions and the difficulty in modeling losses, the analysis was restricted to the stator cylinder and the assumptions were made that there were no surface forces acting on this cylinder and that there were no losses. As a result of the latter assumption, it was impossible to determine any amplitude information concerning the vibrations.

While this analysis was able to correctly predict mode shapes, the values it predicted for the resonant frequencies were not accurate enough to be useful in predicting the acoustic noise. The reason for this is that the resonant frequencies of the experimental VRM had very low equivalent damping factors so that if the predicted frequencies were off by a hundred Hertz or more, significant errors in predicted vibration amplitudes would result. Increasing the size of the model used in the analysis did not improve accuracy, but instead led to round-off errors. Therefore, the main usefulness of the analysis was firstly its ability to give a yes/no answer to the questions of whether the resonant frequencies were in the audible range and whether they were low enough to be excited by radial force harmonics, and secondly, its ability to show how the resonant frequencies changed as the geometry of the stator changed so that they could be raised high enough to prevent harmonic excitation of the resonant frequencies of the stator. When applied to the experimental VRM, the analysis showed that the most effective way of increasing the resonant frequencies was to increase the thickness of the stator cylinder. However, due to the size of the increase needed to prevent harmonic excitation of the lowest resonant frequency, this method of reducing the acoustic noise was deemed to be too costly.

Finally, in Chapter 5, a model for predicting the magnetic forces in the VRM was described and tested. The current-flux relationship of the experimental VRM was approximated by a polynomial in $\lambda$ whose coefficients were functions of the rotor position $\theta$. The polynomial was fit to the flux-current curves measured for the experimental VRM. This polynomial fit allowed the operation of the motor to be simulated even at flux levels were the magnetic material was saturated. Using this polynomial, an analytic expression for the torque was derived enabling the total and average torque to be easily determined. Comparisons between the actual and simulated currents at two different operating points showed that the model performed well. The average torque predicted by the model was also within the range of expected values at both operating points (the means for accurate measurement of the actual torque were not available). The radial force waveform, however, could not be predicted accurately enough out to the seventh harmonic, which was the highest one that could excite the resonant frequency at 2604 Hz. Such accuracy would require that the current waveform be accurate at least out to the seventh harmonic which, as shown in Table 5.3, was not the case.

## 6.2 Suggestions for Future Research

As evident from Chapters 4 and 5, two issues that have to be addressed further concerning the prediction of acoustic noise of VRMs are the determination of the vibrational behavior of the stator and the radial magnetic force waveform. As far as the first issue is concerned, a vibrational model of the VRM is needed that addresses the stator poles and windings, frictional losses, and all of the boundary conditions. Due to the complexity that such a model would have, the best analytical method would probably be finite-element analysis [1].

As far as the second issue is concerned, one of the first things that needs to be done is to identify the sources which contribute to the errors in the predicted radial force waveform. As noted in Chapter 5, these sources could include errors in measuring the flux-current curves of the experimental VRM and fitting polynomials to them, assumptions used in deriving the radial force equation, and errors in measuring the acceleration of the stator. Once the sources are found, then it can be determined whether it would be feasible to try and compensate for them. However, since harmonics of the radial force as high as the seventh have to be accurately predicted, is seems unlikely that a simple solution will be found. As in the previous case, therefore, finite-element analysis would probably be the best method of determining the radial force.

Concerning the reduction of acoustic noise in VRMs, several techniques should be examined. First, damping shells could be added to the stator in order to reduce the quality factor of the resonant peaks. From transient experiments, the equivalent damping factors of the three resonant frequencies of the experimental VRM were found to be .01 or less. The shells, if fitted tightly enough around the stator, could not only increase the damping factor of the stator, but also could add to its stiffness thereby causing the resonant frequencies to increase.

Second, if an adequate means could be found for predicting the radial force given the values of the control parameters and inverter topology, then techniques for shaping the radial force waveform could be investigated. The goal, of course, would be to correctly choose the control parameters so that none of the large radial force harmonics coincided with resonant frequencies of the stator for all speeds of interest.

Third, the turn-on and turn-off angles could be randomly perturbed by $\pm 1$ or $\pm 2$ mechanical degrees so that coherent excitation of the resonant frequencies did not occur. When this technique was applied to the experimental VRM, an 8 dB decrease in the sound level was achieved while the motor was operating at 3720 rpm and at a load of 0.126 Nm. Research in this area could focus on determining the size of the optimum perturbation and the effect of the perturbations on other performance parameters of the VRM.

# Bibliography

[1] K. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[2] B. K. Bose, T. J. E. Miller, P. M. Szczesny, and W. H. Bicknell, "Microcomputer Control of Switched Reluctance Motor", IEEE Transactions on Industry Applications, vol. IA-22, no. 4, July/August 1986, pp. 708–715.

[3] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[4] M. Ehsani, J. T. Bass, T. J. E. Miller, and R. L. Steigerwald, "Development of a Unipolar Converter for Variable Reluctance Motor Drives", IEEE Transactions on Industry Applications, vol. IA-23, no. 3, May/June 1987, pp. 545–553.

[5] *Federal Register*, vol. 36, no. 96, Rule 50-204.10, May 1969, and July 1969.

[6] A. E. Fitzgerald, C. Kingsley, Jr., and S. D. Umans, *Electric Machinery*, 5th ed., McGraw-Hill Book Company, New York, 1990.

[7] W. Flügge, *Stresses in Shells*, 2nd ed., Springer-Verlag, New York, 1973.

[8] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines – EISPACK Guide Extension*, Lecture Notes in Computer Science, vol. 51, Springer-Verlag, 1977.

[9] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[10] R. S. Girgis and S. P. Verma, "Method for accurate determination of resonant frequencites and vibration behaviour of stators of electrical machines", IEE Proceedings, vol. 128, part. B., no. 1, January 1981, pp. 1–11.

[11] W. D. Harris, *Practical Indirect Position Sensing for a Variable Reluctance Motor*, M.S. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1987.

[12] J. R. Hutchinson, "Vibrations of Solid Cylinders", Journal of Applied Mechanics, vol. 47, December 1980, pp. 901–907.

[13] W. C. Hurty and M. F. Rubinstein, *Dynamics of Structures*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1964.

[14] M. Ilić-Spong, R. Marino, S. M. Peresada, and D. G. Taylor, "Feedback Linearizing Control of Switched Reluctance Motors", IEEE Transactions on Automatic Control, vol. AC-32, no. 5, May 1987, pp. 372–379.

[15] R. Krishnan, R. Arumugam, and J. F. Lindsay, "Design Procedure for Switched-Reluctance Motors," IEEE Transactions on Industry Applications, vol. 24, no. 3, May/June 1988, pp. 456–461.

[16] P. J. Lawrenson, J. M. Stephenson, P. T. Blenkinsop, J. Corda, and N. N. Fulton, "Variable-speed switch reluctance motors", IEE Proceedings, vol. 127, part. B., no. 4, July 1980, pp. 253–265.

[17] A. Lumsdaine, *Control of a Variable Reluctance Motor Based on State Observation*, M.S. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1985.

[18] PC-MATLAB User's Guide, The MathWorks Inc., June 1987.

[19] Leonard Meirovitch, *Analytical Methods in Vibrations*, Macmillan Publishing Co., New York, 1967.

[20] Microsoft C 5.0 Programmer's Guide, Microsoft Corporation, Washington, 1987.

[21] T. J. E. Miller, "Converter Volt-Ampere Requirements of the Switched Reluctance Motor Drive", IEEE Transactions on Industry Applications, vol. IA-21, no. 5, September/October 1985, pp. 1136–1144.

[22] W. F. Ray, P. J. Lawrenson, R. M. Davis, J. M. Stephenson, N. N. Fulton, and R. J. Blake, "High-Performance Switched Reluctance Brushless Drives," IEEE Transactions on Industry Applications, vol. IA-22, no. 4, July/August 1986, pp. 722–729.

[23] R. K. Singal and K. Williams, "A Theoretical and Experimental Study of Vibrations of Thick Circular Cylindrical Shells and Rings", Journal of Vibration, Acoustics, Stress, and Reliability in Design, vol. 110, October 1988, pp. 533–537.

[24] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science, vol. 6, 2nd ed., Springer-Verlag, 1976.

[25] S. P. Timoshenko and J. N. Goodier, *Theory of Elasticity*, 3rd ed., McGraw-Hill Book Company, New York, 1970.

[26] D. A. Torrey, *Optimal-Efficiency Constant-Speed Control of Nonlinear Variable Reluctance Motor Drives*, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, January 1988.

[27] S. P. Verma and R. S. Girgis, "Resonance Frequencies of Electrical Machine Stators Having Encased Construction, Part I: Derivation of the General Frequency Equation", IEEE Transactions, PES-92, January 1973, pp. 1577–1585.

[28] H. H. Woodson and J. R. Melcher, *Electromechanical Dynamics*, John Wiley & Sons, Inc., New York, 1968.

[29] S. J. Yang and A. J. Ellison, *Machinery Noise Measurement*, Oxford University Press, New York, 1985.

# Appendix A

# Integrals Used in the Determination of the Kinetic and Potential Energies of the Stator

## A.1 Notation

The integrals used to determine the kinetic and potential energies of the stator are presented in this appendix. In order to make the equations in Chapter 4 more readable, the following notation has been defined:

$$\mathcal{R}_1 \equiv \frac{1-r_n^{j+l+2}}{j+l+2} \qquad \mathcal{R}_2 \equiv \frac{1-r_n^{j+l+1}}{j+l+1}$$

$$\mathcal{R}_3 \equiv \begin{cases} -\ln(r_n) & \text{if } j+l=0 \\ \frac{1-r_n^{j+l}}{j+l} & \text{otherwise} \end{cases} \qquad \mathcal{Z}_1 \equiv \begin{cases} \frac{2l_n^{i+k+1}}{i+k+1} & \text{if } i+k \text{ is even} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathcal{Z}_2 \equiv \begin{cases} \frac{2l_n^{i+k}}{i+k} & \text{if } i+k \text{ is odd} \\ 0 & \text{otherwise} \end{cases} \qquad \mathcal{Z}_3 \equiv \begin{cases} 0 & \text{if } i+k \text{ is odd} \\ 0 & \text{if } i+k=0 \\ \frac{2l_n^{i+k-1}}{i+k-1} & \text{otherwise} \end{cases}$$

$$\mathcal{P}_1 \equiv \int_0^{2\pi} \cos^2(\eta\theta)d\theta = \begin{cases} \pi & \eta \neq 0 \\ 2\pi & \eta = 0 \end{cases}$$

$$\mathcal{P}_2 \equiv \int_0^{2\pi} \sin^2(\eta\theta)d\theta = \begin{cases} \pi & \eta \neq 0 \\ 0 & \eta = 0 \end{cases}$$

$$\sum_{ijkl}^{M-1} \equiv \sum_{i=0}^{M-1}\sum_{j=0}^{M-1}\sum_{k=0}^{M-1}\sum_{l=0}^{M-1} \qquad \sum_{gh}^{N-1} \equiv \sum_{g=0}^{N-1}\sum_{h=0}^{N-1}$$

## A.2 Integrals

The integrals presented here appear in the order that they are used in the determination of the kinetic and potential energies of the stator.

$$\begin{aligned} \int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \phi_r^g\,\phi_r^h\, r dr d\theta dz &= \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl} \int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} r^{j+l+1}\, z^{i+k}\, \cos^2(\eta\theta)\, dr d\theta dz \\ &= \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\; \mathcal{R}_1\mathcal{P}_1\mathcal{Z}_1 \qquad \text{(A.1)} \end{aligned}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \phi_\theta^g\,\phi_\theta^h\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{b}_{ij}\,\overset{h}{b}_{kl}\; \mathcal{R}_1\mathcal{P}_2\mathcal{Z}_1 \qquad \text{(A.2)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \phi_z^g\,\phi_z^h\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{c}_{ij}\,\overset{h}{c}_{kl}\; \mathcal{R}_1\mathcal{P}_1\mathcal{Z}_1 \qquad \text{(A.3)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_r^g}{\partial r}\frac{\partial\phi_r^h}{\partial r}\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl} \int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} j\, l\, r^{j+l-1}\, z^{i+k}\, \cos^2(\eta\theta)\, dr d\theta dz$$

$$= \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\; j\, l\, \mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \qquad \text{(A.4)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_r^g}{\partial r}\frac{\phi_r^h}{r}\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl} \int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} j\, r^{j+l-1}\, z^{i+k}\, \cos^2(\eta\theta)\, dr d\theta dz$$

$$= \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\; j\, \mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \qquad \text{(A.5)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_r^g}{\partial r}\frac{\partial\phi_\theta^h}{r\partial\theta}\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{b}_{kl}\; j\,\eta\, \mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \qquad \text{(A.6)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_r^g}{\partial r}\frac{\partial\phi_z^h}{\partial z}\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{c}_{kl}\; j\, k\, \mathcal{R}_2\mathcal{P}_1\mathcal{Z}_2 \qquad \text{(A.7)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\phi_r^g}{r}\frac{\phi_r^h}{r}\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\; \mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \qquad \text{(A.8)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\phi_r^g}{r}\frac{\partial\phi_\theta^h}{r\partial\theta}\, r dr d\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{b}_{kl}\; \eta\, \mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \qquad \text{(A.9)}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_\theta^g}{r\partial\theta}\frac{\partial\phi_\theta^h}{r\partial\theta}\, rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{b_{ij}}\,\overset{h}{b_{kl}}\;\eta^2\mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \tag{A.10}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\phi_r^g}{r}\frac{\partial\phi_r^h}{\partial r}\, rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a_{ij}}\,\overset{h}{a_{kl}}\; l\mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \tag{A.11}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\phi_r^g}{r}\frac{\partial\phi_z^h}{\partial z}\, rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a_{ij}}\,\overset{h}{c_{kl}}\; k\mathcal{R}_2\mathcal{P}_1\mathcal{Z}_2 \tag{A.12}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_\theta^g}{r\partial\theta}\frac{\partial\phi_z^h}{\partial z}\, rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{b_{ij}}\,\overset{h}{c_{kl}}\; k\eta\mathcal{R}_2\mathcal{P}_1\mathcal{Z}_2 \tag{A.13}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_z^g}{\partial z}\frac{\partial\phi_z^h}{\partial z}\, rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{c_{ij}}\,\overset{h}{c_{kl}}\; ik\mathcal{R}_1\mathcal{P}_1\mathcal{Z}_3 \tag{A.14}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_r^g}{r\partial\theta}\frac{\partial\phi_r^h}{r\partial\theta}\, rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a_{ij}}\,\overset{h}{a_{kl}}\; \eta^2\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.15}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1} \frac{\partial\phi_r^g}{r\partial\theta}\frac{\partial\phi_\theta^h}{\partial r}\, rdrd\theta dz = -\sum_{ijkl}^{M-1} \overset{g}{a_{ij}}\,\overset{h}{b_{kl}}\; l\eta\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.16}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_r^g}{r\partial\theta}\,\frac{\phi_\theta^h}{r}\,rdrd\theta dz = -\sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{b}_{kl}\;\eta\,\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.17}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_\theta^g}{\partial r}\,\frac{\partial\phi_\theta^h}{\partial r}\,rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{b}_{ij}\,\overset{h}{b}_{kl}\;j\,l\,\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.18}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_\theta^g}{\partial r}\,\frac{\phi_\theta^h}{r}\,rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{b}_{ij}\,\overset{h}{b}_{kl}\;j\,\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.19}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\phi_\theta^g}{r}\,\frac{\partial\phi_\theta^h}{\partial r}\,rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{b}_{ij}\,\overset{h}{b}_{kl}\;l\,\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.20}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\phi_\theta^g}{r}\,\frac{\phi_\theta^h}{r}\,rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{b}_{ij}\,\overset{h}{b}_{kl}\;\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.21}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_r^g}{\partial z}\,\frac{\partial\phi_r^h}{\partial z}\,rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\;i\,k\,\mathcal{R}_1\mathcal{P}_1\mathcal{Z}_3 \tag{A.22}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_r^g}{\partial z}\,\frac{\partial\phi_z^h}{\partial r}\,rdrd\theta dz = \sum_{ijkl}^{M-1} \overset{g}{a}_{ij}\,\overset{h}{c}_{kl}\;i\,l\,\mathcal{R}_2\mathcal{P}_1\mathcal{Z}_2 \tag{A.23}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_z^g}{\partial r}\frac{\partial\phi_z^h}{\partial r}\,rdrd\theta dz=\sum_{ijkl}^{M-1} c_{ij}^{g}\,c_{kl}^{h}\;j\,l\,\mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1 \tag{A.24}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_\theta^g}{\partial z}\frac{\partial\phi_\theta^h}{\partial z}\,rdrd\theta dz=\sum_{ijkl}^{M-1} b_{ij}^{g}\,b_{kl}^{h}\;i\,k\,\mathcal{R}_1\mathcal{P}_2\mathcal{Z}_3 \tag{A.25}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_\theta^g}{\partial z}\frac{\partial\phi_z^h}{r\partial\theta}\,rdrd\theta dz=-\sum_{ijkl}^{M-1} b_{ij}^{g}\,c_{kl}^{h}\;i\,\eta\,\mathcal{R}_2\mathcal{P}_2\mathcal{Z}_2 \tag{A.26}$$

$$\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\frac{\partial\phi_z^g}{r\partial\theta}\frac{\partial\phi_z^h}{r\partial\theta}\,rdrd\theta dz=\sum_{ijkl}^{M-1} c_{ij}^{g}\,c_{kl}^{h}\;\eta^2\,\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \tag{A.27}$$

## A.3 Integration of the Potential Energy Equation for the Stator Cylinder

The term-by-term integration of the potential energy equation is given here. For the convenience of the reader, Equation 4.22 is repeated below

$$PE_{cylinder}=\frac{1}{2}\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}(\sigma_r\epsilon_r+\sigma_\theta\epsilon_\theta+\sigma_z\epsilon_z+\tau_{r\theta}\gamma_{r\theta}+\tau_{rz}\gamma_{rz}+\tau_{\theta z}\gamma_{\theta z})\,rdrd\theta dz \tag{A.28}$$

$$\sigma_r\epsilon_r=\frac{2\bar{G}}{1-2\mu}\left[(1-\mu)\left(\frac{\partial\mathcal{U}_r}{\partial r}\right)^2+\mu\frac{\partial\mathcal{U}_r}{\partial r}\left(\frac{\mathcal{U}_r}{r}+\frac{\partial\mathcal{U}_\theta}{r\partial\theta}+\frac{\partial\mathcal{U}_z}{\partial z}\right)\right]$$

$$= \frac{2\bar{G}}{1-2\mu}\sum_{gh}^{N-1} q_g q_h \left[(1-\mu)\frac{\partial\phi_r^g}{\partial r}\frac{\partial\phi_r^h}{\partial r} + \mu\left(\frac{\partial\phi_r^g}{\partial r}\frac{\phi_r^h}{r} + \frac{\partial\phi_r^g}{\partial r}\frac{\partial\phi_\theta^h}{r\partial\theta} + \frac{\partial\phi_r^g}{\partial r}\frac{\partial\phi_z^h}{\partial z}\right)\right]$$

$$\frac{1}{2}\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\sigma_r\epsilon_r r dr d\theta dz = \frac{\bar{G}}{1-2\mu}\sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1}$$

$$\left[\left\{\overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\ (\mu j + (1-\mu)jl) + \overset{g}{a}_{ij}\,\overset{h}{b}_{kl}\ \mu\eta j\right\}\mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1\right.$$

$$\left. + \overset{g}{a}_{ij}\,\overset{h}{c}_{kl}\ \mu j k\,\mathcal{R}_2\mathcal{P}_1\mathcal{Z}_2\right] \tag{A.29}$$

$$\sigma_\theta\epsilon_\theta = \frac{2\bar{G}}{1-2\mu}\left[(1-\mu)\left(\frac{\phi_r^g}{r}\frac{\phi_r^h}{r} + 2\frac{\phi_r^g}{r}\frac{\partial\phi_\theta^h}{r\partial\theta} + \frac{\partial\phi_\theta^g}{r\partial\theta}\frac{\partial\phi_\theta^h}{r\partial\theta}\right)\right.$$

$$\left. + \mu\left(\frac{\phi_r^g}{r}\frac{\partial\phi_r^h}{\partial r} + \frac{\phi_r^g}{r}\frac{\partial\phi_z^h}{\partial z} + \frac{\partial\phi_r^g}{\partial r}\frac{\partial\phi_\theta^h}{r\partial\theta} + \frac{\partial\phi_\theta^g}{r\partial\theta}\frac{\partial\phi_z^h}{\partial z}\right)\right]$$

$$\frac{1}{2}\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\sigma_\theta\epsilon_\theta r dr d\theta dz = \frac{\bar{G}}{1-2\mu}\sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1}$$

$$\left[\left\{\overset{g}{a}_{ij}\,\overset{h}{a}_{kl}\ (\mu l + (1-\mu)) + \overset{g}{a}_{ij}\,\overset{h}{b}_{kl}\ \eta(\mu j + 2(1-\mu)) + \overset{g}{b}_{ij}\,\overset{h}{b}_{kl}\ (1-\mu)\eta^2\right\}\mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1\right.$$

$$+ \left\{ \overset{g}{a}_{ij} \overset{h}{c}_{kl} \ \mu k + \overset{g}{b}_{ij} \overset{h}{c}_{kl} \ \mu k\eta \right\} \mathcal{R}_2 \mathcal{P}_1 \mathcal{Z}_2 \Bigg] \tag{A.30}$$

$$\sigma_z \epsilon_z = \frac{2\bar{G}}{1-2\mu} \sum_{gh}^{N-1} q_g q_h \left[ (1-\mu) \frac{\partial \phi_z^g}{\partial z} \frac{\partial \phi_r^h}{\partial z} + \mu \left( \frac{\partial \phi_r^g}{\partial r} \frac{\partial \phi_z^h}{\partial z} + \frac{\phi_r^g}{r} \frac{\partial \phi_z^h}{\partial z} + \frac{\partial \phi_\theta^g}{r\partial \theta} \frac{\partial \phi_z^h}{\partial z} \right) \right]$$

$$\frac{1}{2} \int_{-l_n}^{l_n} \int_0^{2\pi} \int_{r_n}^{1} \sigma_z \epsilon_z r dr d\theta dz = \frac{\bar{G}}{1-2\mu} \sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1}$$

$$\left[ \left\{ \overset{g}{a}_{ij} \overset{h}{c}_{kl} \ \mu k(1+j) + \overset{g}{b}_{ij} \overset{h}{c}_{kl} \ \mu k\eta \right\} \mathcal{R}_2 \mathcal{P}_1 \mathcal{Z}_2 \right.$$

$$\left. + \overset{g}{c}_{ij} \overset{h}{c}_{kl} \ (1-\mu) ik \mathcal{R}_1 \mathcal{P}_1 \mathcal{Z}_3 \right] \tag{A.31}$$

$$\tau_{r\theta} \gamma_{r\theta} = \bar{G} \sum_{gh}^{N-1} q_g q_h \left[ \frac{\partial \phi_r^g}{r\partial \theta} \frac{\partial \phi_r^h}{r\partial \theta} + 2 \frac{\partial \phi_r^g}{r\partial \theta} \frac{\partial \phi_\theta^h}{\partial r} - 2 \frac{\partial \phi_r^g}{r\partial \theta} \frac{\phi_\theta^h}{r} \right.$$

$$\left. + \frac{\partial \phi_\theta^g}{\partial r} \frac{\partial \phi_\theta^h}{\partial r} - \frac{\partial \phi_\theta^g}{\partial r} \frac{\phi_\theta^h}{r} - \frac{\phi_\theta^g}{r} \frac{\partial \phi_\theta^h}{\partial r} + \frac{\phi_\theta^g}{r} \frac{\phi_\theta^h}{r} \right]$$

$$\frac{1}{2} \int_{-l_n}^{l_n} \int_0^{2\pi} \int_{r_n}^{1} \tau_{r\theta} \gamma_{r\theta} r dr d\theta dz = \frac{\bar{G}}{2} \sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1}$$

$$\left[\overset{g}{a}_{ij}\overset{h}{a}_{kl}\ \eta^2 + \overset{g}{a}_{ij}\overset{h}{b}_{kl}\ 2\eta(1-l) + \overset{g}{b}_{ij}\overset{h}{b}_{kl}\ (j-1)(l-1)\right]\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1 \qquad (A.32)$$

$$\tau_{rz}\gamma_{rz} = \bar{G}\sum_{gh}^{N-1} q_g q_h \left[\frac{\partial\phi_r^g}{\partial z}\frac{\partial\phi_r^h}{\partial z} + 2\frac{\partial\phi_r^g}{\partial z}\frac{\partial\phi_z^h}{\partial r} + \frac{\partial\phi_z^g}{\partial r}\frac{\partial\phi_z^h}{\partial r}\right]$$

$$\frac{1}{2}\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\tau_{rz}\gamma_{rz}\,r\,dr\,d\theta\,dz = \frac{\bar{G}}{2}\sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1}$$

$$\left[\overset{g}{a}_{ij}\overset{h}{a}_{kl}\ ik\,\mathcal{R}_1\mathcal{P}_1\mathcal{Z}_3 + \overset{g}{a}_{ij}\overset{h}{c}_{kl}\ 2il\,\mathcal{R}_2\mathcal{P}_1\mathcal{Z}_2 + \overset{g}{c}_{ij}\overset{h}{c}_{kl}\ jl\,\mathcal{R}_3\mathcal{P}_1\mathcal{Z}_1\right] \qquad (A.33)$$

$$\tau_{\theta z}\gamma_{\theta z} = \bar{G}\sum_{gh}^{N-1} q_g q_h \left[\frac{\partial\phi_\theta^g}{\partial z}\frac{\partial\phi_\theta^h}{\partial z} + 2\frac{\partial\phi_\theta^g}{\partial z}\frac{\partial\phi_z^h}{r\partial\theta} + \frac{\partial\phi_z^g}{r\partial\theta}\frac{\partial\phi_z^h}{r\partial\theta}\right]$$

$$\frac{1}{2}\int_{-l_n}^{l_n}\int_0^{2\pi}\int_{r_n}^{1}\tau_{\theta z}\gamma_{\theta z}\,r\,dr\,d\theta\,dz = \frac{\bar{G}}{2}\sum_{gh}^{N-1} q_g q_h \sum_{ijkl}^{M-1}$$

$$\left[\overset{g}{b}_{ij}\overset{h}{b}_{kl}\ ik\,\mathcal{R}_1\mathcal{P}_2\mathcal{Z}_3 - \overset{g}{b}_{ij}\overset{h}{c}_{kl}\ 2i\eta\,\mathcal{R}_2\mathcal{P}_2\mathcal{Z}_2 + \overset{g}{c}_{ij}\overset{h}{c}_{kl}\ \eta^2\,\mathcal{R}_3\mathcal{P}_2\mathcal{Z}_1\right] \qquad (A.34)$$

# Appendix B

# Circuit Diagrams for the Experimental VRM Drive

The circuit diagrams for the controller and inverter along with additional support circuitry are given in this Appendix. The controller consists of an 8032 microcontroller based system which communicates with a personal computer (PC) through a serial link. Based on parameters supplied by the user and position information supplied by the shaft encoder circuitry, the controller generates the phase on/off signals and the current chopping level. The inverter circuitry is divided into two parts. The first part (referred to as the inverter control circuitry) uses the signals generated by the controller along with measured values of the phase currents to generate the turn-on/off signals for the FET pairs of each phase. The second part is comprised of the power electronic and current sensing circuitry. The shaft encoder circuitry is used to translate the signals from the shaft encoder into 8 bits of position information that are used by the controller.

The connections between the various parts of the VRM drive are given in the following tables.

| controller | inverter control circuitry |
|---|---|
| 14T | 17 |
| 15T | 19 |
| 16T | 27 |
| 17T | 29 |
| 18T | 28 |
| 19T | 30 |
| 20T | 31 |

Table B.1: Connections between the controller and the inverter control circuitry. The suffix "T" refers to the top pins of the edge connector.

| controller | shaft encoder circuitry |
|---|---|
| J1-1 | J1-1 |
| J1-7 | J1-7 |
| J1-8 | J1-8 |
| J1-9 | J1-9 |
| J1-10 | J1-10 |
| J1-11 | J1-11 |
| J1-12 | J1-12 |
| J1-13 | J1-13 |
| J1-14 | J1-14 |
| J1-15 | J1-15 |
| J1-16 | J1-16 |

Table B.2: Connections between the controller and the shaft encoder circuitry. The prefix "J" indicates that the connection is made through a jumper cable.

| inverter control circuitry | inverter power circuitry |
|---|---|
| J1-4 | J1-4 |
| J1-5 | J1-5 |
| J1-6 | J1-6 |
| J1-7 | J1-7 |
| J1-8 | J1-8 |
| J1-9 | J1-9 |
| J1-10 | J1-10 |
| J1-11 | J1-11 |
| J1-12 | J1-12 |
| J2-1 | J2-1 |
| J2-2 | J2-2 |
| J2-3 | J2-3 |
| J2-4 | J2-4 |

Table B.3: Connections between the inverter control and power circuitry. The prefix "J" indicates that the connection is made through a jumper cable.

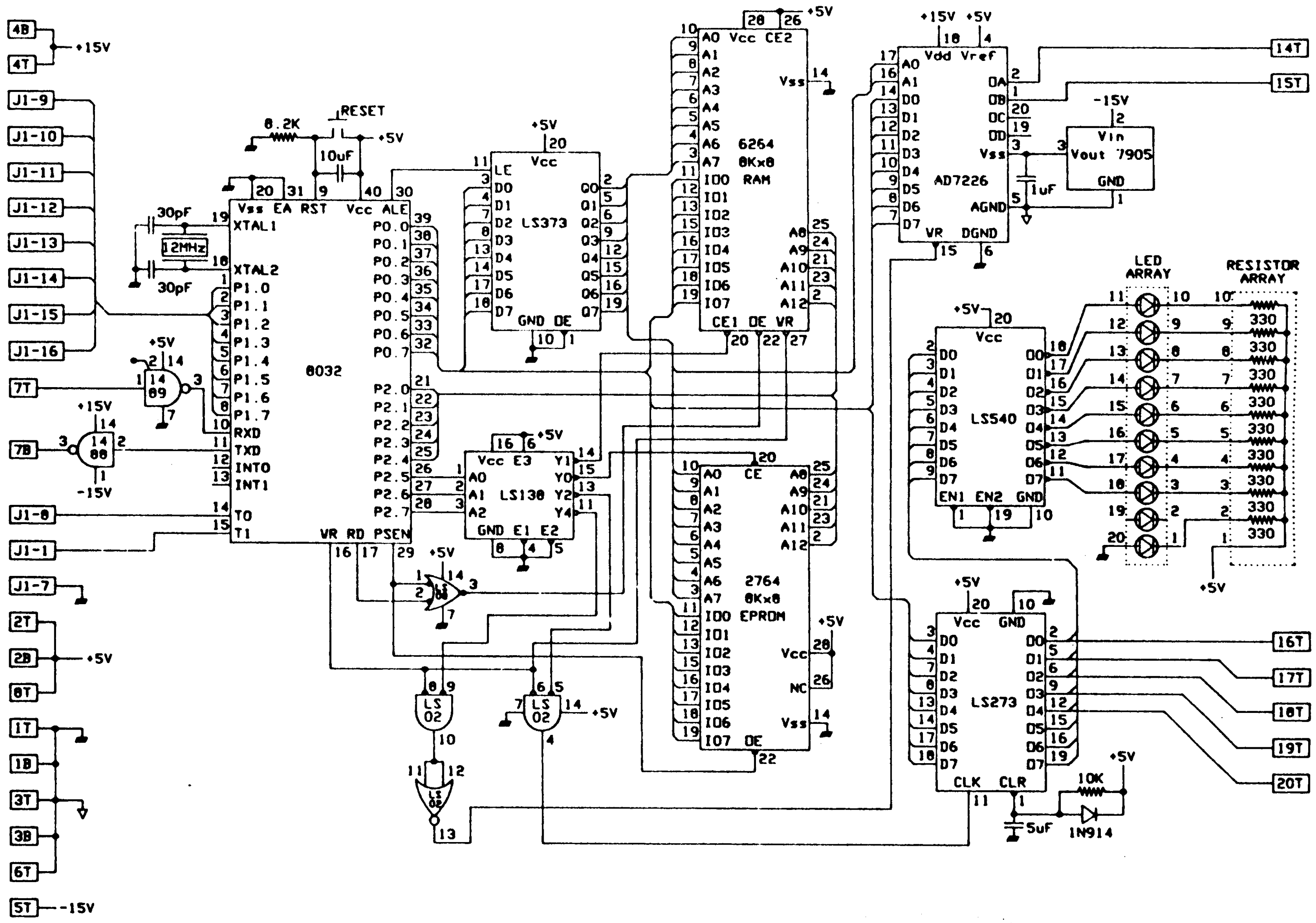

Figure B.1: Controller Circuitry

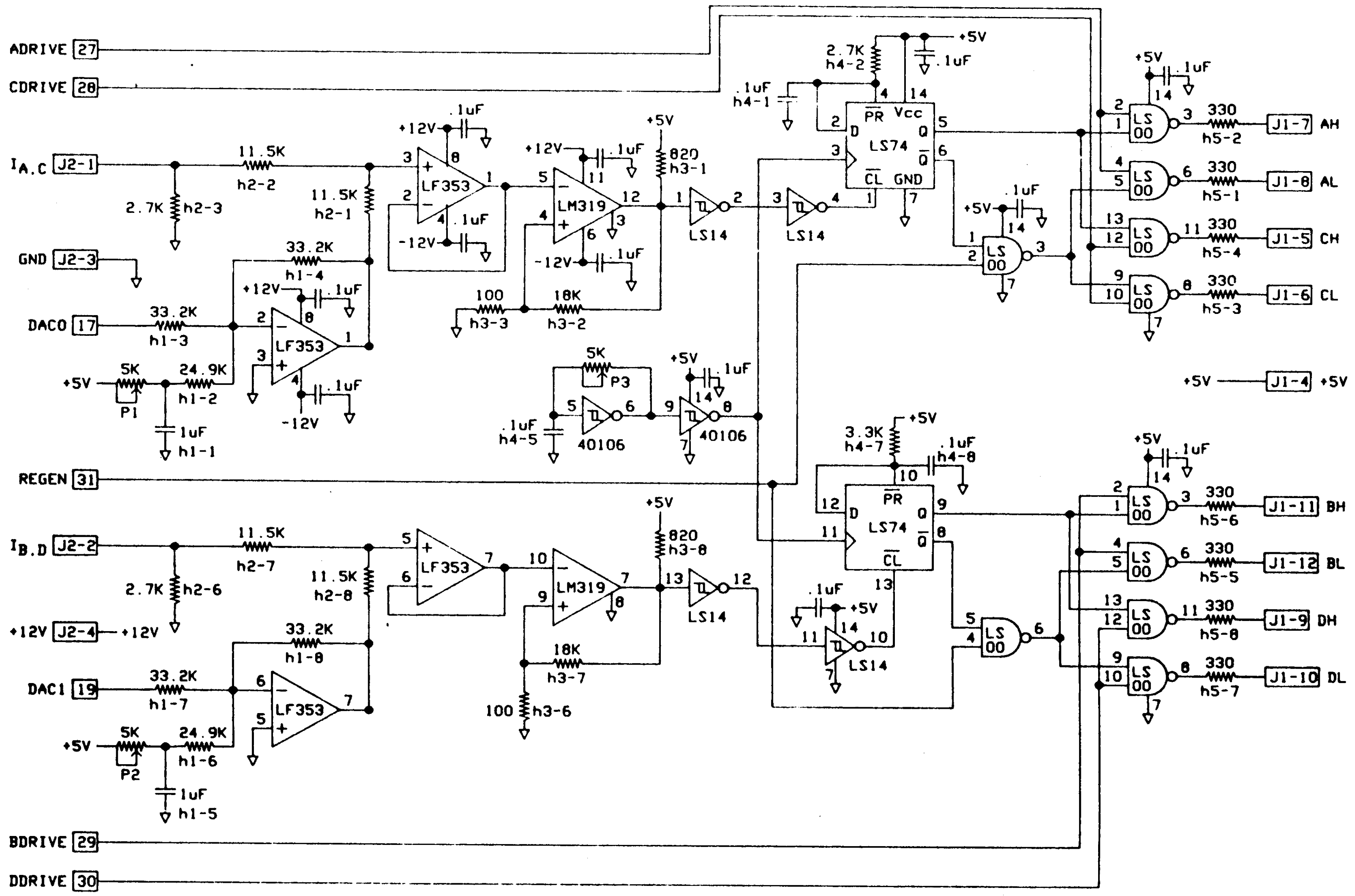

Figure B.2: Inverter Control Circuitry

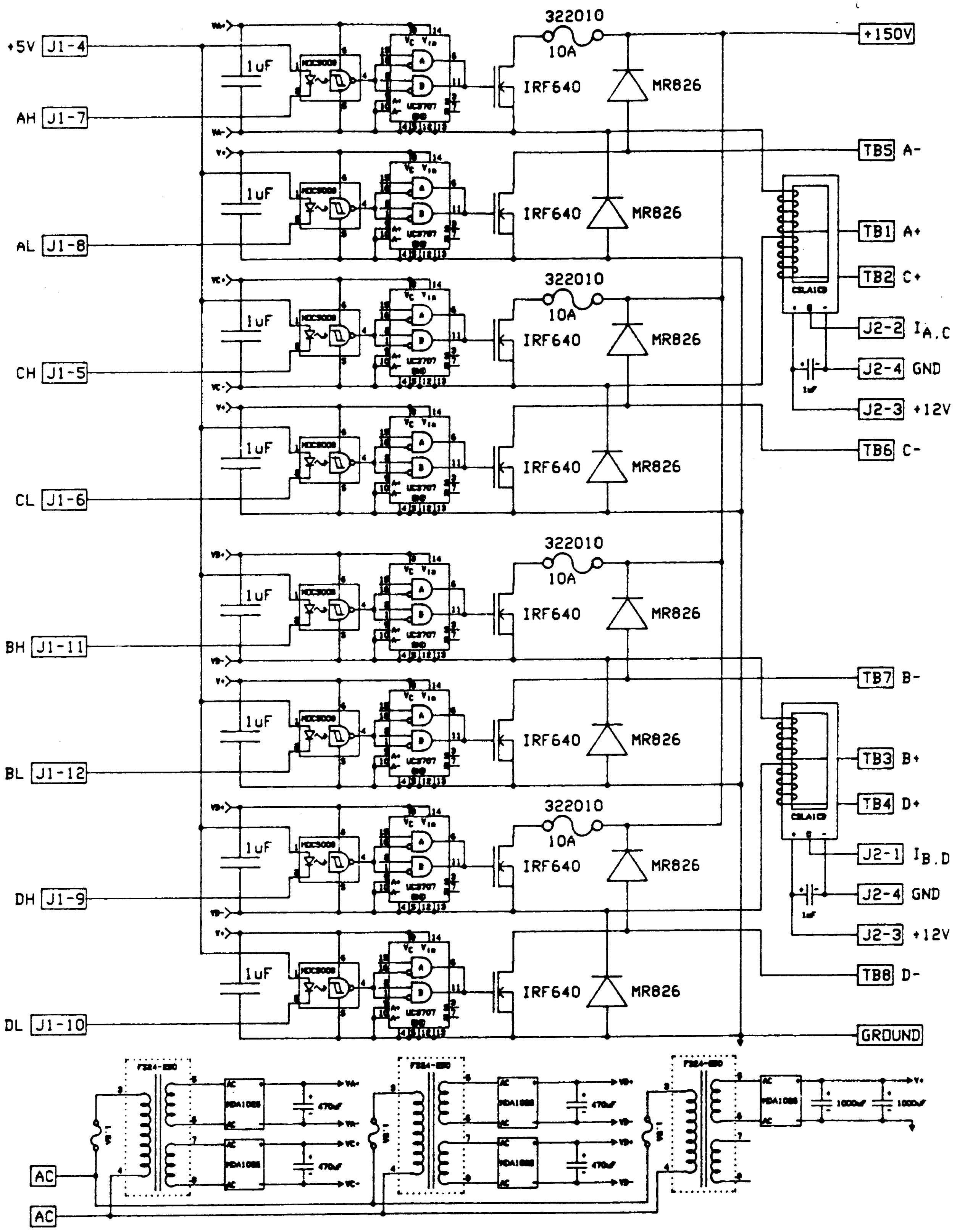

Figure B.3: Inverter Power Circuitry

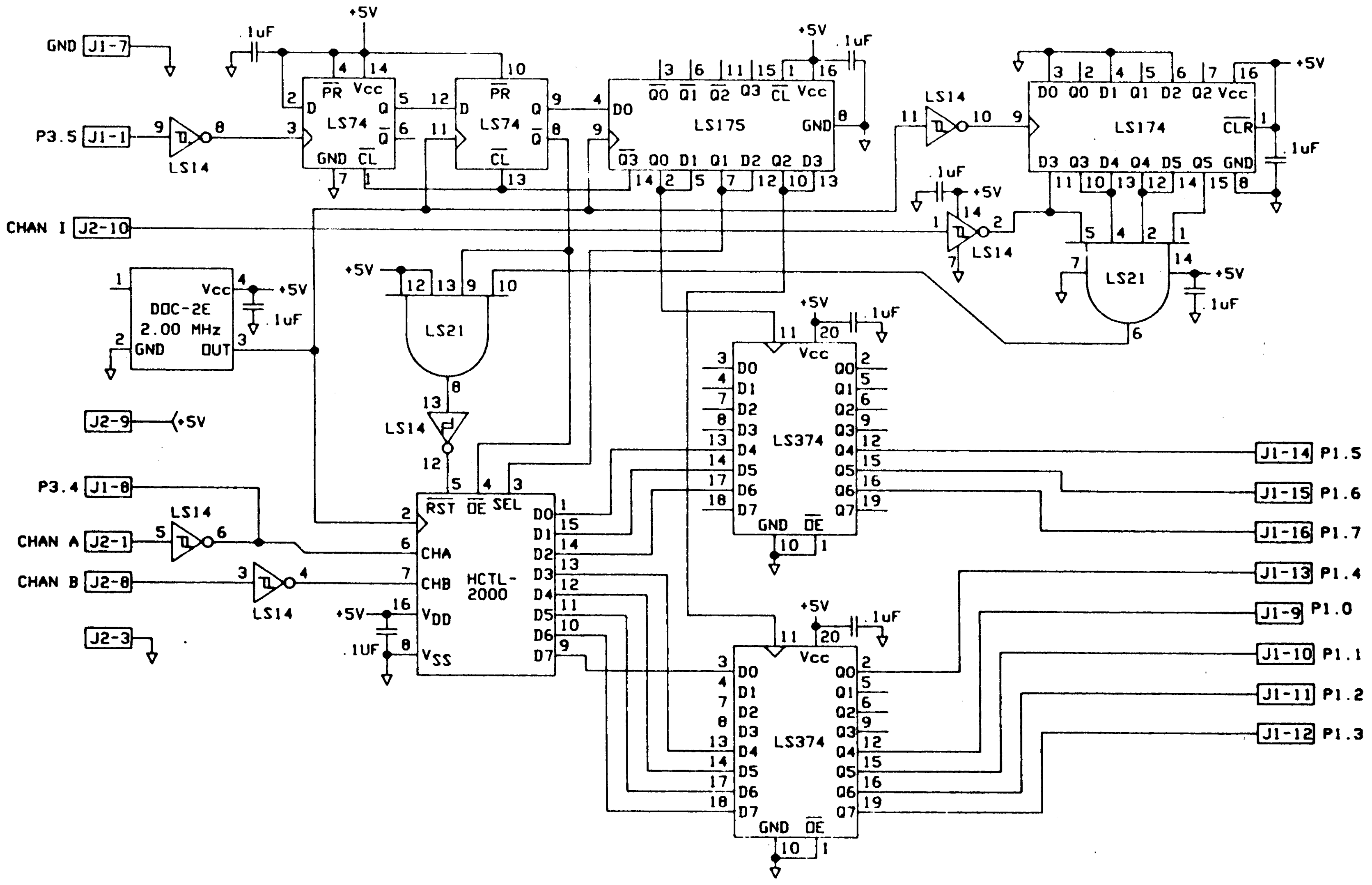

Figure B.4: Shaft Encoder Circuitry

# Appendix C

# Source Code for the 8032 Controller Board

## C.1 Monitor Program

The monitor program MONITOR.MSA resides in the 2764 EPROM on the controller board. It provides an interface between the controller and the PC using a serial link. This allows programs and data to be transfered between the PC and the controller.

The monitor program can be assembled and linked using the Enertec A8051 Cross Assembler and XLINK linker for MS-DOS based computers. Since the EPROM is mapped to the lowest 8K section of the controller's memory, the code segment is set to zero.

```
**************************************************************************
*                                                                        *
*                           MONITOR.MSA ver. 1.0                         *
*                                                                        *
**************************************************************************

;*******************PROGRAM CONSTANTS AND DEFINITIONS*******************

NULL    EQU     00H     ;ascii for control char NULL
BELL    EQU     07H     ;ascii for bell
BS      EQU     08H     ;ascii for back space in hex
LF      EQU     0AH     ;ascii for line feed in hex
CR      EQU     0DH     ;ascii for carriage return in hex
```

```
PROMPT  EQU     3EH     ;ascii for the prompt '>' in hex
DEL     EQU     7FH     ;ascii for delete in hex
ISP     EQU     7FH     ;reset stack location
T2CON   EQU     0C8H    ;this SFR is not defined in the assembler

;The stack resides in the upper half of internal RAM (80H-FFH). The SFRs
;also lie within this address range, but are accessible only by using
;direct addressing. Following are the internal memory locations defined
;for use as variables etc.

ARG1LOW EQU     20H     ;lower byte of 1st argument of an instruction
ARG1HIGH EQU    21H     ;higher byte of 1st argument of an instruction
ARG2LOW  EQU    22H     ;lower byte of 2nd argument of an instruction
ARG2HIGH EQU    23H     ;higher byte of 2nd argument of an instruction
ORIGIN  EQU     24H     ;var. used to store the stack pointer position
CHKSUM  EQU     25H     ;keeps track of checksum in INTEL hex format
RECLEN  EQU     26H     ;to store the record length in INTEL hex form.

;The following are external memory locations.

RAM     EQU     2000H   ;external RAM starting address
DAC     EQU     4000H   ;digital to analog converter base address
LATCH   EQU     8000H   ;address of LS273 latch

;***********************INTERRUPT VECTOR TABLE*****************************

     RSEG    CODE
LOCO:        ;This is location 0H or 2000H depending on loading method.

     JMP     INITIAL            ;reset vector address

     ASEG                       ;The following vectors always reside in the
                                ;1st 8K of memory
     ORG     03H                ;external interrupt 0 vector address
     LJMP    $ + RAM

     ORG     0BH                ;timer0 vector address
     LJMP    $ + RAM

     ORG     13H                ;external interrupt 1 vector address
     LJMP    $ + RAM

     ORG     1BH                ;timer1 vector address
     LJMP    $ + RAM
     ORG      23H                  ;serial port vector address
     LJMP     $ + RAM

     ORG      2BH                  ;timer2 vector address
     LJMP     $ + RAM

;**********************INITIALIZATION ROUTINE*****************************

     RSEG     CODE                 ;resume coding in relocatable segment
     ORG      LOCO+30H             ;code begins after interrupt vectors
INITIAL:
     CLR      A                    ;clear the accumulator
     MOV      IE,A                 ;disable all interrupts
     MOV      DPTR,#LATCH          ;get address of latch
     MOVX     @DPTR,A              ;clear the latch
     MOV      PSW,A                ;select register bank 0
     MOV      DPTR,#DAC            ;get address of DAC
     MOVX     @DPTR,A              ;clear DAC A
     INC      DPTR                 ;select DAC B
     MOVX     @DPTR,A              ;clear DAC B
     INC      DPTR                 ;select DAC C
     MOVX     @DPTR,A              ;clear DAC C
     INC      DPTR                 ;select DAC D
     MOVX     @DPTR,A              ;clear DAC D
     MOV      R0,#0FFH             ;prepare to clear all internal RAM except
INITIAL1:                          ;for the SFRs
     MOV      @R0,A                ;clear internal RAM pointed to by R0
     DJNZ     R0,INITIAL1          ;continue until done
     MOV      SP,#ISP              ;initialize the stack pointer to 80H
     MOV      TMOD,#23H            ;timer1-mode 2, timer0-mode3, no gating
     MOV      TH1,#230             ;load #230 into timer 1 higher byte for a
                                   ;baud rate of 1200
     MOV      SCON,#078H           ;mode 2, 1 start, 8 data, and 1 stop bit
     SETB     TI                   ;enable serial transmission
     CLR      RI                   ;clear the receive interrupt flag if set
BANNER:
     MOV      DPTR,#RESET_MSG
     CALL     SSTRING              ;display the reset banner
CMD_SUMM:
     MOV      DPTR,#HELP_MSG
     CALL      SSTRING             ;display the command summary
     MOV      ORIGIN,SP            ;initialize ORIGIN variable

;***************************THE COMMAND LEVEL****************************
```

```
;Characters are read in from the console and stored on the stack, which is
;used as the command buffer. The characters are properly echoed and a
;DELETE operation can be performed by using the backspace key.

NEXT:
      MOV     SP,ORIGIN           ;reset the stack pointer
      CLR     RI                  ;clear the receive interrupt flag
      CALL    CR_LF               ;send out a carriage return and line feed
      MOV     A,#PROMPT           ;put the prompt '>' out
      CALL    SP_OUT
NEXT1:
      CALL    SP_IN               ;get the 1st char of the command
      CJNE    A,#BS,NEXT2         ;jump if it isn't a back space
      JMP     NEXT1               ;wait for another char
NEXT2:
      CJNE    A,#CR,NEXT3         ;see if the char was a carriage return
      JMP     NEXT                ;if so, return to top of command level
NEXT3:
      MOV     ORIGIN,SP           ;if not, save the current value of SP
      PUSH    ACC                 ;save the char in the command buffer
      CALL    SP_OUT              ;echo it to the console
NEXT4:
      CALL    SP_IN               ;wait for another char
      CJNE    A,#CR,NEXT5         ;jump if it isn't a CR
      PUSH    ACC                 ;put the CR in the command buffer
      CALL    CR_LF               ;echo a CR and LF to the console
      JMP     INTERP              ;jump to interpreter
NEXT5:
      CJNE    A,#BS,NEXT6         ;jump if char isn't a BS
      POP     ACC                 ;remove last char from command buffer
      MOV     A,#BS
      CALL    SP_OUT              ;move the console's cursor back one place
      MOV     A,#' '              ;send a SPACE to the console to erase the
      CALL    SP_OUT              ;last char typed
      MOV     A,#BS
      CALL    SP_OUT              ;move the console's cursor back again
      MOV     A,SP                ;get the current value of the stack pointer
      CJNE    A,ORIGIN,NEXT4      ;jump if command buffer is not empty
      JMP     NEXT1               ;go to top of command level
NEXT6:
      PUSH    ACC                 ;put the char in the command buffer
      CALL    SP_OUT              ;echo it to the console
      MOV     A,SP                ;get current value of the stack pointer
      CJNE    A,#(ISP+20),NEXT4   ;jump if command length < 20 chars
      MOV     DPTR,#ERROR3        ;get pointer to appropriate error message
```

```
     CALL    SSTRING             ;send it to the console
     JMP     NEXT                ;go to top of command level


;*************************THE INTERPRETATION LEVEL*************************

;Once the command buffer contains a complete input command, terminated by a
;<CR>, the command interpreter decodes it and dispatches control to the
;proper command handler.

INTERP:
     MOV     RO,ORIGIN           ;RO points to beginning of command buffer
     INC     RO
     MOV     A,@RO               ;get 1st char of the command
INTERP_u:
     CJNE    A,#'u',INTERP_U     ;jump if char isn't 'u'
     JMP     UPLOAD_INT          ;jump to UPLOAD INTERNAL RAM cmd handler
INTERP_U:
     CJNE    A,#'U',INTERP_d     ;jump if char isn't 'U'
     JMP     UPLOAD_EXT          ;jump to UPLOAD EXTERNAL RAM cmd handler
INTERP_d:
     CJNE    A,#'d',INTERP_g     ;jump if char isn't 'd'
     JMP     DNLOAD              ;jump to DOWNLOAD EXTERNAL RAM cmd handler
INTERP_g:
     CJNE    A,#'g',INTERP_h     ;jump if char isn't 'g'
     JMP     GO                  ;jump to GO ADDRESS command handler
INTERP_h:
     CJNE    A,#'h',INTERP_s     ;jump if char isn't 'h'
     JMP     HELP                ;jump to HELP command handler
INTERP_s:
     CJNE    A,#'s',INTERP_S     ;jump if char isn't 's'
     JMP     STORE_INT           ;jump to STORE INTERNAL MEMORY cmd handler
INTERP_S:
     CJNE    A,#'S',INTERP_c     ;jump if char isn't 'S'
     JMP     STORE_EXT           ;jump to STORE EXTERNAL MEMORY cmd handler
INTERP_c:
     CJNE    A,#'c',BAD_CMD      ;jump if char isn't 'c'
     JMP     CLEAR_EXT           ;jump to CLEAR EXTERNAL RAM command handler
BAD_CMD:
     MOV     DPTR,#ERROR1        ;point to proper error message
     CALL    SSTRING             ;print the error message
     JMP     NEXT                ;return to command level

;The HELP routine displays the help message and returns to the command
;level.
```

```
HELP:
     CALL    CHK_RET            ;see if a <CR> followed the 'h'
     MOV     SP,ORIGIN          ;reset SP to beginning of command buffer
     JMP     CMD_SUMM           ;display the help message.

;CLEAR_EXT clears the external RAM memory.

CLEAR_EXT:
     CALL    CHK_RET            ;see if a <CR> followed the 'c'
     MOV     DPTR,#RAM          ;prepare to clear all external RAM
CLEAR_EXT1:
     CLR     A                  ;clear ACC
     MOVX    @DPTR,A            ;clear external RAM pointed to by DPTR
     INC     DPTR               ;increment external memory pointer
     MOV     A,DPH              ;get high byte of DPTR
     CJNE    A,#40H,CLEAR_EXT1 ;continue until done
     JMP     NEXT               ;return to command level

;STORE_INT stores a byte of data in internal memory. The 1st argument
;specifies the internal memory location. The 2nd argument is the value to
;be written.

STORE_INT:
     CALL    GET_ARG            ;get the internal memory address
     MOV     ARG1LOW,A          ;save it in ARG1LOW
     INC     R0                 ;point to char following the address
     MOV     A,@R0              ;get it
     CJNE    A,#',',NO_COMMA    ;jump if it isn't a comma
     CALL    GET_ARG            ;get the data byte
     MOV     ARG2LOW,A          ;save it in ARG2LOW
     CALL    CHK_RET            ;see if 2nd argument was followed by a CR
     MOV     R0,ARG1LOW         ;move the address into R0
     MOV     A,ARG2LOW          ;get the data byte
     MOV     @R0,A              ;store it in internal memory
     JMP     NEXT               ;return to command level

;STORE_EXT stores a byte of data in external memory. The 1st argument
;specifies the external memory location. The 2nd argument is the value to
;be written.

STORE_EXT:
     CALL    GET_ARG            ;get high byte of the memory address
     MOV     ARG1HIGH,A         ;save it in ARG1HIGH
     CALL    GET_ARG            ;get low byte of the address
     MOV     ARG1LOW,A           ;save it in ARG1LOW
     INC     R0                  ;point to char following the address
     MOV     A,@R0               ;get it
     CJNE    A,#',',NO_COMMA     ;jump if it isn't a comma
     CALL    GET_ARG             ;get the data byte
     MOV     ARG2LOW,A           ;save it in ARG2LOW
     CALL    CHK_RET             ;see if 2nd argument was followed by a CR
     MOV     DPL,ARG1LOW         ;move the address into DPTR
     MOV     DPH,ARG1HIGH
     MOV     A,ARG2LOW           ;get the data byte
     MOVX    @DPTR,A             ;store it in external memory
     JMP     NEXT                ;return to command level

;UPLOAD_INT sends a block of internal RAM bytes to the console. The 1st
;argument is the starting address of the block. The 2nd argument is the #
;of bytes to be sent. If the command syntax is incorrect, the command will
;be aborted.

UPLOAD_INT:
     CALL    GET_ARG             ;get the starting address of the upload
     MOV     ARG1LOW,A           ;save it in ARG1LOW
     INC     R0                  ;point to the next char in the cmd buffer
     MOV     A,@R0               ;get it
     CJNE    A,#',',NO_COMMA     ;abort if it isn't a comma
     CALL    GET_ARG             ;get the number of bytes to be uploaded
     MOV     ARG2LOW,A           ;save it in ARG2LOW
     CALL    CHK_RET             ;make sure a <CR> follows the second arg
     MOV     R0,ARG1LOW          ;move the starting address into R0
UPLOAD_INT1:
     MOV     A,@R0               ;move byte pointed to by R0 into the ACC
     INC     R0                  ;increment R0 to point to the next byte
     CALL    SEND_BYTE           ;send the byte to the console
     DJNZ    ARG2LOW,UPLOAD_INT1 ;continue until all bytes have been sent
UPLOAD_INT2:
     JMP     NEXT                ;return to the command level

;NO_COMMA is the abort routine if a comma in a command line is missing

NO_COMMA:
     MOV     DPTR,#ERROR4        ;point to proper error message
     CALL    SSTRING             ;send it to the console
     JMP     NEXT                ;return to the command level


;UPLOAD_EXT sends a block of external memory to the console. The 1st
```

```
;2-byte argument is the starting address of the block. The 2nd 2-byte
;argument is the number of bytes to be sent. The routine aborts if the
;command syntax is incorrect.

UPLOAD_EXT:
     CALL    GET_ARG              ;get high byte of the memory address
     MOV     ARG1HIGH,A           ;save it in ARG1HIGH
     CALL    GET_ARG              ;get low byte of the address
     MOV     ARG1LOW,A            ;save it in ARG1LOW
     INC     R0                   ;point to next char in the command buffer
     MOV     A,@R0                ;get it
     CJNE    A,#',',NO_COMMA      ;abort if it is not a comma
     CALL    GET_ARG              ;get high byte of the block length
     MOV     ARG2HIGH,A           ;save it in ARG2HIGH
     CALL    GET_ARG              ;get low byte of the block length
     MOV     ARG2LOW,A            ;save it in ARG2LOW
     CALL    CHK_RET              ;make sure a <CR> follows the 2nd argument
     MOV     DPL,ARG1LOW          ;load DPTR with the address of the block
     MOV     DPH,ARG1HIGH
UPLOAD_EXT1:
     CLR     A                    ;clear accumulator
     MOVC    A,@A + DPTR          ;get the byte pointed to by the data pointer
     CALL    SEND_BYTE            ;send it to the console
     INC     DPTR                 ;increment DPTR to point to the next byte
     CLR     C                    ;clear carry bit for subtraction
     MOV     A,ARG2LOW            ;get low byte of block length
     SUBB    A,#1                 ;decrement it by one
     MOV     ARG2LOW,A            ;save result in ARG2LOW
     MOV     A,ARG2HIGH           ;get high byte of block length
     SUBB    A,#0                 ;decrement it if borrow needed for low byte
     MOV     ARG2HIGH,A           ;save result in ARG2HIGH
     ORL     A,ARG2LOW            ;check the count
     JNZ     UPLOAD_EXT1          ;jump if count is not zero
     JMP     NEXT                 ;return to command level

;DNLOAD after confirming correct command syntax downloads an Intel hex file
;into external memory. It checks each data record for correct form and
;terminates when an End of File record (:00000001FF) is received.

DNLOAD:
     CALL    CHK_RET              ;check for a CR at the end of the command
DNLOAD1:
     MOV     CHKSUM,#00           ;clear CHKSUM used to store the checksum
     CALL    SP_IN                ;get 1st character of the hex record
     PUSH    ACC                  ;save it on the stack
```

```
        CALL    SP_OUT              ;echo it to the console
        POP     ACC                 ;get the character off the stack
        CJNE    A,#':',DNLOAD4      ;jump if it is not ':'
        CALL    GET_BYTE            ;get the record length
        JNZ     DNLOAD2             ;jump if record length is not zero
        CALL    GET_BYTE            ;this is last record, see if address is 0000
        JNZ     DNLOAD4             ;jump if high byte of address isn't zero
        CALL    GET_BYTE            ;get low byte of address
        JNZ     DNLOAD4             ;jump if low address byte isn't zero
        CALL    GET_BYTE            ;get the record type
        CJNE    A,#01,DNLOAD4       ;jump if it isn't 01
        CALL    GET_BYTE            ;get the check sum
        CJNE    A,#0FFH,DNLOAD4     ;jump if it isn't FF hex
        CALL    SP_IN               ;check for a CR
        CJNE    A,#CR,DNLOAD4       ;jump if CR is missing
        MOV     A,#BELL             ;signal that download was successful
        CALL    SP_OUT
        MOV     A,#BELL
        CALL    SP_OUT
        MOV     A,#BELL
        CALL    SP_OUT
        JMP     NEXT                ;jump to the command level
DNLOAD2:
        MOV     RECLEN,A            ;save record length in RECLEN
        ADD     A,CHKSUM            ;update the check sum
        MOV     CHKSUM,A
        CALL    GET_BYTE            ;get the high byte of the starting address
        MOV     DPH,A               ;put it in DPH
        ADD     A,CHKSUM            ;update the check sum
        MOV     CHKSUM,A
        CALL    GET_BYTE            ;get the low byte of the starting address
        MOV     DPL,A               ;put it in DPL
        ADD     A,CHKSUM            ;update the check sum
        MOV     CHKSUM,A
        CALL    GET_BYTE            ;get the record type
        JNZ     DNLOAD4             ;abort if the record type isn't 00
DNLOAD3:
        CALL    GET_BYTE            ;get the next data byte of the record
        MOVX    @DPTR,A             ;store it in external memory
        INC     DPTR                ;increment the data pointer
        ADD     A,CHKSUM            ;add the last data byte to the check sum
        MOV     CHKSUM,A            ;save the check sum in CHKSUM
        DJNZ    RECLEN,DNLOAD3      ;continue processing all data bytes
        CALL    GET_BYTE            ;get the check sum byte of the record
        ADD     A,CHKSUM            ;add it to CHKSUM
```

```
     JNZ     DNLOAD5             ;jump if sum of record bytes is not zero
     CALL    SP_IN               ;check for a CR
     CJNE    A,#CR,DNLOAD4       ;jump if there is none
     CALL    CR_LF               ;echo the carriage return (with linefeed)
     JMP     DNLOAD1             ;prepare to receive a new record
DNLOAD4:
     MOV     DPTR,#ERROR6        ;point to proper error message
     CALL    SSTRING             ;send it to the console
     JMP     NEXT                ;return to the command level
DNLOAD5:
     MOV     DPTR,#ERROR7        ;point to proper error message
     CALL    SSTRING             ;send it to the console
     JMP     NEXT                ;return to the command level


;GO does the syntax analysis of the command and if it is good, it takes the
;16 bit address argument, moves it into the data pointer and transfers
;control to the address pointed to by the data pointer

GO:
     CALL    GET_ARG             ;get the high byte of the address
     MOV     ARG1HIGH,A          ;save if in ARG1HIGH
     CALL    GET_ARG             ;get the low byte of the address
     MOV     ARG1LOW,A           ;save it in ARG1LOW
     CALL    CHK_RET             ;check for a CR at the end of the command
     MOV     DPL,ARG1LOW         ;move the jump address into the data pointer
     MOV     DPH,ARG1HIGH
     CLR     A                   ;clear the accumulator
     JNB     TI,$                ;wait for transmissions on the serial port
     JMP     @A +DPTR            ;to end and then jump to the address in the
                                 ;data pointer (since the acc. is clear)

;**********************SERIAL PORT INPUT OUTPUT ROUTINES*******************

;SP_OUT sends the character in ACC to the console. It calculates odd parity
;for the 7-bit ascii character and puts it in the 8th bit of ACC before
;sending ACC out. The routine waits for TI to be asserted before sending
;the character.

SP_OUT:
     MOV     C,P                 ;move the parity into bit C
     CPL     C                   ;complement the bit C for odd parity
     MOV     ACC.7,C             ;append the parity bit to the 7bit ascii
     JNB     TI,$                ;wait for the last transmission to complete
     CLR     TI                  ;clear the transmission interrupt flag
     MOV     SBUF,A              ;send the the data byte down the serial link
     RET                         ;return to the calling routine

;SP_IN waits for a character to be sent from the console. It then moves
;the byte into the accumulator, strips off the parity bit, and returns to
;the caller.

SP_IN:
     JNB     RI,$                ;wait for an input
     CLR     RI                  ;clear the receive interrupt flag
     MOV     A,SBUF              ;move the input byte into the accumulator
     ANL     A,#7FH              ;strip the parity bit off the data byte
     RET                         ;return to the calling routine with the
                                 ;7 bit ascii value of the input in the acc.

;CR_LF sends a carriage return and line feed to the console (PC). The
;accumulator is affected.

CR_LF:
     MOV     A,#CR               ;move carriage return into the acc.
     CALL    SP_OUT              ;send it to the console
     MOV     A,#LF               ;move ascii for line feed into the acc.
     CALL    SP_OUT              ;send it out to the console
     RET                         ;return to the calling routine


;GET_ARG takes two characters from the command buffer, representing a
;hexadecimal data byte, and converts them into a binary value. The result
;is returned in the accumulator.

GET_ARG:
     CALL    GET_CHAR            ;get binary value of next hex char in cmd
     SWAP    A                   ;buffer and put it in upper half of ACC
     MOV     B,A                 ;save the ACC
     CALL    GET_CHAR            ;get binary value of 2nd hex char
     ADD     A,B                 ;add B to it to get the result
     RET                         ;binary value is return in ACC

;GET_CHAR takes a hex character from the command buffer and converts it to
;its binary value. If the conversion isn't possible then the routine
;aborts.

GET_CHAR:
     INC     R0                  ;make R0 point to the next character in the
     MOV     A,@R0               ;command buffer and move it into the acc.
```

```
GET_CHAR1:
     CALL     TO_UPPER           ;convert char to upper case if necessary
     CLR      C                  ;clear carry bit
     SUBB     A,#'0'             ;subtract off the offset of the char '0'
     JC       BAD_CHAR           ;jump if ascii value of the char < '0'
     CJNE     A,#0AH,GET_CHAR2   ;see if the char <= '9'
GET_CHAR2:
     JC       GET_CHAR3          ;jump if it is
     CLR      C                  ;clear carry bit for subtraction
     SUBB     A,#7H              ;compensate for chars between '9' and 'A'
GET_CHAR3:
     CJNE     A,#10H,GET_CHAR4   ;see if the char > 'F'
GET_CHAR4:
     JNC      BAD_CHAR           ;jump if ascii value of the char > 'F'
     RET                         ;return to caller with binary value in ACC

;BAD_CHAR is the abort routine if an invalid hex character is received.

BAD_CHAR:
     MOV      DPTR,#ERROR2       ;get the proper error message
     CALL     SSTRING            ;send it to the console
     JMP      NEXT               ;return to the command level

;GET_BYTE receives two hex characters from the serial link, decodes their
;ascii value and puts them together into one binary byte and returns the
;value in the accumulator.

GET_BYTE:
     CALL     SP_IN              ;get the next character from the serial link
     PUSH     ACC                ;save it on the stack
     CALL     SP_OUT             ;echo it to the console
     POP      ACC                ;restore the character in the ACC
     CALL     GET_CHAR1          ;decode the ascii char into its binary value
     SWAP     A                  ;since this is the higher 4 bit, put them in
     MOV      B,A                ;their right place
     CALL     SP_IN              ;get the 2nd hex char from the serial link
     PUSH     ACC                ;save it on the stack
     CALL     SP_OUT             ;echo it to the console
     POP      ACC                ;restore the character in the acc.
     CALL     GET_CHAR1          ;decode its ascii
     ADD      A,B                ;add the two to get a one byte binary number
     RET                         ;return with the result in ACC

;CHK_RET checks to see if a command in the command buffer was correctly
;terminated by a carriage return, and if so, it returns to the calling
```

```
;program; otherwise it aborts.

CHK_RET:
     INC     R0                  ;point to the next character in the command
     MOV     A,@R0
     CJNE    A,#CR,CHK_RET1      ;jump if it is a carriage return
     RET                         ;return to the calling routine
CHK_RET1:
     MOV     DPTR,#ERROR5        ;point to proper error message
     CALL    SSTRING             ;send it to the console
     JMP     NEXT                ;return to command level

;SEND_BYTE converts the binary number in ACC into its two character hex
;equivalent and sends the resulting bytes to the console.

SEND_BYTE:
     MOV     B,A                 ;save the binary byte in B
     ANL     A,#0F0H             ;extract the top 4 bits
     SWAP    A                   ;move them into the lower half
     CALL    SEND_CHAR           ;send them down the serial link
     MOV     A,B
     ANL     A,#0FH              ;get the lower 4 bits of the initial binary
     CALL    SEND_CHAR           ;byte and again use SEND_CHAR to transmit
     RET                         ;its ascii down the serial link and then
                                 ;return to the calling routine

;SEND_CHAR takes a nibble in the accumulator, converts it into an ascii
;char, and sends it down the serial link to the console.

SEND_CHAR:
     ADD     A,#'0'              ;add on the offset for '0'
     CJNE    A,#':',SEND_CHAR1 ;see if hex char is a letter (A-F)
SEND_CHAR1:
     JC      SEND_CHAR2          ;jump if char is an ascii number (0-9)
     ADD     A,#7                ;compensate for chars between '9' and 'A'
SEND_CHAR2:
     CALL    SP_OUT              ;send the char to the console
     RET


;TO_UPPER converts the ascii character in ACC in the range 'a'-'f' to its
;upper case equivalent.

TO_UPPER:
     CJNE    A,#'a',TU1          ;jump if char not 'a'
```

```
        MOV     A,#'A'              ;convert 'a' to 'A' and return
        RET
TU1:
        CJNE    A,#'b',TU2          ;jump if char not 'b'
        MOV     A,#'B'              ;convert 'b' to 'B' and return
        RET
TU2:
        CJNE    A,#'c',TU3          ;jump if char not 'c'
        MOV     A,#'C'              ;convert 'c' to 'C' and return
        RET
TU3:
        CJNE    A,#'d',TU4          ;jump if char not 'd'
        MOV     A,#'D'              ;convert 'd' to 'D' and return
        RET
TU4:
        CJNE    A,#'e',TU5          ;jump if char not 'e'
        MOV     A,#'E'              ;convert 'e' to 'E' and return
        RET
TU5:
        CJNE    A,#'f',TU6          ;jump if char not 'f'
        MOV     A,#'F'              ;convert 'f' to 'F' and return
TU6:
        RET

;*************ROUTINES USED TO DISPLAY MESSAGES ON CONSOLE***************

;SSTRING sends the string whose starting address is in DPTR over the serial
;link. A NULL character signifies the end of a string.

SSTRING:
        CLR     A                   ;clear the accumulator
        MOVC    A,@A + DPTR         ;get a character
        JZ      SSTRING1            ;jump if finished
        CALL    SP_OUT              ;otherwise, send the character
        INC     DPTR                ;point to the next character
        JMP     SSTRING             ;continue until done
SSTRING1:
        RET                         ;return to caller

;****************************MESSAGES***********************************

RESET_MSG:
        DB      CR,LF,LF,'                    '
        DB      'WELCOME TO THE 8032 MONITOR PROGRAM',CR,LF
        DB      '                                 VER. 1.0',CR,LF
```

```
     DB      NULL           ;end of message marker
HELP_MSG:
     DB      CR,LF,LF,'                                  '
     DB      'COMMAND SUMMARY',CR,LF,LF
     DB      'uaa,cc<CR> ----- UPLOAD INTERNAL RAM sends cc bytes of '
     DB      'memory beginning at',CR,LF,'                address aa in '
     DB      'internal RAM to the console',CR,LF
     DB      'Uaaaa,cccc<CR> - UPLOAD EXTERNAL RAM sends cccc bytes of '
     DB      'memory beginning',CR,LF,'                at address aaaa '
     DB      'in external memory to the console',CR,LF
     DB      'd<CR> ---------- DOWNLOAD EXTERNAL RAM from the console to '
     DB      'internal RAM',CR,LF
     DB      'c<CR> ---------- CLEAR EXTERNAL RAM',CR,LF
     DB      'saa,cc<CR> ----- STORE INTERNAL MEMORY stores the byte cc '
     DB      'at the address',CR,LF,'                aa in internal '
     DB      'memory',CR,LF
     DB      'Saaaa,cc<CR> --- STORE EXTERNAL MEMORY stores the byte cc '
     DB      'at the address',CR,LF,'                aaaa in external '
     DB      'memory',CR,LF
     DB      'gaaaa<CR> ------ GO ADDRESS transfers program execution to '
     DB      'the 2 byte',CR,LF,'                address aaaa',CR,LF
     DB      'h<CR> ---------- HELP displays the command summary',CR,LF,LF
     DB      'Note:  Downloads are expected to be in INTEL HEX FORMAT. '
     DB      'Uploads are just',CR,LF,' strings of hex bytes. All command '
     DB      'arguments are hex numbers.',CR,LF
     DB      NULL           ;end of message marker

ERROR1:      DB     CR,LF,'*ERROR - INVALID COMMAND*',NULL
ERROR2:      DB     CR,LF,'*ERROR - BAD HEX CHARACTER*',NULL
ERROR3:      DB     CR,LF,'*ERROR - INPUT LINE TOO LONG*',NULL
ERROR4:      DB     CR,LF,'*ERROR - COMMA IS MISSING*',NULL
ERROR5:      DB     CR,LF,'*ERROR - CARRIAGE RETURN IS MISSING*',NULL
ERROR6:      DB     CR,LF,'*ERROR - BAD HEX RECORD*',NULL
ERROR7:      DB     CR,LF,'*ERROR - BAD CHECK SUM*',NULL

     END                        ;end of program
```

## C.2 VRM Controller Program

The controller program SPIN.MSA is used to drive the VRM. It allows the user to interactively set the turn-on/off angles and the current chopping level while the motor is running. It is loaded into the 6264 RAM on the controller board using the DOWNLOAD EXTERNAL RAM command of the monitor program. Since the RAM occupies the second 8K section of memory, the code segment should be set to 2000 hex when linking SPIN.MSA.

```
****************************************************************************
*                                                                          *
*                            SPIN.MSA ver. 1.0                             *
*                                                                          *
****************************************************************************

;THE SFRS ASSOCIATED WITH TIMER2 ARE NOT DEFINED IN THE ASSEMBLER. THEY
;HAVE TO BE DEFINED USING EQUATES.

T2CON   EQU     0C8H            ;TIMER2 CONTROL REGISTER
TF2     EQU     T2CON.7         ;TIMER2 OVERFLOW FLAG
EXF2    EQU     T2CON.6         ;TIMER2 EXTERNAL FLAG
TR2     EQU     T2CON.2         ;START/STOP CONTROL FOR TIMER2
RCAP2L  EQU     0CAH            ;TIMER2 CAPTURE REGISTER (LOW BYTE)
RCAP2H  EQU     0CBH            ;TIMER2 CAPTURE REGISTER (HIGH BYTE)
TL2     EQU     0CCH            ;TIMER2 (LOW BYTE)
TH2     EQU     0CDH            ;TIMER2 (HIGH BYTE)
ET2     EQU     IE.5            ;TIMER2 INTERRUPT ENABLE BIT

;PROGRAM CONSTANTS

ISP     EQU     7FH             ;RESET STACK LOCATION
DACADD  EQU     4000H           ;BASE ADDRESS FOR DAC
LATCH   EQU     8000H           ;LS273 PORT ADDRESS
FREQ    EQU     65536-50        ;ROTOR POSITION SAMPLED AT 20KHZ RATE

;PROGRAM VARIABLES

REGEN   EQU     28H             ;VARIABLE FOR THE REGEN SIGNAL
THETA   EQU     29H             ;VARIABLE CONTAINING ROTOR POSITION
CTABLEL EQU     2AH             ;CTABLEH:CTABLEL IS ADDRESS OF CURRENT
```

```
CTABLEH EQU     2BH                 ;TABLE
DAC     EQU     2CH                 ;CURRENT VALUE IN DAC LATCH
ON      EQU     2DH                 ;ON ANGLE
OFF     EQU     2EH                 ;OFF ANGLE
TABLE   EQU     2FH                 ;CONTAINS NUMBER OF TABLE CURRENTLY IN USE
ONANL   EQU     30H                 ;ONANH:ONANL CONTAINS SCALED VALUE OF THE
ONANH   EQU     31H                 ;ON ANGLE
OFFANL  EQU     32H                 ;OFFANH:OFFANL CONTAINS SCALED VALUE OF THE
OFFANH  EQU     33H                 ;OFF ANGLE
UTABLEL EQU     34H                 ;UTABLEH:UTABLEL IS ADDRESS OF UPDATE
UTABLEH EQU     35H                 ;TABLE
INDL    EQU     36H                 ;INDH:INDL IS A TEMPORARY 16-BIT VARIABLE
INDH    EQU     37H

;***************************INTERRUPT VECTOR TABLE***************************

     RSEG    CODE
LOCO:                              ;THIS LOCATION IS SPECIFIED DURING LINKING.

     JMP     INITIAL               ;RESET VECTOR ADDRESS

     ORG     LOCO + 03H            ;EXTERNAL INTERRUPT 0 VECTOR ADDRESS
     RETI

     ORG     LOCO + 0BH            ;TIMER0 VECTOR ADDRESS
     RETI

     ORG     LOCO + 13H            ;EXTERNAL INTERRUPT 1 VECTOR ADDRESS
     RETI

     ORG     LOCO + 1BH            ;TIMER1 VECTOR ADDRESS
     RETI

     ORG     LOCO + 23H            ;SERIAL PORT VECTOR ADDRESS
     RETI

     ORG     LOCO + 2BH            ;TIMER2 VECTOR ADDRESS
     JMP     TIM2_INT

;*************************INITIALIZATION ROUTINE***************************

     ORG     LOCO+30H              ;CODE BEGINS AFTER INTERRUPT VECTORS
INITIAL:
     CLR     A                     ;CLEAR THE ACCUMULATOR
     MOV     IE,A                  ;DISABLE ALL INTERRUPTS
     MOV     DPTR,#LATCH         ;CLEAR THE LATCH
     MOVX    @DPTR,A
     MOV     DPTR,#DACADD        ;GET BASE ADDRESS OF DAC
     MOVX    @DPTR,A             ;SET DAC A OUTPUT VOLTAGE TO ZERO
     INC     DPTR
     MOVX    @DPTR,A             ;SET DAC B OUTPUT VOLTAGE TO ZERO
     INC     DPTR
     MOVX    @DPTR,A             ;SET DAC C OUTPUT VOLTAGE TO ZERO
     INC     DPTR
     MOVX    @DPTR,A             ;SET DAC D OUTPUT VOLTAGE TO ZERO
     MOV     PSW,A               ;SELECT REGISTER BANK 0
     MOV     SP,#ISP             ;INITIALIZE THE STACK POINTER TO 80H
     MOV     R0,#0FFH            ;PREPARE TO CLEAR ALL INTERNAL RAM EXCEPT
INITIAL1:                        ;FOR THE SFRS
     MOV     @R0,A               ;CLEAR INTERNAL RAM POINTED TO BY R0
     DJNZ    R0,INITIAL1         ;CONTINUE UNTIL DONE
     MOV     TCON,#0             ;MAKE SURE TIMERS ARE OFF
     MOV     TMOD,#21H           ;TIMER1-MODE 2, TIMER0-MODE 1, NO GATING
     MOV     TH1,#230            ;LOAD #230 INTO TIMER1 HIGHER BYTE FOR A
                                 ;BAUD RATE OF 1200
     MOV     SCON,#078H          ;MODE 2, 1 START, 8 DATA, AND 1 STOP BIT
     MOV     T2CON,#0H           ;PUT TIMER2 IN AUTO-RELOAD MODE
     MOV     RCAP2H,#HIGH(FREQ) ;INITIALIZE TIMER2 RECAPTURE REGISTER
     MOV     RCAP2L,#LOW(FREQ)
     MOV     REGEN,#0            ;INITIALIZE REGEN VARIABLE
     MOV     ON,#30              ;TURN ON AT MINIMUM INDUCTANCE POINT
     MOV     OFF,#45             ;15 DEGREE CONDUCTION ANGLE
     MOV     TABLE,#1            ;TABLE0 WILL BE SET FIRST
     MOV     DAC,#0              ;INITIALIZE DAC VARIABLE
     MOV     CTABLEL,#LOW(TABLE0) ;INITIALIZE CURRENT TABLE TO TABLE0
     MOV     CTABLEH,#HIGH(TABLE0)
     SETB    TR1                 ;START TIMER1
     SETB    TI                  ;ENABLE SERIAL TRANSMISSION
     CLR     RI                  ;CLEAR THE RECEIVE INTERRUPT FLAG IF SET
     MOV     IE,#0A0H            ;ENABLE TIMER2 INTERRUPTS
     SETB    TR2                 ;START TIMER2
     CALL    CMD_SUMM            ;DISPLAY THE COMMAND SUMMARY
     CALL    SETANGLE            ;DISPLAY ADVANCEMENT ANGLE AND TABLE ADDRESS
     CALL    SHAFT               ;DISPLAY SHAFT ENCODER POSITION

;****************************THE COMMAND LEVEL******************************

START:
     CALL    CR_LF               ;OUTPUT A CR AND LF
     CALL    SPROMPT             ;OUTPUT PROMPT
```

```
START1:
     CALL    SP_INN              ;GET CHARACTER FROM CONSOLE
     JZ      START1              ;JUMP IF NO CHAR WAS INPUT
     CALL    TO_UPPER            ;CONVERT CHAR TO UPPER CASE
CHECKA:
     CJNE    A,#'A',CHECKC
     CALL    SETANGLE            ;SET ON AND OFF ANGLES
     JMP     START
CHECKC:
     CJNE    A,#'C',CHECKD
     CALL    SETCUR              ;SET CURRENT
     JMP     START
CHECKD:
     CJNE    A,#'D',CHECKE
     CALL    DUMPTAB             ;SEND LOOKUP TABLE TO CONSOLE
     JMP     START
CHECKE:
     CJNE    A,#'E',CHECKH
     JMP     EXIT                ;EXIT
CHECKH:
     CJNE    A,#'H',CHECKR
     CALL    CMD_SUMM            ;COMMAND SUMMARY
     JMP     START
CHECKR:
     CJNE    A,#'R',CHECKS
     CALL    TOGREGEN            ;TOGGLE REGEN SIGNAL
     JMP     START
CHECKS:
     CJNE    A,#'S',BADCMD
     CALL    SHAFT               ;DISPLAY SHAFT POSITION
     JMP     START
BADCMD:
     MOV     DPTR,#ERR_MSG       ;POINT TO ERROR MESSAGE
     CALL    SSTRING             ;SEND IT TO THE CONSOLE
     JMP     START

CMD_SUMM:
     MOV     DPTR,#CMD_MSG
     CALL    SSTRING             ;DISPLAY THE COMMAND SUMMARY
     RET



SETCUR:
     MOV     DPTR,#C_MSG         ;DISPLAY PRESENT CURRENT CHOPPING LEVEL
     CALL    SSTRING
```

```
        MOV     A,DAC               ;R2:R1:R0 = DAC * 100
        MOV     B,#100
        MUL     AB
        MOV     R0,A
        MOV     R1,B
        MOV     R2,#0
        MOV     R3,#135             ;R4:R3 = 135
        MOV     R4,#0
        CALL    UDIV                ;R0 = DAC * 100 / 135
        CALL    SENDNUM             ;OUTPUT RESULT TO CONSOLE
        CALL    CR_LF               ;OUTPUT A CR AND LF
SETCUR1:
        MOV     DPTR,#CUR_MSG       ;GET POINTER TO CURRENT MESSAGE
        CALL    SSTRING             ;DISPLAY PROMPT MESSAGE
        CALL    INPUT               ;GET NEW CURRENT VALUE FROM CONSOLE
        CALL    GETNUM              ;TRANSLATE IT INTO BINARY FORM
        MOV     A,R1                ;GET HIGH BYTE OF CURRENT LEVEL
        JNZ     SETCUR1             ;JUMP IF CURRENT LEVEL IS TOO HIGH
        MOV     A,R0                ;GET LOW BYTE OF CURRENT LEVEL
        CJNE    A,#101,SETCUR2      ;JUMP IF CURRENT LEVEL ISN'T 101
SETCUR2:
        JNC     SETCUR1             ;JUMP IF CURRENT LEVEL IS TOO HIGH
        MOV     B,#135              ;B:A = 135 * CURRENT
        MUL     AB
        MOV     R0,A                ;MOV B:A INTO 24-BIT DIVIDEND R2:R1:R0
        MOV     R1,B
        MOV     R2,#0
        MOV     R3,#100             ;MOVE 100 INTO 16-BIT DIVISOR R4:R3
        MOV     R4,#0
        CALL    UDIV                ;R1:R0 = 135 * CURRENT / 100
        MOV     A,R0                ;GET THE RESULT
        MOV     DAC,A               ;SAVE IT
        MOV     DPTR,#DACADD        ;GET BASE ADDRESS OF DAC
        MOVX    @DPTR,A             ;SET DAC0 AND DAC1 TO THE NEW VALUE
        INC     DPTR
        MOVX    @DPTR,A
        RET

EXIT:
        MOV     IE,#0               ;DISABLE ALL INTERRUPTS
        MOV     DPTR,#LATCH         ;GET THE ADDRESS OF THE LATCH
        MOV     A,#80H              ;TURN OFF ALL PHASES AND SIGNAL END OF RUN
        MOVX    @DPTR,A
        CLR     A                   ;CLEAR ACC
        MOV     DAC,A
```

```
     MOV     DPTR,#DACADD        ;GET BASE ADDRESS OF DAC
     MOVX    @DPTR,A             ;SET DAC A OUTPUT VOLTAGE TO ZERO
     INC     DPTR
     MOVX    @DPTR,A             ;SET DAC B OUTPUT VOLTAGE TO ZERO
     MOV     RO,A                ;CLEAR RO
     DJNZ    RO,$                ;DELAY LOOP
     MOVX    @DPTR,A             ;CLEAR END OF RUN SIGNAL
     MOV     DPTR,#0             ;CLEAR DPTR
     JMP     @A+DPTR             ;JUMP TO MONITOR PROGRAM

TOGREGEN:
     XRL     REGEN,#0F0H         ;TOGGLE THE REGEN SIGNALS
     MOV     A,REGEN             ;GET THE REGEN VARIABLE
     JZ      TOGREGEN1           ;JUMP IF REGEN SIGNAL IS OFF
     MOV     DPTR,#REGENON       ;SIGNAL USER THAT REGEN SIGNAL IS ON
     CALL    SSTRING
     RET
TOGREGEN1:
     MOV     DPTR,#REGENOFF      ;SIGNAL USER THAT REGEN SIGNAL IS OFF
     CALL    SSTRING
     RET

SETANGLE:
     MOV     DPTR,#ON_MSG        ;DISPLAY ON ANGLE
     CALL    SSTRING
     MOV     RO,ON               ;R1:RO = ON ANGLE
     MOV     R1,#0
     CALL    SENDNUM             ;SEND THE ANGLE
     CALL    CR_LF
     MOV     DPTR,#AN_MSG        ;DISPLAY ANGLE PROMPT
     CALL    SSTRING
     CALL    INPUT               ;GET THE NEW ANGLE
     CALL    GETNUM              ;TRANSLATE IT TO BINARY FORM
     MOV     A,RO                ;GET NEW ANGLE
     CJNE    A,#61,SETANGLE1     ;TEST IT
SETANGLE1:
     JNC     SETANGLE            ;JUMP IF ANGLE IS TOO LARGE
     MOV     ON,A                ;SAVE IT
     CALL    FIXAN
     MOV     ONANL,RO            ;ONANH:ONANL = 100 * ON / 9
     MOV     ONANH,R1
SETANGLE2:
     MOV     DPTR,#OFF_MSG       ;DISPLAY OFF ANGLE
     CALL    SSTRING
     MOV     RO,OFF              ;R1:RO = OFF ANGLE
```

```
        MOV     R1,#0
        CALL    SENDNUM             ;SEND THE ANGLE
        CALL    CR_LF
        MOV     DPTR,#AN_MSG        ;DISPLAY ANGLE PROMPT
        CALL    SSTRING
        CALL    INPUT               ;GET THE NEW ANGLE
        CALL    GETNUM              ;TRANSLATE IT TO BINARY FORM
        MOV     A,R0                ;GET NEW ANGLE
        CJNE    A,#61,SETANGLE3     ;TEST IT
SETANGLE3:
        JNC     SETANGLE2           ;JUMP IF ANGLE IS TOO LARGE
        MOV     OFF,A               ;SAVE IT
        CLR     C                   ;OFF - ON
        SUBB    A,ON
        JNC     SETANGLE6           ;JUMP IF OFF >= ON
        MOV     A,OFF               ;ADD 60 TO OFF ANGLE
        ADD     A,#60
        MOV     OFF,A
SETANGLE6:
        MOV     A,OFF               ;GET THE OFF ANGLE
        CALL    FIXAN
        MOV     OFFANL,R0           ;OFFANH:OFFANL = 100 * OFF / 9
        MOV     OFFANH,R1
        MOV     A,TABLE
        JNZ     SETANGLE4
        MOV     TABLE,#1            ;TABLE 1 WILL BE UPDATED
        MOV     UTABLEL,#LOW(TABLE1)
        MOV     UTABLEH,#HIGH(TABLE1)
        MOV     DPTR,#TABLE1        ;CLEAR TABLE1
        CALL    CLRTAB
        JMP     SETANGLE5
SETANGLE4:
        MOV     TABLE,#0            ;TABLE 0 WILL BE UPDATED
        MOV     UTABLEL,#LOW(TABLE0)
        MOV     UTABLEH,#HIGH(TABLE0)
        MOV     DPTR,#TABLE0        ;CLEAR TABLE0
        CALL    CLRTAB
SETANGLE5:
        MOV     R3,#1               ;TABLE DATA FOR PHASE A
        MOV     DPTR,#MAXA
        CALL    SETT
        MOV     R3,#2               ;TABLE DATA FOR PHASE B
        MOV     DPTR,#MAXB
        CALL    SETT
        MOV     R3,#4               ;TABLE DATA FOR PHASE C
```

```
          MOV      DPTR,#MAXC
          CALL     SETT
          MOV      R3,#8              ;TABLE DATA FOR PHASE D
          MOV      DPTR,#MAXD
          CALL     SETT
          CLR      EA                 ;DISABLE INTERRUPTS
          MOV      DPTR,#LATCH        ;GET THE ADDRESS OF THE LATCH
          CLR      A
          MOVX     @DPTR,A            ;TURN OFF ALL PHASES
          MOV      CTABLEH,UTABLEH    ;CTABLEH:CTABLEL = UTABLEH:UTABLEL
          MOV      CTABLEL,UTABLEL
          SETB     EA                 ;ENABLE INTERRUPTS
          RET

     FIXAN:
          MOV      B,#100             ;SCALED ANGLE = 100 * ANGLE / 9
          MUL      AB
          MOV      R0,A
          MOV      R1,B
          MOV      R2,#0
          MOV      R3,#9
          MOV      R4,#0
          CALL     UDIV
          MOV      A,R5
          CJNE     A,#5,FIXAN1
     FIXAN1:
          JC       FIXAN2             ;JUMP IF REMAINDER < 5
          MOV      A,R0               ;ADD 1 TO QUOTIENT
          ADD      A,#1
          MOV      R0,A
          MOV      A,R1
          ADDC     A,#0
          MOV      R1,A
     FIXAN2:
          RET

     CLRTAB:
          CLR      A                  ;CLEAR TABLE POINTED TO BY DPTR
          MOV      R0,A
     CLRTAB1:
          MOVX     @DPTR,A
          INC      DPTR
          DJNZ     R0,CLRTAB1
          RET
```

```
SETT:
     MOV     R2,#6              ;R2 IS A COUNTER
SETT1:
     MOVX    A,@DPTR            ;GET MAX INDUCTANCE POSITION
     MOV     INDH,A
     INC     DPTR
     MOVX    A,@DPTR
     MOV     INDL,A
     INC     DPTR
     MOV     A,ONANL            ;ADD IT TO ONANH:ONANL
     ADD     A,INDL
     MOV     R4,A
     MOV     A,ONANH
     ADDC    A,INDH
     MOV     R5,A
     MOV     A,R4               ;DIVIDE RESULT BY 16, ROUND UP IF REMAINDER
     ANL     A,#0F0H            ;IS GREATER THAN 7
     SWAP    A
     MOV     B,A
     MOV     A,R5
     SWAP    A
     MOV     R7,A
     ANL     A,#0F0H
     ADD     A,B
     MOV     R5,A
     MOV     A,R7
     ANL     A,#0FH
     MOV     R6,A
     MOV     A,R4               ;CHECK THE REMAINDER
     ANL     A,#0FH
     CLR     C
     SUBB    A,#8
     JC      SETT9
     MOV     A,R5
     ADD     A,#1
     MOV     R5,A
     MOV     A,R6
     ADDC    A,#0
     MOV     R6,A
SETT9:
     CLR     C                  ;RESULT MODULO 250
     MOV     A,R5
     SUBB    A,#250
     MOV     R0,A
```

```
        MOV     A,R6
        SUBB    A,#0
        JNC     SETT2
        MOV     A,R5
        MOV     R0,A
   SETT2:
        MOV     A,OFFANL            ;ADD IT TO OFFANH:OFFANL
        ADD     A,INDL
        MOV     R4,A
        MOV     A,OFFANH
        ADDC    A,INDH
        MOV     R5,A
        MOV     A,R4                ;DIVIDE RESULT BY 16, ROUND UP IF REMAINDER
        ANL     A,#0F0H             ;IS GREATER THAN 7
        SWAP    A
        MOV     B,A
        MOV     A,R5
        SWAP    A
        MOV     R7,A
        ANL     A,#0F0H
        ADD     A,B
        MOV     R5,A
        MOV     A,R7
        ANL     A,#0FH
        MOV     R6,A
        MOV     A,R4                ;CHECK THE REMAINDER
        ANL     A,#0FH
        CLR     C
        SUBB    A,#8
        JC      SETT10
        MOV     A,R5
        ADD     A,#1
        MOV     R5,A
        MOV     A,R6
        ADDC    A,#0
        MOV     R6,A
   SETT10:
        CLR     C                   ;RESULT MODULO 250
        MOV     A,R5
        SUBB    A,#250
        MOV     R1,A
        MOV     A,R6
        SUBB    A,#0
        JNC     SETT3
        MOV     A,R5
```

```
     MOV     R1,A
SETT3:
     MOV     A,R1                ;OFF ANGLE - ON ANGLE
     CLR     C
     SUBB    A,R0
     PUSH    DPL                 ;SAVE DPTR
     PUSH    DPH
     JC      SETT5               ;JUMP IF ON ANGLE > OFF ANGLE
     INC     A                   ;# TABLE POSITIONS = OFF - ON + 1
     MOV     R1,A
     MOV     A,UTABLEL           ;POINT TO R0TH LOCATION IN TABLE
     ADD     A,R0
     MOV     DPL,A
     MOV     A,UTABLEH
     ADDC    A,#0
     MOV     DPH,A
SETT4:
     MOVX    A,@DPTR             ;GET CURRENT TABLE VALUE
     ADD     A,R3                ;ADD R3 TO IT
     MOVX    @DPTR,A             ;PLACE RESULT BACK IN TABLE
     INC     DPTR
     DJNZ    R1,SETT4
     POP     DPH
     POP     DPL
     DJNZ    R2,SETT8            ;REPEAT FOR OTHER INDUCTANCE POINTS
     RET
SETT5:
     MOV     DPL,UTABLEL
     MOV     DPH,UTABLEH
     INC     R1                  ;PROCESS POSITIONS 0 THROUGH OFF + 1
SETT6:
     MOVX    A,@DPTR             ;GET CURRENT TABLE VALUE
     ADD     A,R3                ;ADD R3 TO IT
     MOVX    @DPTR,A             ;PLACE RESULT BACK IN TABLE
     INC     DPTR
     DJNZ    R1,SETT6
     MOV     A,UTABLEL           ;POINT TO R0TH LOCATION IN TABLE
     ADD     A,R0
     MOV     DPL,A
     MOV     A,UTABLEH
     ADDC    A,#0
     MOV     DPH,A
     MOV     A,#250              ;# LOCATIONS = 250 - R0
     CLR     C
     SUBB    A,R0
```

```
        MOV     R0,A
SETT7:
        MOVX    A,@DPTR             ;GET CURRENT TABLE VALUE
        ADD     A,R3                ;ADD R3 TO IT
        MOVX    @DPTR,A             ;PLACE RESULT BACK IN TABLE
        INC     DPTR
        DJNZ    R0,SETT7
        POP     DPH
        POP     DPL
        DJNZ    R2,SETT8            ;REPEAT FOR OTHER INDUCTANCE POINTS
        RET
SETT8:
        JMP     SETT1

SHAFT:
        MOV     DPTR,#SHAFT_MSG     ;DISPLAY SHAFT ENCODER MESSAGE
        CALL    SSTRING
        MOV     R0,THETA            ;SEND THE SHAFT POSITION
        MOV     R1,#0
        CALL    SENDNUM             ;SEND IT TO THE CONSOLE
        CALL    CR_LF               ;OUTPUT A CR AND LF
        RET

DUMPTAB:
        MOV     DPL,UTABLEL         ;GET ADDRESS OF CURRENT TABLE
        MOV     DPH,UTABLEH
        MOV     INDL,#250
DUMPTAB1:
        MOVX    A,@DPTR             ;GET NEXT TABLE VALUE
        INC     DPTR
        PUSH    DPL                 ;SAVE DPTR
        PUSH    DPH
        MOV     R0,A
        MOV     R1,#0
        CALL    SENDNUM             ;SEND VALUE TO THE CONSOLE
        CALL    CR_LF
        POP     DPH
        POP     DPL
        DJNZ    INDL,DUMPTAB1
        CALL    SP_IN               ;WAIT FOR CHAR
        RET

;TIMER2 INTERRUPT ROUTINE.

TIM2_INT:
      PUSH    PSW
      PUSH    ACC
      PUSH    DPL
      PUSH    DPH
TIM2_INT1:
      CLR     P3.5              ;START THE SHAFT ENCODER READING PROCESS
      MOV     DPH,CTABLEH       ;GET THE BASE ADDRESS OF THE LOOKUP TABLE
      MOV     DPL,CTABLEL
      MOV     A,P1              ;READ IN THE ROTOR POSITION
      SETB    P3.5              ;END SHAFT ENCODER READING PROCESS
      MOV     THETA,A           ;SAVE THE ROTOR POSITION
      MOVC    A,@A+DPTR         ;USE THE LOOKUP TABLE TO GET PROPER PHASE
      ORL     A,REGEN           ;SET THE REGEN SIGNALS IF NECESSARY
      MOV     DPTR,#LATCH       ;WRITE THE RESULT TO THE LATCH
      MOVX    @DPTR,A
      POP     DPH
      POP     DPL
      POP     ACC
      POP     PSW
      RETI

;THE LOOKUP TABLES TABLE0 AND TABLE1 ARE SET BY THE COMMAND SETANGLE. ONE
;TABLE IS THE CURRENT TABLE AND THE OTHER IS THE UPDATE TABLE. THE CURRENT
;TABLE IS THE ONE THAT IS USED BY THE TIMER2 INTERRUPT ROUTINE TO TURN THE
;PHASES ON AND OFF. WHEN THE SETANGLE COMMAND IS EXECUTED, THE UPDATE TABLE
;IS FILLED ACCORDING TO THE NEW TURN-ON/OFF ANGLES. AFTER IT IS FILLED, THE
;ROLES OF THE CURRENT AND UPDATE TABLES ARE SWAPPED

TABLE0:
      DS      256
TABLE1:
      DS      256

;THE FOLLOWING TABLES CONTAIN THE ALIGNED POSITIONS FOR THE FOUR PHASES.
;THESE TABLES ARE USED BY THE SETANGLE COMMAND TO GENERATE THE LOOKUP
;TABLES. THESE TABLES HAVE TO BE UPDATED WHENEVER THE SHAFT ENCODER IS
;REMOVED FROM THE VRM.

MAXA:
      DW      176,843,1509,2176,2843,3509      ;PHASE A ALIGNED POSITIONS
MAXB:
      DW      343,1009,1676,2343,3009,3676     ;PHASE B ALIGNED POSITIONS
MAXC:
      DW      509,1176,1843,2509,3176,3843     ;PHASE C ALIGNED POSITIONS
MAXD:
     DW      9,676,1343,2009,2676,3343       ;PHASE D ALIGNED POSITIONS

;THE FOLLOWING LINE INCLUDES THE FILE UTIL.MSA WHICH CONTAINS THE MATH AND
;I/O ROUTINES USED IN THIS PROGRAM

$UTIL.MSA

;*****************************MESSAGES*************************************

CMD_MSG:
     DB      CR,LF,LF,'COMMAND SUMMARY',CR,LF,LF
     DB      'A - SET THE ON AND OFF ANGLES',CR,LF,LF
     DB      'C - SET THE CURRENT CHOPPING LEVEL',CR,LF,LF
     DB      'D - DUMP LOOKUP TABLE',CR,LF,LF
     DB      'E - JUMP TO MONITOR PROGRAM',CR,LF,LF
     DB      'H - DISPLAY THE COMMAND SUMMARY',CR,LF,LF
     DB      'R - TOGGLE THE REGEN SIGNAL',CR,LF,LF
     DB      'S - GET SHAFT POSITION',CR,LF,LF
     DB      NULL              ;END OF MESSAGE MARKER

AN_MSG:
     DB      'ENTER NEW ANGLE BETWEEN 0 AND 60 DEGREES: ',NULL

C_MSG:
     DB      'CHOPPING CURRENT IN TENTHS OF AN AMP IS: ',NULL

CUR_MSG:
     DB      'ENTER NEW CURRENT IN TENTHS OF AN AMP (0-100): ',NULL

ERR_MSG:
     DB      '*ERROR - BAD COMMAND*',CR,LF,NULL

OFF_MSG:
     DB      'OFF ANGLE IS: ',NULL

ON_MSG:
     DB      'ON ANGLE IS: ',NULL

REGENOFF:
     DB      'REGEN SIGNAL IS OFF',CR,LF,NULL

REGENON:
     DB      'REGEN SIGNAL IS ON',CR,LF,NULL

SHAFT_MSG:
```

```
DB        'SHAFT POSITION IS: ',NULL

END                              ;END OF PROGRAM
```

```
***************************************************************************
*                                                                         *
*                           UTIL.MSA ver. 1.0                             *
*                                                                         *
***************************************************************************

;ascii constants

NULL    EQU     00H             ;ascii for control char NULL
BELL    EQU     07H             ;ascii for bell
BS      EQU     08H             ;ascii for back space in hex
LF      EQU     0AH             ;ascii for line feed in hex
CR      EQU     0DH             ;ascii for carriage return in hex
PROMPT  EQU     3EH             ;ascii for the prompt '>' in hex
DEL     EQU     7FH             ;ascii for delete in hex

;program constants

BUFFER  EQU     60H             ;command buffer address in internal RAM
ENDBUF  EQU     70H             ;end of command buffer, it is 16 chars long

;program variables

MATHL   EQU     20H             ;MATHH:MATHL is a 16-bit scratch variable
MATHH   EQU     21H             ;for math calculations
FLAGS   EQU     22H             ;8-bit scratch variable
POINTER EQU     23H             ;pointer into command buffer
TEMP    EQU     24H             ;8-bit scratch variable

;UDIV divides the 24-bit unsigned number in registers R2:R1:R0 (R2 is most
;significant byte) by the 16-bit unsigned number in R4:R3 (R4 is MSB) using
;a subtract and shift algorithm. Registers R6:R5 form the remainder word,
;while R7 is used as a counter. The 16-bit result is placed in the R1:R0
;word.

UDIV:
     MOV     R5,#0              ;clear the remainder word
     MOV     R6,#0
     MOV     R7,#24             ;24-bit divide operation
     CLR     C                  ;clear carry bit
UDIV1:
     MOV     A,R0               ;shift the 24-bit dividend R2:R1:R0 left by
     RLC     A                  ;1 bit. The least significant bit of R0 will
```

```
        MOV     R0,A            ;receive the carry bit, while the bit
        MOV     A,R1            ;shifted out of R2 will go into the carry
        RLC     A               ;bit.
        MOV     R1,A
        MOV     A,R2
        RLC     A
        MOV     R2,A
        MOV     A,R5            ;now shift remainder word left by 1 bit,
        RLC     A               ;putting the carry bit into the least
        MOV     R5,A            ;significant bit of R5
        MOV     A,R6
        RLC     A
        MOV     R6,A
        MOV     FLAGS.0,C       ;save the carry bit
        MOV     MATHH,R6        ;save the remainder in MATHH:MATHL word
        MOV     MATHL,R5
        CLR     C               ;subtract the divisor from the remainder and
        MOV     A,R5            ;leave the result in the remainder
        SUBB    A,R3
        MOV     R5,A
        MOV     A,R6
        SUBB    A,R4
        MOV     R6,A
        JNC     UDIV2           ;jump if carry bit is clear
        JNB     FLAGS.0,UDIV2   ;jump if high bit of remainder is clear
        CPL     C               ;negate carry bit
UDIV2:
        CPL     C               ;C will now be set if remainder >= divisor
        JC      UDIV3           ;jump if subtraction successful
        MOV     R5,MATHL        ;otherwise, restore remainder to its
        MOV     R6,MATHH        ;original value before the subtraction
UDIV3:
        DJNZ    R7,UDIV1        ;repeat for all 24 bits of dividend
        MOV     A,R0            ;shift dividend 1 more time. It will then
        RLC     A               ;contain the quotient.
        MOV     R0,A            ;put the quotient in R1:R0
        MOV     A,R1
        RLC     A
        MOV     R1,A
        RET

;SP_OUT is passed an argument (an ascii value) in the accumulator. It
;calculates odd parity for it and puts it in the 8th bit of the
;accumulator. The new data byte (7 bit ascii with the 8th bit as the odd
;parity bit) is then transmitted over the serial link. The routine waits
```

```
;until the transmit buffer is empty before sending the character.

SP_OUT:
     MOV     C,P                  ;move the parity into bit C
     CPL     C                    ;complement the bit C for odd parity
     MOV     ACC.7,C              ;append the parity bit to the 7bit ascii
     JNB     TI,$                 ;wait for the last transmission to complete
     CLR     TI                   ;clear the transmission interrupt flag
     MOV     SBUF,A               ;send the the data byte down the serial link
     RET                          ;return to the calling routine

;SP_IN waits for a character to be sent from the console. It then moves
;the byte into the accumulator, strips off the parity bit, and returns to
;the caller.

SP_IN:
     JNB     RI,$                 ;wait for an input
     CLR     RI                   ;clear the receive interrupt flag
     MOV     A,SBUF               ;move the input byte into the accumulator
     ANL     A,#7FH               ;strip the parity bit off the data byte
     RET                          ;return to the calling routine with the
                                  ;7 bit ascii value of the input in the acc.

;SP_INN checks to see if char is in input buffer. If so, char is processed
;as in SP_IN routine. Otherwise, a NULL char is returned.

SP_INN:
     JB      RI,SP_INN1           ;continue if char is available
     CLR     A                    ;clear ACC
     RET
SP_INN1:
     CLR     RI                   ;clear the receive interrupt flag
     MOV     A,SBUF               ;move the input byte into the accumulator
     ANL     A,#7FH               ;strip the parity bit off the data byte
     RET                          ;return to the calling routine with the
                                  ;7 bit ascii value of the input in the acc.


;CR_LF sends a carriage return and line feed to the console (PC). The
;accumulator is affected.

CR_LF:
     MOV     A,#CR                ;move carriage return into the acc.
     CALL    SP_OUT               ;send it to the console
     MOV     A,#LF                ;move ascii for line feed into the acc.
     CALL    SP_OUT              ;send it out to the console
     RET                         ;return to the calling routine

;SEND_BYTE takes a binary number in the accumulator and extracts its two
;hex parts the top 4 bits and the lower 4 bits. It then encodes the hex
;bytes into ascii by looking up the conversion table and then transmits
;the resulting ascii values down the serial link

SEND_BYTE:
     MOV     B,A                 ;save the binary byte in B
     ANL     A,#0F0H             ;extract the top 4 bits
     SWAP    A                   ;move them into the lower half
     CALL    SEND_CHAR           ;use SEND_CHAR to send the ascii for the hex
     MOV     A,B                 ;character in acc. down the serial link
     ANL     A,#0FH              ;get the lower 4 bits of the initial binary
     CALL    SEND_CHAR           ;byte and again use SEND_CHAR to transmit
     RET                         ;its ascii down the serial link and then
                                 ;return to the calling routine

;SEND_CHAR takes a nibble in the accumulator, converts it into an ascii
;character, and sends it down the serial link to the console.

SEND_CHAR:
     ADD     A,#'0'              ;add on the offset for '0'
     CJNE    A,#':',SEND_CHAR1 ;see if hex char is a letter (A-F)
SEND_CHAR1:
     JC      SEND_CHAR2          ;jump if char is an ascii number (0-9)
     ADD     A,#7                ;compensate for chars between '9' and 'A'
SEND_CHAR2:
     CALL    SP_OUT              ;send the char to the console
     RET

;TO_UPPER converts the ascii character in ACC in the range 'a'-'f' to its
;upper case equivalent.

TO_UPPER:
     CJNE    A,#123,TU1
TU1:
     JNC     TU3                 ;JUMP IF CHAR NOT LOWER CASE LETTER
     CJNE    A,#97,TU2
TU2:
     JC      TU3                 ;JUMP IF CHAR NOT LOWER CASE LETTER
     CLR     C
     SUBB    A,#32               ;CONVERT CHAR TO UPPER CASE
TU3:
     RET

;TO_HEX converts an ascii char to hex digit.

TO_HEX:
     CALL    TO_UPPER          ;convert char to upper case if necessary
     CLR     C
     SUBB    A,#'0'            ;subtract off the offset of char '0'
     CJNE    A,#0AH,TO_HEX1    ;see if char <= '9'
TO_HEX1:
     JC      TO_HEX3           ;jump if it is
     CLR     C
     SUBB    A,#7              ;compensate for chars between '9' and 'A'
     CJNE    A,#16,TO_HEX2     ;SEE IF VALUE IS <= 15
TO_HEX2:
     JC      TO_HEX3           ;JUMP IF IT IS
     CLR     A                 ;INVALID HEX DIGITS ARE SET TO ZERO
TO_HEX3:
     RET

;TO_DEC converts the ascii character in ACC to a decimal digit.

TO_DEC:
     CALL    TO_UPPER          ;convert char to upper case if necessary
     CLR     C
     SUBB    A,#'0'            ;subtract off the offset of char '0'
     CJNE    A,#0AH,TO_DEC1    ;see if char is valid decimal digit
TO_DEC1:
     JNC     TO_DEC2           ;jump if it isn't
     RET
TO_DEC2:
     MOV     DPTR,#ERROR1      ;get pointer to error message
     CALL    SSTRING           ;send it
     MOV     SP,#ISP           ;reinitialize stack pointer
     JMP     START             ;return to command level

;INPUT reads a line of input from the console and stores it in the command
;buffer. If one or more characters were input, then ACC will contain the
;1st character typed, otherwise it will contain NULL. A DELETE operation
;can be performed by using either the delete or backspace keys.

INPUT:
     MOV     POINTER,#BUFFER   ;reset the buffer pointer
INPUT1:
     CALL    SP_IN             ;get a char
     CJNE    A,#CR,INPUT3       ;jump if char not CR
     CALL    CR_LF              ;echo the CR
     MOV     A,POINTER          ;get the value of th
     CJNE    A,#BUFFER,INPUT2   ;jump if buffer not
     MOV     A,#NULL            ;signal that buffe
     RET
INPUT2:
     MOV     A,BUFFER           ;get the 1st char in buffer
     RET
INPUT3:
     CJNE    A,#BS,INPUT6       ;jump if char not BS
INPUT4:
     MOV     A,POINTER          ;get buffer pointer
     CJNE    A,#BUFFER,INPUT5   ;jump if buffer not empty
     JMP     INPUT              ;buffer empty, ignore BS command
INPUT5:
     DEC     POINTER            ;remove last char from buffer
     MOV     A,#BS              ;perform delete by sending a BS, ' ',
     CALL    SP_OUT             ;and another BS to the console
     MOV     A,#' '
     CALL    SP_OUT
     MOV     A,#BS
     CALL    SP_OUT
     JMP     INPUT1             ;continue receiving characters
INPUT6:
     CJNE    A,#DEL,INPUT7      ;jump if char not DEL
     JMP     INPUT4             ;perform delete operation
INPUT7:
     MOV     RO,POINTER         ;get buffer pointer
     MOV     @RO,A              ;save char in buffer
     INC     POINTER            ;increment buffer pointer
     CALL    SP_OUT             ;echo the char to the console
     MOV     A,POINTER          ;get buffer pointer
     CJNE    A,#ENDBUF,INPUT1   ;continue if buffer not full
     CALL    CR_LF              ;buffer full, output CR and LF
     MOV     A,BUFFER           ;get 1st char in buffer
     RET

;GETWORD takes 4 ascii hex chars from the buffer, converts them into their
;16-bit binary equivalent, and stores the result in R1:RO

GETWORD:
     MOV     RO,#BUFFER         ;point to beginning of buffer
     CALL    GETBYTE            ;get high byte of word
     MOV     R1,A               ;save it in R1
     CALL    GETBYTE             ;get low byte of word
     MOV     R0,A                ;save it in R0
     RET

GETBYTE:
     MOV     A,@R0               ;get the 1st char
     CALL    TO_HEX              ;convert it to a hex digit
     SWAP    A                   ;put it in upper nibble of ACC
     MOV     B,A                 ;save ACC
     INC     R0                  ;point to next char
     MOV     A,@R0               ;get it
     CALL    TO_HEX              ;convert it
     ADD     A,B                 ;combine it with 1st digit
     INC     R0                  ;point to next char
     RET

;GETNUM converts the decimal number in the command buffer to it binary
;equivalent. If the number is greater than 65535, then the routine aborts.

GETNUM:
     MOV     MATHL,#0            ;initialize MATHH:MATHL word to zero
     MOV     MATHH,#0
     MOV     R0,#(BUFFER-1)      ;point to beginning of buffer
GETNUM1:
     INC     R0                  ;increment buffer pointer
     MOV     A,R0                ;put it in ACC
     CJNE    A,POINTER,GETNUM2 ;jump if there are more characters
     MOV     R0,MATHL            ;PUT RESULT IN R1:R0
     MOV     R1,MATHH
     RET
GETNUM2:
     MOV     TEMP,MATHH          ;save MATHH
     MOV     A,MATHL             ;get the low byte of the answer
     MOV     B,#10
     MUL     AB                  ;multiply it by 10
     MOV     MATHH,B             ;put 16 bit result in MATHH:MATHL
     MOV     MATHL,A
     MOV     A,TEMP              ;get high byte of the answer
     MOV     B,#10
     MUL     AB                  ;multiply it by 10
     ADD     A,MATHH             ;add low byte of 16 bit product to MATHH
     MOV     MATHH,A             ;put sum in MATHH
     JC      GETNUM3             ;jump if the number is too large
     MOV     A,B                 ;get high byte of result
     JNZ     GETNUM3             ;jump if the number is too large
```

```
        MOV     A,@R0                   ;get the next char
        CALL    TO_DEC                  ;convert it to a decimal digit
        ADD     A,MATHL                 ;add it to MATHH:MATHL word
        MOV     MATHL,A
        CLR     A
        ADDC    A,MATHH
        MOV     MATHH,A
        JNC     GETNUM1                 ;continue if number isn't too large
GETNUM3:
        MOV     DPTR,#ERROR2            ;point to error message
        CALL    SSTRING                 ;send it to the console
        MOV     SP,#ISP                 ;reinitialize stack pointer
        JMP     START

;SENDNUM converts the 16-bit binary number in R1:R0 to its decimal
;equivalent and sends it down the serial link.

SENDNUM:
        MOV     R2,#0                   ;clear R2
        MOV     TEMP,#5                 ;there are 5 decimal digits
        MOV     R3,#10                  ;initialize divisor R4:R3 to '10'
        MOV     R4,#0
SENDNUM1:
        CALL    UDIV                    ;divide the number by ten
        MOV     A,R5                    ;get the remainder
        PUSH    ACC                     ;save it
        DJNZ    TEMP,SENDNUM1           ;get all 5 digits
        MOV     TEMP,#4
SENDNUM2:                               ;remove leading zeros from number
        POP     ACC                     ;get the next digit
        JNZ     SENDNUM4                ;jump if it is not zero
        DJNZ    TEMP,SENDNUM2
        POP     ACC
SENDNUM3:
        CALL    SEND_CHAR               ;send LSD to console
        RET
SENDNUM4:
        CALL    SEND_CHAR               ;send the digit to the console
        POP     ACC                     ;get the next digit
        DJNZ    TEMP,SENDNUM4
        JMP     SENDNUM3

;SSTRING sends the string whose starting address is in DPTR over the
;serial link. A NULL character signifies the end of a string.
```

```
SSTRING:
     CLR     A                  ;clear the accumulator
     MOVC    A,@A + DPTR        ;get a character
     JZ      SSTRING1           ;jump if finished
     CALL    SP_OUT             ;otherwise, send the character
     INC     DPTR               ;point to the next character
     JMP     SSTRING            ;continue until done
SSTRING1:
     RET                        ;return to caller

;SPROMPT sends a prompt to the console.

SPROMPT:
     MOV     A,#PROMPT          ;get the prompt char
     CALL    SP_OUT             ;send it to the console
     RET

;******************************MESSAGES***********************************

ERROR1:
     DB      '*ERROR - BAD DECIMAL DIGIT*',CR,LF,NULL
ERROR2:
     DB      '*ERROR - NUMBER TOO LARGE*',CR,LF,NULL
```

# Appendix D

# Programs for Determining the Resonant Frequencies and Mode Shapes

## D.1 MATLAB Programs

This section contains the MATLAB routines used to determine the resonant frequencies and mode shapes of the stator cylinder based on the analysis described in Chapter 4. They were written to run on MATLAB version 3.13 on an IBM PC compatible machine.

The program RES.M is the main driver program. Each time it is run, it reads the MATLAB data file CONST.MAT which contains the following variables (stored in binary format):

| | | |
|---|---|---|
| mu | – | Poisson's ratio |
| rho | – | mass density (kg/m$^3$) |
| e | – | modulus of elasticity (Pa) |
| lz | – | axial length of the stator (m) |
| r1 | – | inner radius of the stator (m) |
| r2 | – | outer radius of the stator (m) |

The user is allowed to change any of these variables before the analysis begins. The program also prompts the user for the mode number "n" and the model size "m". All of these variables are stored in the vector variable "args". .

The matrix "A" in Equation 4.16 is generated by a call to the C routine GENA.EXE. This routine returns the matrix "anu" which is used by the MATLAB command NULL to determine the matrix "nu", the nullspace of "anu". The columns of "nu" contain the expansion coefficients of the deflection functions.

The vector "args" and the matrix "nu" are used by the C routine GENAB.EXE to generate the frequency equation of the stator (see Equation 4.4). This equation is solved using the MATLAB command EIG.

```
****************************************************************************
*                                                                          *
*                                  RES.M                                   *
*                                                                          *
****************************************************************************

clear
load const
y=1; n=NaN;
update=input('update motor parameters, (y) or (n) ? ');
if update == y,
  fprintf('mu = %6.5e ',mu), update=input('enter poisson ratio: ');
  if update ~= n, mu=update; end,
  fprintf('rho = %6.5e ',rho), update=input('enter the mass density: ');
  if update ~= n, rho=update; end,
  fprintf('e = %6.5e ',e),
  update=input('enter modulus of elasticity: ');
  if update ~= n, e=update; end,
  fprintf('lz = %6.5e ',lz), update=input('enter stator length: ');
  if update ~= n, lz=update; end,
  fprintf('r1 = %6.5e ',r1), update=input('enter inner stator radius: ');
  if update ~= n, r1=update; end,
  fprintf('r2 = %6.5e ',r2), update=input('enter outer stator radius: ');
  if update ~= n, r2=update; end,
  update=input('save new constants, (y) or (n) ? ');
  if update == y, save const e lz mu r1 r2 rho, end,
  fprintf('\nupdate finished\n'),
end
n=input('enter circumferential mode number : ');
m=input('order of model = ');
```

```
args=[m n e lz mu r1 r2 rho];
save d:zzzz args
!gena d:zzzz.mat
load d:zzzz
!del d:zzzz.mat
nu=null(anu); [nr,nc]=size(nu);
save res1 args nu
nn=n; n=NaN;
update=input('plot deflection funct
if update == y,
  rn=r1/r2; ln=lz/(2*r2); n=nn; gnu=nu;
  !copy zfuns.exe d:
  !copy rfuns.exe d:
  !d:
  fprintf('\nnumber of vectors = %2.0f\n',nc);
  zp=input('zp = '); args(9)=zp; vec=input('vector number = ');
  while vec ~= 0,
    args(10)=vec;
    save zzzz args gnu
    !rfuns zzzz.mat
    load zzzz
    plot(r,[f1 f2 f3]), grid, pause,
    fprintf('\nnumber of vectors = %2.0f  ',nc);
    vec=input('vector number = ');
  end,
  rp=input('rp = '); args(9)=rp; vec=input('vector number = ');
  while vec ~= 0,
    args(10)=vec;
    save zzzz args gnu
    !zfuns zzzz.mat
    load zzzz
    plot(z,[f1 f2 f3]), grid, pause,
    fprintf('\nnumber of vectors = %2.0f  ',nc);
    vec=input('vector number = ');
  end,
  !del rfuns.exe
  !del zfuns.exe
  !del zzzz.mat
  !c:
end
clear
fprintf('\ngenerating frequency equation\n'),
!genab res1.mat res2.mat
load res2
fprintf('\nsolving generalized eigenvalue problem\n'),
```

cement distribution

le shape can be

ut 101 radial

specified

ts at

```
[x,d]=eig(a,b);
d=diag(d);
f=find(d==NaN);
d(f)=[]; x(:,f)=[];
if all(imag(d)==0)==0, d=real(d); x=real(x); end
f=find(d<=0);
d(f)=[]; x(:,f)=[];
[d f]=sort(d); x=x(:,f);
w0sq=e/(2*(1+mu)*rho*r2*r2);
eval=sqrt(w0sq*d)/(2*pi); f=[]; f1=20e3;
while length(f)<2, f=find(eval<=f1); f=find(eval(f)>1); f1=f1+10e3; end
f=[f eval(f)], [nrf,ncf]=size(f); g=(1:length(x))';
for i = 1:nrf,
  g1=x(:,f(i,1));
  g=[g,g1/norm(g1)];
end
g=g(:,2:nrf+1);
fmin=min(eval(find(eval>1)));
fprintf('fmin = %4.0f\n',fmin)
clear nrf ncf i g1 eval d x
y=1; nn=n; n=NaN;
update=input('plot mode shapes, (y) or (n) ? ');
n=nn;
if update == y, u, end
```

The program U.M is used to plot the components of the displacement distribution function (Equation 4.8 evaluated with $\theta = 0$ and $t = 0$) so that the mode shape can be determined. It calls the C routine RFUNS.C which calculates $\mathcal{U}_r$, $\mathcal{U}_\theta$, and $\mathcal{U}_z$ at 101 radial positions between the inner and outer radii "r1" and "r2", respectively, at a user specified axial position. The C routine ZFUNS.C is called to determine the three displacements at 101 positions along the length of the stator at a user specified radial position.

```
***************************************************************************
*                                                                         *
*                                   U.M                                   *
*                                                                         *
***************************************************************************

rn=r1/r2; ln=lz/(2*r2);
[nr,nc]=size(g); gnu=nu*g;
!copy zfuns.exe d:
!copy rfuns.exe d:
!d:
fprintf('\nnumber of vectors = %2.0f\n',nc);
rp=input('rp = ');
args=[m n e lz mu r1 r2 rho rp];
for vec=1:nc,
  args(10)=vec;
  save zzzz args gnu
  !zfuns zzzz.mat
  load zzzz
  titl=['n=',int2str(n),'  m=',int2str(m),' r=',num2str(rp),'  f='];
  titl=[titl,int2str(round(f(vec,2)))];
  plot(z,f1), xlabel('z'), ylabel('ur'), title(titl), grid,
  keyboard,
  plot(z,f2), xlabel('z'), ylabel('utheta'), title(titl), grid,
  keyboard,
  plot(z,f3), xlabel('z'), ylabel('uz'), title(titl), grid,
  keyboard,
end
zp=input('zp = '); args(9)=zp;
for vec=1:nc,
  args(10)=vec;
  save zzzz args gnu
  !rfuns zzzz.mat
  load zzzz
  titl=['n=',int2str(n),'  m=',int2str(m),' z=',num2str(zp),'  f='];
  titl=[titl,int2str(round(f(vec,2)))];
  plot(r,f1), xlabel('r'), ylabel('ur'), title(titl), grid,
  keyboard,
  plot(r,f2), xlabel('r'), ylabel('utheta'), title(titl), grid,
  keyboard,
  plot(r,f3), xlabel('r'), ylabel('uz'), title(titl), grid,
  keyboard,
end
!del zfuns.exe
!del rfuns.exe
!del zzzz.mat
!c:
fprintf('done\n');
```

## D.2 C Programs

This section contains the source code for the C programs used by RES.M and U.M. These programs were compiled using the Microsoft 5.0 C Language Compiler [20].

The program GENA.C is used to generate the matrix "A" in Equation 4.16 (this matrix is labeled "anu" in this program and in RES.M). Assuming that GENA.EXE is on the default drive, the program can be executed from MATLAB by typing the following line at the MATLAB prompt:

!gena datafile

where the command line argument DATAFILE is a MATLAB binary datafile containing the variable "args" defined below:

| | | |
|---|---|---|
| args[0] = m | – | model size parameter |
| args[1] = n | – | circumferential mode number |
| args[2] = e | – | modulus of elasticity (Pa) |
| args[3] = lz | – | axial length of the stator (m) |
| args[4] = mu | – | Poisson's ratio |
| args[5] = r1 | – | inner radius of the stator (m) |
| args[6] = r2 | – | outer radius of the stator (m) |
| args[7] = rho | – | mass density (kg/m$^3$) |

After the matrix "anu" is generated, it is appended to the end of DATAFILE.

```
****************************************************************************
*                                                                          *
*                                 GENA.C                                   *
*                                                                          *
****************************************************************************

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#define MIN(x, y)      (((x) < (y)) ? (x) : (y))
#define MAX(x, y)      (((x) > (y)) ? (x) : (y))
```

```
/* function declarations */

loadm(FILE *fp,char *pname,size_t *mrows,size_t *ncols,size_t *imagf,
     double far *xreal,double far *ximag);
savem(FILE *fp,char *pname,size_t mrows,size_t ncols,size_t imagf,
     double far *xreal,double far *ximag);

int n, m, nr, nc;
double mu, rho, e, lz, r1, r2;
double far *anu, far args[12];

main(argc,argv)
int argc;
char *argv[];
{
     size_t rows, cols, imag;
     FILE *fp, *fopen();
     if ((anu=(double far *)_fmalloc(8188*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          exit(1); }
     if (argc != 2) {
          printf("wrong number of arguments\n");
          printf("correct usage is: gena datafile\n");
          exit(1); }
     if ((fp=fopen(argv[1],"ab+"))==NULL) {
          printf("can't open %s\n",argv[1]);
          _ffree(anu); exit(1); }
     if (loadm(fp,"args",&rows,&cols,&imag,args,args) != 0) {
          printf("can't read args in %s\n",argv[1]);
          quit(fp); exit(1); }
     m=args[0]; n=args[1];
     nr = 4*m+2; nc = 3*m*m;     /* anu will be nr x nc */
     for (rows=0;rows<nr*nc;rows++) anu[rows]=0.0;
     e=args[2]; lz=args[3]; mu=args[4]; r1=args[5];
     r2=args[6]; rho=args[7];
     init();
     gena();
     if (savem(fp,"anu",nr,nc,0,anu,anu) != 0) {
          printf("error writing anu to file %s\n",argv[1]);
          quit(fp); exit(1); }
     quit(fp);
}

quit(fp)
```

```
FILE *fp;
{
     _ffree(anu);
     fclose(fp);
}

static double ln[10], mln[10], rn[10];

init()
{
     int i;
     double ri, li;
     ri=r1/r2; li=lz/(2*r2);
     ln[0]=mln[0]=rn[0]=1.0;
     for (i=1; i<10; i++) {
          rn[i]=rn[i-1]*ri;
          ln[i]=ln[i-1]*li;
          mln[i]=mln[i-1]*(-li); }
}

gena()
{
     int i, j, k, br1, br2, br3, bdij, beij;
     double kb;
     kb=mu/(1-mu); bdij=nr*m*m; beij=2*bdij;
     br1=m+1; br2=2*br1; br3=br2+m;
     for (j=0; j<m; j++)
          for (i=0; i<m; i++) {
               k=(i*m+j)*nr+j;
               anu[k]=kb*(j+1)*ln[i];
               anu[bdij+k]=kb*n*ln[i];
               if (i==0) anu[beij+k+1]=0;
               else anu[beij+k+1]=i*ln[i-1];
               anu[k+br1]=kb*(j+1)*mln[i];
               anu[bdij+k+br1]=kb*n*mln[i];
               if (i==0) anu[beij+k+br1+1]=0;
               else anu[beij+k+br1+1]=i*mln[i-1]; }
     for (i=0; i<m; i++)
          for (j=0; j<m; j++) {
               k=(i*m+j)*nr+i;
               if (j==0) {
                    anu[br2+k]=(j+kb)/rn[1];
                    anu[br2+bdij+k]=n*kb/rn[1]; }
               else {
                    anu[br2+k]=(j+kb)*rn[j-1];
```

```
                        anu[br2+bdij+k]=n*kb*rn[j-1]; }
                   if (i!=0) anu[br2+beij+k-1]=i*kb*rn[j];
                   anu[br3+k]=j+kb;
                   anu[br3+bdij+k]=n*kb;
                   if (i!=0) anu[br3+beij+k-1]=i*kb; }
}

\* The following structure appears at the beginning of all MATLAB
   binary datafiles. *\

typedef struct {
     long type;     /* type */
     long mrows;    /* row dimension */
     long ncols;    /* column dimension */
     long imagf;    /* flag indicating imag part */
     long namlen;   /* name length (including NULL) */
} Fmatrix;

\* The procedure LOADM is used to read a matrix variable from a MATLAB
   binary datafile. */

loadm(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;           /* file pointer */
char *pname;        /* pointer to matrix name */
size_t *mrows;      /* row dimension */
size_t *ncols;      /* column dimension */
size_t *imagf;      /* imaginary flag */
double far *xreal;  /* array of real data */
double far *ximag;  /* array of imaginary data */
{
     char mname[20];
     Fmatrix x;
     size_t i, j, nr, nc, namlen, fread(), found;
     double buf;
     found=0;
     while (fread((char *)&x, sizeof(Fmatrix), 1, fp) == 1) {
          if (x.type != 0) {
               printf("bad variable type\n");
               return(1); }
          *mrows = nr = x.mrows;
          *ncols = nc = x.ncols;
          *imagf = x.imagf;
          namlen = x.namlen;
          if (fread(mname, sizeof(char), namlen, fp) != namlen) {
               printf("error reading variable name\n");
```

```
                    return(1); }
              if (strcmp(mname,pname) == 0) { found=1; break; }
              for (j=0;j<nc;j++)
                    for (i=0;i<nr;i++)
                          if (fread(&buf, sizeof(double), 1, fp) != 1) {
                          printf("error reading real data\n");
                          return(1); }
              if (x.imagf) {
                    for (j=0;j<nc;j++)
                          for (i=0;i<nr;i++)
                                if (fread(&buf, sizeof(double), 1, fp) != 1) {
                                printf("error reading imaginary data\n");
                                return(1); } } }
        if (found == 0) return(1);
        for (j=0;j<nc;j++)
              for (i=0;i<nr;i++) {
                    if (fread(&buf, sizeof(double), 1, fp) != 1) {
                          printf("error reading real data\n");
                          return(1); }
                    *xreal=buf;
                    xreal++; }
        if (x.imagf) {
              for (j=0;j<nc;j++)
                    for (i=0;i<nr;i++) {
                          if (fread(&buf, sizeof(double), 1, fp) != 1) {
                                printf("error reading imaginary data\n");
                                return(1); }
                          *ximag=buf;
                          ximag++; } }
        rewind(fp);
        return(0);
}

\* The procedure SAVEM is used to store a matrix variable in a MATLAB
   binary datafile. */

savem(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;            /* File pointer */
char *pname;         /* pointer to matrix name */
size_t mrows;        /* row dimension */
size_t ncols;        /* column dimension */
size_t imagf;        /* imaginary flag */
double far *xreal;   /* pointer to real data */
double far *ximag;   /* pointer to imag data */
{
    Fmatrix x;
    size_t i, j, fwrite();
    double buf;
    x.type = 0;
    x.mrows = mrows;
    x.ncols = ncols;
    x.imagf = imagf;
    x.namlen = strlen(pname) + 1;
    if (fwrite(&x, sizeof(Fmatrix), 1, fp) != 1) return(1);
    if (fwrite(pname, sizeof(char), (size_t)x.namlen, fp) != x.namlen)
         return(1);
    for (j=0;j<ncols;j++)
         for (i=0;i<mrows;i++) {
              buf=*xreal;
              xreal++;
              if (fwrite(&buf, sizeof(double), 1, fp) != 1) return(1); }
    if (imagf)
         for (j=0;j<ncols;j++)
              for (i=0;i<mrows;i++) {
                   buf=*ximag;
                   ximag++;
                   if (fwrite(&buf,sizeof(double),1,fp) != 1) return(1); }
    return(0);
}
```

The program GENAB.C is used to generate the "A" and "B" matrices in Equation 4.4. Assuming that GENAB.EXE is on the default drive, the program can be executed from MATLAB by typing the following line at the MATLAB prompt:

!genab infile outfile

where the command line arguments INFILE and OUTFILE are MATLAB binary datafiles. The file INFILE contains the variable "args" which was defined above and the variable "nu", the matrix containing the expansion coefficients for the deflection functions. After the "A" and "B" matrices are generated, they are written, along with the variables "m", "n", "mu", "rho", "e", "r1", "r2", and "lz" and the matrix "nu" to the output file OUTFILE. The file is processed by the MATLAB M-file RES.M to yield the resonant frequencies and their corresponding eigenvectors. The eigenvectors are normalized to unity length and are stored columnwise in the matrix "g".

```
*****************************************************************************
*                                                                           *
*                                 GENAB.C                                   *
*                                                                           *
*****************************************************************************

#include <stdio.h>
#include <math.h>

/* function declarations */

loadm(FILE *fp,char *pname,size_t *mrows,size_t *ncols,size_t *imagf,
     double *xreal,double *ximag);
savem(FILE *fp,char *pname,size_t mrows,size_t ncols,size_t imagf,
     double *xreal,double *ximag);

double c1[256], c2[256], c3[256], c4[256];
double c5[256], c6[256], c7[256], c8[256];
double a[900], b[900], nu[1440], t1[20];
int n, m;
```

```
double mu, rho, e, lz, r1, r2, rn, ln, w0sq;

main(argc,argv)
int argc;
char *argv[];
{
     size_t i, j, ind, rows, cols, imag, ierr, eflag, size;
     FILE *fp, *fopen();
     if (argc != 3) {
          printf("wrong number of arguments\n");
          printf("correct usage is: genab infile outfile\n");
          exit(1); }
     if ((fp=fopen(argv[1],"rb"))==NULL) {
          printf("can't open %s\n",argv[1]);
          exit(1); }
     if (loadm(fp,"args",&rows,&cols,&imag,t1,t1) != 0) {
          printf("can't read args in %s\n",argv[1]);
          fclose(fp); exit(1); }
     m=t1[0]; printf("m = %d\n",m); n=t1[1]; printf("n = %d\n",n);
     e=t1[2]; lz=t1[3]; mu=t1[4]; r1=t1[5];
     r2=t1[6]; rho=t1[7];
     printf("mu  = %6.5e\n",mu); printf("rho = %6.5e\n",rho);
     printf("e   = %6.5e\n",e);  printf("lz  = %6.5e\n",lz);
     printf("r1  = %6.5e\n",r1); printf("r2  = %6.5e\n",r2);
     if (loadm(fp,"nu",&rows,&cols,&imag,nu,a) != 0) {
          printf("can't read nu in %s\n",argv[1]);
          fclose(fp); exit(1); }
     fclose(fp);
     genab(rows,cols);
     if ((fp=fopen(argv[2],"wb"))==NULL) {
          printf("can't open %s\n",argv[2]);
          exit(1); }
     if (savem(fp,"a",cols,cols,0,a,a) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"b",cols,cols,0,b,b) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"nu",rows,cols,0,nu,nu) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     t1[0]=m;
     if (savem(fp,"m",1,1,0,t1,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     t1[0]=n;
     if (savem(fp,"n",1,1,0,t1,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"mu",1,1,0,&mu,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"rho",1,1,0,&rho,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"e",1,1,0,&e,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"lz",1,1,0,&lz,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"r1",1,1,0,&r1,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     if (savem(fp,"r2",1,1,0,&r2,t1) != 0) {
          printf("error writing to file %s\n",argv[2]);
          fclose(fp);
          exit(1); }
     fclose(fp);
}

genab(nr,nc)
int nr, nc;
{
     int g, h, i, j, k, l, jpl, ipk, indg, indh;
     int ns, ms, ms2, ij, kl, x1, x2, x3, x4;
     double pi, den, mu1, temp, p1, p2;
     double fr1(), fr2(), fr3(), z1(), z2(), z3();
     double rr1, rr2, rr3, rz1, rz2, rz3;
     double *pa, *pb, *pc1, *pc2, *pc3;
```

```
    double *pc4, *pc5, *pc6, *pc7, *pc8;
    rn=r1/r2; ln=lz/(2*r2); ns=n*n; ms=m*m; ms2=2*ms; pi=acos(-1.0);
    den=1-2*mu; mu1=1-mu;
    if (n==0) { p1=2*pi; p2=0; }
    else p1=p2=pi;
    printf("p1=%6.5e p2=%6.5e\n",p1,p2);
    printf("\n");
    pc1=c1; pc2=c2; pc3=c3; pc4=c4; pc5=c5; pc6=c6; pc7=c7; pc8=c8;
    for (i=0;i<m;i++)
      for (j=0;j<m;j++)
        for (k=0;k<m;k++)
          for (l=0;l<m;l++) {
            printf("i=%d j=%d k=%d l=%d\r",i,j,k,l);
            jpl=j+l; ipk=i+k; rr1=fr1(jpl); rr2=fr2(jpl); rr3=fr3(jpl);
            rz1=z1(ipk); rz2=z2(ipk); rz3=z3(ipk);
            temp=(ns*p2+2*p1*(mu*jpl+mu1*(1+j*l))/den)*rr3*rz1;
            *pc1=temp+i*k*p1*rr1*rz3;
            *pc2=2*n*((1-l)*p2+2*p1*(mu*j+mu1)/den)*rr3*rz1;
            *pc3=2*p1*(i*l+2*mu*k*(j+1)/den)*rr2*rz2;
            temp=(2*p1*ns*mu1/den+(j-1)*(l-1)*p2)*rr3*rz1;
            *pc4=temp+i*k*p2*rr1*rz3;
            *pc5=2*n*(2*mu*k*p1/den-i*p2)*rr2*rz2;
            temp=(2*mu1*i*k/den)*p1*rr1*rz3;
            *pc6=temp+(j*l*p1+ns*p2)*rr3*rz1;
            *pc7=2*p1*rr1*rz1;
            *pc8=2*p2*rr1*rz1;
            pc1++; pc2++; pc3++; pc4++; pc5++; pc6++; pc7++; pc8++; }
    printf("\ninitialization complete\n\n");
    printf("A and B matrices will have size %d x %d\n\n",nc,nc);
    pa=&a[0]; pb=&b[0];

/*  Since the A and B matrices are symmetric only the lower triangular
    portion is calculated. The upper portion is filled in later.    */

    for (g=0;g<nc;g++) { indg=g*nr;
    for (h=g;h<nc;h++) { indh=h*nr; *pa=*pb=0.0;
    printf("g=%2d\th=%2d\r",g,h);
    pc1=c1; pc2=c2; pc3=c3; pc4=c4; pc5=c5; pc6=c6; pc7=c7; pc8=c8;
    for (i=0;i<m;i++)
      for (j=0;j<m;j++) { ij=i*m+j; x1=indg+ij; x3=indh+ij;
        for (k=0;k<m;k++)
          for (l=0;l<m;l++) {
            kl=k*m+l; x2=indg+kl; x4=indh+kl;
            temp=2*nu[x1]*nu[x4]*(*pc1);
            temp+=(nu[x1]*nu[x4+ms]+nu[x3]*nu[x2+ms])*(*pc2);
```

```
                temp+=(nu[x1]*nu[x4+ms2]+nu[x3]*nu[x2+ms2])*(*pc3);
                *pa+=temp+2*nu[x1+ms]*nu[x4+ms]*(*pc4);
                temp=(nu[x1+ms]*nu[x4+ms2]+nu[x3+ms]*nu[x2+ms2])*(*pc5);
                *pa+=temp+2*nu[x1+ms2]*nu[x4+ms2]*(*pc6);
                temp=(nu[x1]*nu[x4]+nu[x1+ms2]*nu[x4+ms2])*(*pc7);
                *pb+=temp+(nu[x1+ms]*nu[x4+ms])*(*pc8);
                pc1++; pc2++; pc3++; pc4++; pc5++; pc6++; pc7++; pc8++; } }
                pa++; pb++; } pa+=g+1; pb+=g+1; }
     printf("\n\n");

/*   Fill in the upper portion   */

     for (g=0;g<nc;g++)
          for (h=g+1;h<nc;h++) {
               printf("g=%2d\th=%2d\r",g,h);
               a[nc*h+g]=a[nc*g+h];
               b[nc*h+g]=b[nc*g+h]; }
     printf("\n\n");
}

double fr1(x)
int x;
{
     return((1.0-pow(rn,x+2.0))/(x+2.0));
}

double fr2(x)
int x;
{
     return((1.0-pow(rn,x+1.0))/(x+1.0));
}

double fr3(x)
int x;
{
     if (x==0) return(-log(rn));
     return((1.0-pow(rn,(double)x))/(double)x);
}

double z1(x)
int x;
{
     if ((x & 1) != 0) return(0.0);
     return(pow(ln,x+1.0)/(x+1.0));
}
```

```
double z2(x)
int x;
{
     if ((x & 1) == 0) return(0.0);
     if (x==0) return(0.0);
     return(pow(ln,(double)x)/(double)x);
}

double z3(x)
int x;
{
     if ((x & 1) != 0) return(0.0);
     if (x<=1) return(0.0);
     else return(pow(ln,x-1.0)/(x-1.0));
}

\* The following structure appears at the beginning of all MATLAB
   binary datafiles. *\

typedef struct {
     long type;      /* type */
     long mrows;     /* row dimension */
     long ncols;     /* column dimension */
     long imagf;     /* flag indicating imag part */
     long namlen;    /* name length (including NULL) */
} Fmatrix;

\* The procedure LOADM is used to read a matrix variable from a MATLAB
   binary datafile. */


loadm(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;           /* file pointer */
char *pname;        /* pointer to matrix name */
size_t *mrows;      /* row dimension */
size_t *ncols;      /* column dimension */
size_t *imagf;      /* imaginary flag */
double far *xreal;  /* array of real data */
double far *ximag;  /* array of imaginary data */
{
     char mname[20];
     Fmatrix x;
     size_t i, j, nr, nc, namlen, fread(), found;
     double buf;
```

```
      found=0;
      while (fread((char *)&x, sizeof(Fmatrix), 1, fp) == 1) {
           if (x.type != 0) {
                printf("bad variable type\n");
                return(1); }
           *mrows = nr = x.mrows;
           *ncols = nc = x.ncols;
           *imagf = x.imagf;
           namlen = x.namlen;
           if (fread(mname, sizeof(char), namlen, fp) != namlen) {
                printf("error reading variable name\n");
                return(1); }
           if (strcmp(mname,pname) == 0) { found=1; break; }
           for (j=0;j<nc;j++)
                for (i=0;i<nr;i++)
                     if (fread(&buf, sizeof(double), 1, fp) != 1) {
                     printf("error reading real data\n");
                     return(1); }
           if (x.imagf) {
                for (j=0;j<nc;j++)
                     for (i=0;i<nr;i++)
                          if (fread(&buf, sizeof(double), 1, fp) != 1) {
                          printf("error reading imaginary data\n");
                          return(1); } } }
      if (found == 0) return(1);
      for (j=0;j<nc;j++)
           for (i=0;i<nr;i++) {
                if (fread(&buf, sizeof(double), 1, fp) != 1) {
                     printf("error reading real data\n");
                     return(1); }
                *xreal=buf;
                xreal++; }
      if (x.imagf) {
           for (j=0;j<nc;j++)
                for (i=0;i<nr;i++) {
                     if (fread(&buf, sizeof(double), 1, fp) != 1) {
                          printf("error reading imaginary data\n");
                          return(1); }
                     *ximag=buf;
                     ximag++; } }
      rewind(fp);
      return(0);
}

\* The procedure SAVEM is used to store a matrix variable in a MATLAB
```

```
   binary datafile. */

savem(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;            /* File pointer */
char *pname;         /* pointer to matrix name */
size_t mrows;        /* row dimension */
size_t ncols;        /* column dimension */
size_t imagf;        /* imaginary flag */
double far *xreal;   /* pointer to real data */
double far *ximag;   /* pointer to imag data */
{
     Fmatrix x;
     size_t i, j, fwrite();
     double buf;
     x.type = 0;
     x.mrows = mrows;
     x.ncols = ncols;
     x.imagf = imagf;
     x.namlen = strlen(pname) + 1;
     if (fwrite(&x, sizeof(Fmatrix), 1, fp) != 1) return(1);
     if (fwrite(pname, sizeof(char), (size_t)x.namlen, fp) != x.namlen)
          return(1);
     for (j=0;j<ncols;j++)
          for (i=0;i<mrows;i++) {
               buf=*xreal;
               xreal++;
               if (fwrite(&buf, sizeof(double), 1, fp) != 1) return(1); }
     if (imagf)
          for (j=0;j<ncols;j++)
               for (i=0;i<mrows;i++) {
                    buf=*ximag;
                    ximag++;
                    if (fwrite(&buf,sizeof(double),1,fp) != 1) return(1); }
     return(0);
}
```

The program RFUNS.C is used by U.M to generate the radial, tangential, and axial components of either the displacement distribution function (Equation 4.8) or the deflection function (Equation 4.9) at 101 radial positions between the inner and outer radii of the stator. Assuming that RFUNS.EXE is on the default drive, the program can be executed from MATLAB by typing the following line at the MATLAB prompt:

!rfuns datafile

where the command line argument DATAFILE is a MATLAB binary datafile containing the variables "args" and "gnu". The variable "args" is the same as above except for two additional elements defined as follows:

| | | |
|---|---|---|
| args[8] = zp | – | normalized axial position |
| args[9] = vec | – | column number of gnu |

To calculate the i'th displacement distribution function, the variable "vec" should be set to i and the matrix "gnu" to the matrix product of "nu" and "g". To calculate the i'th deflection function, the variable "vec" should again be set to i and the matrix "gnu" to "nu".

```
****************************************************************************
*                                                                          *
*                                RFUNS.C                                   *
*                                                                          *
****************************************************************************

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#define MIN(x, y)     (((x) < (y)) ? (x) : (y))
#define MAX(x, y)     (((x) > (y)) ? (x) : (y))

/* function declarations */
```

```
loadm(FILE *fp,char *pname,size_t *mrows,size_t *ncols,size_t *imagf,
     double far *xreal,double far *ximag);
savem(FILE *fp,char *pname,size_t mrows,size_t ncols,size_t imagf,
     double far *xreal,double far *ximag);

int n, m, vec;
double e, lz, mu, r1, r2, rho, zp;
double far *f1, far *f2, far *f3, far *r, far *gnu, far args[20];

main(argc,argv)
int argc;
char *argv[];
{
     size_t rows, cols, imag, offset;
     FILE *fp, *fopen();
     if (argc != 2) {
          printf("wrong number of arguments\n");
          printf("correct usage is: rfuns datafile\n");
          exit(1); }
     memalloc();
     if ((fp=fopen(argv[1],"ab+"))==NULL) {
          printf("can't open %s\n",argv[1]);
          memfree(); exit(1); }
     if (loadm(fp,"args",&rows,&cols,&imag,args,args) != 0) {
          printf("can't read args in %s\n",argv[1]);
          quit(fp); exit(1); }
     m=args[0]; n=args[1]; e=args[2]; lz=args[3]; mu=args[4];
     r1=args[5]; r2=args[6]; rho=args[7]; zp=args[8]; vec=args[9];
     if (loadm(fp,"gnu",&rows,&cols,&imag,gnu,gnu) != 0) {
          printf("can't read gnu in %s\n",argv[1]);
          quit(fp); exit(1); }
     offset=(vec-1)*rows;
     gnu=gnu+offset;
     rfuns();
     if (savem(fp,"f1",101,1,0,f1,f1) != 0) {
          printf("error writing f1 to file %s\n",argv[1]);
          quit(fp); exit(1); }
     if (savem(fp,"f2",101,1,0,f2,f2) != 0) {
          printf("error writing f2 to file %s\n",argv[1]);
          quit(fp); exit(1); }
     if (savem(fp,"f3",101,1,0,f3,f3) != 0) {
          printf("error writing f3 to file %s\n",argv[1]);
          quit(fp); exit(1); }
     if (savem(fp,"r",101,1,0,r,r) != 0) {
          printf("error writing r to file %s\n",argv[1]);
          quit(fp); exit(1); }
     quit(fp);
}

memalloc()
{
     f1=f2=f3=r=gnu=NULL;
     if ((f1=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          exit(1); }
     if ((f2=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree();
          exit(1); }
     if ((f3=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree();
          exit(1); }
     if ((r=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree();
          exit(1); }
     if ((gnu=(double far *)_fmalloc(8188*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree();
          exit(1); }
}

memfree()
{
     _ffree(f1);
     _ffree(f2);
     _ffree(f3);
     _ffree(r);
     _ffree(gnu);
}

quit(fp)
FILE *fp;
{
     memfree();
     fclose(fp);
}

rfuns()
```

```
{
     int i, j, k, ind1, ind2, ind3;
     double rn, rpj, zpi, temp;
     rn=r1/r2; temp=(1-rn)/100;
     for (i=0; i<101; i++) r[i]=rn+i*temp;
     for (k=0; k<101; k++) {
          f1[k]=f2[k]=f3[k]=0;
          for (i=0; i<m; i++) {
               zpi=pow(zp,(double)i);
               for (j=0; j<m; j++) {
                    rpj=pow(r[k],(double)j);
                    ind1=i*m+j; ind2=m*m+ind1; ind3=2*m*m+ind1;
                    f1[k]+=gnu[ind1]*rpj*zpi;
                    f2[k]+=gnu[ind2]*rpj*zpi;
                    f3[k]+=gnu[ind3]*rpj*zpi; } } }
}

typedef struct {
     long type;     /* type */
     long mrows;    /* row dimension */
     long ncols;    /* column dimension */
     long imagf;    /* flag indicating imag part */
     long namlen;   /* name length (including NULL) */
} Fmatrix;

\* The procedure LOADM is used to read a matrix variable from a MATLAB
   binary datafile. */

loadm(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;             /* file pointer */
char *pname;          /* pointer to matrix name */
size_t *mrows;        /* row dimension */
size_t *ncols;        /* column dimension */
size_t *imagf;        /* imaginary flag */
double far *xreal;    /* array of real data */
double far *ximag;    /* array of imaginary data */
{
     char mname[20];
     Fmatrix x;
     size_t i, j, nr, nc, namlen, fread(), found;
     double buf;
     found=0;
     while (fread((char *)&x, sizeof(Fmatrix), 1, fp) == 1) {
          if (x.type != 0) {
               printf("bad variable type\n");
```

```
                return(1); }
            *mrows = nr = x.mrows;
            *ncols = nc = x.ncols;
            *imagf = x.imagf;
            namlen = x.namlen;
            if (fread(mname, sizeof(char), namlen, fp) != namlen) {
                printf("error reading variable name\n");
                return(1); }
            if (strcmp(mname,pname) == 0) { found=1; break; }
            for (j=0;j<nc;j++)
                for (i=0;i<nr;i++)
                    if (fread(&buf, sizeof(double), 1, fp) != 1) {
                    printf("error reading real data\n");
                    return(1); }
            if (x.imagf) {
                for (j=0;j<nc;j++)
                    for (i=0;i<nr;i++)
                        if (fread(&buf, sizeof(double), 1, fp) != 1) {
                        printf("error reading imaginary data\n");
                        return(1); } } }
        if (found == 0) return(1);
        for (j=0;j<nc;j++)
            for (i=0;i<nr;i++) {
                if (fread(&buf, sizeof(double), 1, fp) != 1) {
                    printf("error reading real data\n");
                    return(1); }
                *xreal=buf;
                xreal++; }
        if (x.imagf) {
            for (j=0;j<nc;j++)
                for (i=0;i<nr;i++) {
                    if (fread(&buf, sizeof(double), 1, fp) != 1) {
                        printf("error reading imaginary data\n");
                        return(1); }
                    *ximag=buf;
                    ximag++; } }
        rewind(fp);
        return(0);
}

\* The procedure SAVEM is used to store a matrix variable in a MATLAB
   binary datafile. */

savem(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;           /* File pointer */
```

```
char *pname;          /* pointer to matrix name */
size_t mrows;         /* row dimension */
size_t ncols;         /* column dimension */
size_t imagf;         /* imaginary flag */
double far *xreal;    /* pointer to real data */
double far *ximag;    /* pointer to imag data */
{
     Fmatrix x;
     size_t i, j, fwrite();
     double buf;
     x.type = 0;
     x.mrows = mrows;
     x.ncols = ncols;
     x.imagf = imagf;
     x.namlen = strlen(pname) + 1;
     if (fwrite(&x, sizeof(Fmatrix), 1, fp) != 1) return(1);
     if (fwrite(pname, sizeof(char), (size_t)x.namlen, fp) != x.namlen)
          return(1);
     for (j=0;j<ncols;j++)
          for (i=0;i<mrows;i++) {
               buf=*xreal;
               xreal++;
               if (fwrite(&buf, sizeof(double), 1, fp) != 1) return(1); }
     if (imagf)
          for (j=0;j<ncols;j++)
               for (i=0;i<mrows;i++) {
                    buf=*ximag;
                    ximag++;
                    if (fwrite(&buf,sizeof(double),1,fp) != 1) return(1); }
     return(0);
}
```

The program ZFUNS.C is used by U.M to generate the radial, tangential, and axial components of either the displacement distribution function (Equation 4.8) or the deflection function (Equation 4.9) at 101 axial positions along the length of the stator. Assuming that ZFUNS.EXE is on the default drive, the program can be executed from MATLAB by typing the following line at the MATLAB prompt:

```
!zfuns datafile
```

where the command line argument DATAFILE is a MATLAB binary datafile containing the variables "args" and "gnu". The variable "args" is the same as above except for two additional elements defined as follows:

| | | |
|---|---|---|
| args[8] = rp | – | normalized radial position |
| args[9] = vec | – | column number of gnu |

To calculate the i'th displacement distribution function, the variable "vec" should be set to i and the matrix "gnu" to the matrix product of "nu" and "g". To calculate the i'th deflection function, the variable "vec" should again be set to i and the matrix "gnu" to "nu".

```
*************************************************************************
*                                                                       *
*                               ZFUNS.C                                 *
*                                                                       *
*************************************************************************

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#define MIN(x, y)     (((x) < (y)) ? (x) : (y))
#define MAX(x, y)     (((x) > (y)) ? (x) : (y))

/* function declarations */
```

```
loadm(FILE *fp,char *pname,size_t *mrows,size_t *ncols,size_t *imagf,
     double far *xreal,double far *ximag);
savem(FILE *fp,char *pname,size_t mrows,size_t ncols,size_t imagf,
     double far *xreal,double far *ximag);

int n, m, vec;
double e, lz, mu, r1, r2, rho, rp;
double far *f1, far *f2, far *f3, far *z, far *gnu, far args[20];

main(argc,argv)
int argc;
char *argv[];
{
     size_t rows, cols, imag, offset;
     FILE *fp, *fopen();
     if (argc != 2) {
          printf("wrong number of arguments\n");
          printf("correct usage is: zfuns datafile\n");
          exit(1); }
     memalloc();
     if ((fp=fopen(argv[1],"ab+"))==NULL) {
          printf("can't open %s\n",argv[1]);
          memfree(); exit(1); }
     if (loadm(fp,"args",&rows,&cols,&imag,args,args) != 0) {
          printf("can't read args in %s\n",argv[1]);
          quit(fp); exit(1); }
     m=args[0]; n=args[1]; e=args[2]; lz=args[3]; mu=args[4];
     r1=args[5]; r2=args[6]; rho=args[7]; rp=args[8]; vec=args[9];
     if (loadm(fp,"gnu",&rows,&cols,&imag,gnu,gnu) != 0) {
          printf("can't read gnu in %s\n",argv[1]);
          quit(fp); exit(1); }
     offset=(vec-1)*rows;
     gnu=gnu+offset;
     zfuns();
     if (savem(fp,"f1",101,1,0,f1,f1) != 0) {
          printf("error writing f1 to file %s\n",argv[1]);
          quit(fp); exit(1); }
     if (savem(fp,"f2",101,1,0,f2,f2) != 0) {
          printf("error writing f2 to file %s\n",argv[1]);
          quit(fp); exit(1); }
     if (savem(fp,"f3",101,1,0,f3,f3) != 0) {
          printf("error writing f3 to file %s\n",argv[1]);
          quit(fp); exit(1); }
     if (savem(fp,"z",101,1,0,z,z) != 0) {
          printf("error writing z to file %s\n",argv[1]);
          quit(fp); exit(1); }
      quit(fp);
}

memalloc()
{
      f1=f2=f3=z=gnu=NULL;
      if ((f1=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
            printf("insufficient memory available\n");
            exit(1); }
      if ((f2=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
            printf("insufficient memory available\n");
            memfree();
            exit(1); }
      if ((f3=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
            printf("insufficient memory available\n");
            memfree();
            exit(1); }
      if ((z=(double far *)_fmalloc(101*sizeof(double)))==NULL) {
            printf("insufficient memory available\n");
            memfree();
            exit(1); }
      if ((gnu=(double far *)_fmalloc(8188*sizeof(double)))==NULL) {
            printf("insufficient memory available\n");
            memfree();
            exit(1); }
}

memfree()
{
      _ffree(f1);
      _ffree(f2);
      _ffree(f3);
      _ffree(z);
      _ffree(gnu);
}

quit(fp)
FILE *fp;
{
      memfree();
      fclose(fp);
}

zfuns()
```

```
{
     int i, j, k, ind1, ind2, ind3;
     double ln, rpj, zpi, temp;
     ln=lz/(2*r2); temp=ln/50;
     for (i=0; i<101; i++) z[i]=-ln+i*temp;
     for (k=0; k<101; k++) {
          f1[k]=f2[k]=f3[k]=0;
          for (i=0; i<m; i++) {
               zpi=pow(z[k],(double)i);
               for (j=0; j<m; j++) {
                    rpj=pow(rp,(double)j);
                    ind1=i*m+j; ind2=m*m+ind1; ind3=2*m*m+ind1;
                    f1[k]+=gnu[ind1]*rpj*zpi;
                    f2[k]+=gnu[ind2]*rpj*zpi;
                    f3[k]+=gnu[ind3]*rpj*zpi; } } }
}

typedef struct {
     long type;     /* type */
     long mrows;    /* row dimension */
     long ncols;    /* column dimension */
     long imagf;    /* flag indicating imag part */
     long namlen;   /* name length (including NULL) */
} Fmatrix;

\* The procedure LOADM is used to read a matrix variable from a MATLAB
   binary datafile. */

loadm(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;            /* file pointer */
char *pname;         /* pointer to matrix name */
size_t *mrows;       /* row dimension */
size_t *ncols;       /* column dimension */
size_t *imagf;       /* imaginary flag */
double far *xreal;   /* array of real data */
double far *ximag;   /* array of imaginary data */
{
     char mname[20];
     Fmatrix x;
     size_t i, j, nr, nc, namlen, fread(), found;
     double buf;
     found=0;
     while (fread((char *)&x, sizeof(Fmatrix), 1, fp) == 1) {
          if (x.type != 0) {
               printf("bad variable type\n");
```

```
              return(1); }
          *mrows = nr = x.mrows;
          *ncols = nc = x.ncols;
          *imagf = x.imagf;
          namlen = x.namlen;
          if (fread(mname, sizeof(char), namlen, fp) != namlen) {
              printf("error reading variable name\n");
              return(1); }
          if (strcmp(mname,pname) == 0) { found=1; break; }
          for (j=0;j<nc;j++)
              for (i=0;i<nr;i++)
                  if (fread(&buf, sizeof(double), 1, fp) != 1) {
                  printf("error reading real data\n");
                  return(1); }
          if (x.imagf) {
              for (j=0;j<nc;j++)
                  for (i=0;i<nr;i++)
                      if (fread(&buf, sizeof(double), 1, fp) != 1) {
                      printf("error reading imaginary data\n");
                      return(1); } } }
      if (found == 0) return(1);
      for (j=0;j<nc;j++)
          for (i=0;i<nr;i++) {
              if (fread(&buf, sizeof(double), 1, fp) != 1) {
                  printf("error reading real data\n");
                  return(1); }
              *xreal=buf;
              xreal++; }
      if (x.imagf) {
          for (j=0;j<nc;j++)
              for (i=0;i<nr;i++) {
                  if (fread(&buf, sizeof(double), 1, fp) != 1) {
                      printf("error reading imaginary data\n");
                      return(1); }
                  *ximag=buf;
                  ximag++; } }
      rewind(fp);
      return(0);
}

\* The procedure SAVEM is used to store a matrix variable in a MATLAB
   binary datafile. */

savem(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;             /* File pointer */
```

```
char *pname;         /* pointer to matrix name */
size_t mrows;        /* row dimension */
size_t ncols;        /* column dimension */
size_t imagf;        /* imaginary flag */
double far *xreal;   /* pointer to real data */
double far *ximag;   /* pointer to imag data */
{
     Fmatrix x;
     size_t i, j, fwrite();
     double buf;
     x.type = 0;
     x.mrows = mrows;
     x.ncols = ncols;
     x.imagf = imagf;
     x.namlen = strlen(pname) + 1;
     if (fwrite(&x, sizeof(Fmatrix), 1, fp) != 1) return(1);
     if (fwrite(pname, sizeof(char), (size_t)x.namlen, fp) != x.namlen)
          return(1);
     for (j=0;j<ncols;j++)
          for (i=0;i<mrows;i++) {
               buf=*xreal;
               xreal++;
               if (fwrite(&buf, sizeof(double), 1, fp) != 1) return(1); }
     if (imagf)
          for (j=0;j<ncols;j++)
               for (i=0;i<mrows;i++) {
                    buf=*ximag;
                    ximag++;
                    if (fwrite(&buf,sizeof(double),1,fp) != 1) return(1); }
     return(0);
}
```

# Appendix E

# VRM Simulation Programs

## E.1 Current-Flux Fitting Programs

The MATLAB M-file IFIT.M is used to determine the expansion coefficients $\beta_{kl}$ in Equation 5.9. It reads the data file IVSLAM.MAT which contains points of the flux-current curves at 16 different angles (see Figure 2.3) stored in the matrix variables "current" and "lambda". The expansion coefficients are stored in the variable "b" in the data file IFIT.MAT.

```
*************************************************************************
*                                                                       *
*                                IFIT.M                                 *
*                                                                       *
*************************************************************************

clear
n1=input('enter order of lambda polynomial: ');
n2=input('enter order of Fourier cosine series in theta: ');
load ivslam
fprintf('theta = ');
for theta=0:2:30,
  fprintf('%.0f,',theta);
  c1=theta/2+1; lam=lambda(:,c1);
  coef=pfit(lam,current(:,c1),n1);
  a(c1,:)=coef;
end
fprintf('\n');
for c1=17:31, a(c1,:)=a(32-c1,:); end
```

```
theta=pi*[(0:2:60)']/180;
theta1=pi*[(0:.6:60)']/180;
fprintf('index = ');
for c1=1:n1,
  fprintf('%.0f,',c1);
  coef=cosfit(6*theta,a(:,c1),n2);
  b(c1,:)=coef;
  fit1(:,c1)=cosval(coef,6*theta1);
end
fprintf('\n');
save ifit b
for c1=1:n1, plot((0:2:60)',a(:,c1),'o',(0:.6:60)',fit1(:,c1),'-'), pause,
end
theta=pi*[(0:2:30)']/180;
for c1=1:n1, afit(:,c1)=cosval(b(c1,:),6*theta); end
for c1=1:length(theta),
  lam=lambda(:,c1);
  fit(:,c1)=polyval([afit(c1,:) 0],lam);
end
for c1=1:length(theta),
  titl=[int2str(2*c1-2)];
  plot(current(:,c1),lambda(:,c1),fit(:,c1),lambda(:,c1)), title(titl),
  pause,
end
clear c1 coef titl lam
```

```
****************************************************************************
*                                                                          *
*                                 PFIT.M                                   *
*                                                                          *
****************************************************************************

function p = pfit(x,y,n)

% PFIT(x,y,n) finds the coefficients of a polynomial formed from the data
% in vector x of degree n that fits the data in vector y in a least-squares
% sense where it is known that the polynomial passes through the origin.
% The polynomial has the following form for n=3:
% y = p1 * x^3 + p2 * x^2 + p3 * x

if size(x) ~= size(y)
     disp('X and Y vectors must be the same size')
     return
end
x = x(:); y = y(:);
A = zeros(max(size(x)),n);
for j=1:n
     A(:,j) = x.^(n-j+1);
end
p = (A\y).';
```

```
**************************************************************************
*                                                                        *
*                                COSFIT.M                                *
*                                                                        *
**************************************************************************

function p = cosfit(x,y,n)

% COSFIT(x,y,n) finds the coefficients of a cosine polynomial formed from
% the data in vector x of degree n that fits the data in vector y in a
% least-squares sense. The polynomial has the following form for n=3:
% y = p1 * cos(3x) + p2 * cos(2x) + p3 * cos(x) + p4

if size(x) ~= size(y),
     disp('X and Y vectors must be the same size'),
     return,
end
x = x(:); y = y(:);
n = n + 1;
a=zeros(length(x),n);
for j=1:n,
     a(:,j)=cos((n-j)*x);
end
p = (a\y).';
```

```
***************************************************************************
*                                                                         *
*                               COSVAL.M                                  *
*                                                                         *
***************************************************************************

function y = cosval(c,x)

% COSVAL(x,y,n) evaluates the cosine polynomial described by the
% coefficients in the vector c at all points in the vector x.
% The polynomial has the following form for n=3:
% y = c1 * cos(3x) + c2 * cos(2x) + c3 * cos(x) + c4

n = length(c);
[nr,nc]=size(x);
y = zeros(nr,nc);
for i=1:n,
y=y+c(i)*cos((n-i)*x);
end
```

## E.2 Motor Simulation Programs

The program MOTOR.M is the top-level driver for the simulation program. It reads the data file MOTOR.MAT which contains the following variables:

| | | |
|---|---|---|
| b | – | matrix of expansion coefficients (see Table 5.1) |
| ichop | – | current chopping level (amps) |
| itheta | – | initial rotor position (radians) |
| npts | – | number of simulation points |
| nt | – | number of turns per phase |
| off | – | turn-off angle (radians) |
| on | – | turn-on angle (radians) |
| tchop | – | chopping period (seconds) |
| vs | – | supply voltage (volts) |
| w | – | rotor speed (rad/sec) |

The user is allowed to make changes to any of these variables before control is passed to the C program MOTOR.EXE which performs the actual simulations.

```
*****************************************************************************
*                                                                           *
*                                 MOTOR.M                                   *
*                                                                           *
*****************************************************************************

clear
load motor
y=1; n=NaN;
res=input('update motor parameters, (y) or (n) ? ');
if res == y,
  fprintf('npts=%.0f',npts), res=input('enter number of data points: ');
  if res ~= n, npts=res; end,
  fprintf('w=%.5g',w*30/pi), res=input('enter motor speed (rpm): ');
  if res ~= n, w=res*pi/30; end,
  fprintf('ichop=%.5g',ichop), res=input('enter maximum current (amps): ');
  if res ~= n, ichop=res; end,
  fprintf('on=%.5g',on*180/pi), res=input('enter on angle (degrees): ');
  if res ~= n, on=res*pi/180; end,
  fprintf('off=%.5g',off*180/pi), res=input('enter off angle (degrees): ');
  if res ~= n, off=res*pi/180; end,
  fprintf('itheta=%.5g',itheta*180/pi),
  res=input('enter initial rotor position (degrees): ');
  if res ~= n, itheta=res*pi/180; end,
  fprintf('vs=%.5g',vs), res=input('enter supply voltage (volts): ');
  if res ~= n, vs=res; end,
  fprintf('tchop=%.5e',tchop),
  res=input('enter chopping period (seconds): ');
  if res ~= n, tchop=res; end,
  res=input('save new parameters, (y) or (n) ? ');
  if res == y,
    save motor b ichop itheta npts nt off on tchop vs w
  end,
end
save mottemp b ichop itheta npts nt off on tchop vs w
clear
!motor mottemp.mat
load mottemp
!del mottemp.mat
if exist('t')==1,
  clg, xlab='time in seconds';
  titl=['on=',num2str(on*180/pi),' off=',num2str(off*180/pi)];
  titl=[titl,' vs=',num2str(vs),' tchop=',num2str(tchop)];
  ylab='current'; subplot(211)
  plot(t,i), grid, xlabel(xlab), ylabel(ylab), title(titl)
  ylab='flux'; subplot(212)
  plot(t,lam), grid, xlabel(xlab), ylabel(ylab), title(titl), pause
  clg, subplot(211), xlab='rotor position in degrees'; ylab='current';
  angle=180*theta/pi;
  plot(angle,i), grid, xlabel(xlab), ylabel(ylab), title(titl)
  ylab='torque'; subplot(212)
  plot(angle,torque), grid, xlabel(xlab), ylabel(ylab), title(titl),
  pause,
  clg, subplot(211), xlab='rotor position in degrees'; ylab='current';
  plot(angle,i), grid, xlabel(xlab), ylabel(ylab), title(titl)
  ylab='radial force'; subplot(212)
  plot(angle,rf), grid, xlabel(xlab), ylabel(ylab), title(titl), pause
  subplot(111)
  plot(angle,[area/norm(area) i/norm(i) rf/norm(rf) lam/norm(lam)]),
  xlabel(xlab)
end
```

The program MOTOR.C performs the actual motor simulation. It is invoked by typing the following line at the MATLAB prompt:

!motor datafile

where the command line argument DATAFILE is a MATLAB binary datafile containing the same variables that are in the file MOTOR.MAT. Since the phases of the VRM are identical, only one phase is actually simulated. The results of the simulation, which consist of the following variables, are appended to the end of DATAFILE. All variables are vectors of data points unless otherwise noted.

| | | |
|---|---|---|
| t | – | time (seconds) |
| lam | – | flux (volt-seconds) |
| theta | – | rotor position (radians) |
| i | – | current (amperes) |
| torque | – | torque for one phase (Newton-meters) |
| tav | – | average torque (Newton-meters, scalar) |
| tort | – | total torque of all phases (Newton-meters) |
| rf | – | radial force (Newtons) |
| area | – | air gap overlap area (meters$^2$) |

```
****************************************************************************
*                                                                          *
*                                 MOTOR.C                                  *
*                                                                          *
****************************************************************************

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#define MIN(x, y)     (((x) < (y)) ? (x) : (y))
#define MAX(x, y)     (((x) > (y)) ? (x) : (y))
#define NS 2
#define MAXLEN 4096

/* function declarations */

loadm(FILE *fp,char *pname,size_t *mrows,size_t *ncols,size_t *imagf,
     double far *xreal,double far *ximag);
savem(FILE *fp,char *pname,size_t mrows,size_t ncols,size_t imagf,
     double far *xreal,double far *ximag);

double far *tout, far *lam, far *angle, far *cur, far *area;
double far *tor, far *tort, far temp[10], far *tp, far *rf;
double b[100], ichop, itheta, off, on, tav, tchop, vs, w;
int nr, nc, npts, nt;

main(argc, argv)
int argc;
char *argv[];
{
     size_t rows, cols, imag;
     FILE *fp, *fopen();
     if (argc != 2) {
          printf("wrong number of arguments\n");
          printf("correct usage is - motor datafile\n");
          exit(1); }
     if ((fp=fopen(argv[1],"ab+"))==NULL) {
          printf("can't open %s\n",argv[1]);
          exit(1); }
     tp=&b[0];
     if (loadm(fp,"b",&nr,&nc,&imag,tp,tp) != 0) {
          printf("can't read 'b' in %s\n",argv[1]);
          quit(fp); }
     if (loadm(fp,"ichop",&rows,&cols,&imag,temp,temp) != 0) {
          printf("can't read 'ichop' in %s\n",argv[1]);
          quit(fp); }
     ichop=temp[0]; printf("ichop=%.5g\n",ichop);
     if (loadm(fp,"itheta",&rows,&cols,&imag,temp,temp) != 0) {
          printf("can't read 'itheta' in %s\n",argv[1]);
          quit(fp); }
     itheta=temp[0]; printf("itheta=%.5g\n",itheta);
     if (loadm(fp,"npts",&rows,&cols,&imag,temp,temp) != 0) {
          printf("can't read 'npts' in %s\n",argv[1]);
          quit(fp); }
     npts=temp[0]; printf("npts=%d\n",npts);
     if (loadm(fp,"nt",&rows,&cols,&imag,temp,temp) != 0) {
          printf("can't read 'nt' in %s\n",argv[1]);
          quit(fp); }
     nt=temp[0]; printf("nt=%d\n",nt);
     if (loadm(fp,"off",&rows,&cols,&imag,temp,temp) != 0) {
          printf("can't read 'off' in %s\n",argv[1]);
          quit(fp); }
```

```
off=temp[0]; printf("off=%.5g\n",off);
if (loadm(fp,"on",&rows,&cols,&imag,temp,temp) != 0) {
     printf("can't read 'on' in %s\n",argv[1]);
     quit(fp); }
on=temp[0]; printf("on=%.5g\n",on);
if (loadm(fp,"tchop",&rows,&cols,&imag,temp,temp) != 0) {
     printf("can't read 'tchop' in %s\n",argv[1]);
     quit(fp); }
tchop=temp[0]; printf("tchop=%.5e\n",tchop);
if (loadm(fp,"vs",&rows,&cols,&imag,temp,temp) != 0) {
     printf("can't read 'vs' in %s\n",argv[1]);
     quit(fp); }
vs=temp[0]; printf("vs=%.5g\n",vs);
if (loadm(fp,"w",&rows,&cols,&imag,temp,temp) != 0) {
     printf("can't read 'w' in %s\n",argv[1]);
     quit(fp); }
w=temp[0]; printf("w=%.5g\n",w);
if (npts>MAXLEN) {
     printf("error: maximum number of points is %d\n",MAXLEN);
     quit(fp); }
if (memalloc()) quit(fp);
rk();
forces();
if (savem(fp,"t",npts,1,0,tout,tout) != 0) {
     printf("error writing to file %s\n",argv[1]);
     memfree(); quit(fp); }
if (savem(fp,"lam",npts,1,0,lam,lam) != 0) {
     printf("error writing to file %s\n",argv[1]);
     memfree(); quit(fp); }
if (savem(fp,"theta",npts,1,0,angle,angle) != 0) {
     printf("error writing to file %s\n",argv[1]);
     memfree(); quit(fp); }
if (savem(fp,"i",npts,1,0,cur,cur) != 0) {
     printf("error writing to file %s\n",argv[1]);
     memfree(); quit(fp); }
if (savem(fp,"torque",npts,1,0,tor,tor) != 0) {
     printf("error writing to file %s\n",argv[1]);
     memfree(); quit(fp); }
temp[0]=tav;
if (savem(fp,"tav",1,1,0,temp,temp) != 0) {
     printf("error writing to file %s\n",argv[1]);
     memfree(); quit(fp); }
if (savem(fp,"tort",npts,1,0,tort,tort) != 0) {
     printf("error writing to file %s\n",argv[1]);
     memfree(); quit(fp); }
     if (savem(fp,"area",npts,1,0,area,area) != 0) {
          printf("error writing to file %s\n",argv[1]);
          memfree(); quit(fp); }
     if (savem(fp,"rf",npts,1,0,f,f) != 0) {
          printf("error writing to file %s\n",argv[1]);
          memfree(); quit(fp); }
     memfree();
     fclose(fp);
}

mot(double t, double *y, double *m);
double current(double theta, double lambda);
double torque(double theta, double lambda);
double an15, an30, an45, an60, h, pi, tlast;

int rk()
{
     int i, ind;
     double t, y[NS], yt[NS], m1[NS], m2[NS], m3[NS], m4[NS];
     double atemp;
     pi=acos(-1.0); an60=pi/3.0; an45=pi/4.0; an30=pi/6.0; an15=pi/12.0;
     h=pi/(3.0*w*npts); tlast=t=0.0; y[0]=0.0; y[1]=itheta;
     atemp=2*(2.54e-4)*.02/(4*pi*(1e-7)*nt*nt);
     for (ind=0;ind<npts;ind++,t+=h) {
          tout[ind]=t; lam[ind]=y[0]; angle[ind]=y[1];
          cur[ind]=current(y[1],y[0]); tor[ind]=torque(y[1],y[0]);
          rf[ind]=lam[ind]*cur[ind]/(4*2.54e-4);
          area[ind]=atemp/current(y[1],0.02);
          mot(t,y,m1);
          for (i=0;i<NS;i++) yt[i]=y[i]+h*m1[i]/2;
          mot(t+h/2,yt,m2);
          for (i=0;i<NS;i++) yt[i]=y[i]+h*m2[i]/2;
          mot(t+h/2,yt,m3);
          for (i=0;i<NS;i++) yt[i]=y[i]+h*m3[i];
          mot(t+h,yt,m4);
          for (i=0;i<NS;i++) y[i]+=h*(m1[i]+2*(m2[i]+m3[i])+m4[i])/6;
          if (t-tlast>tchop) tlast+=tchop;
          printf("ind=%4d\r",ind);
     }
     printf("\n");
}

forces()
{
     int i, offb, offc, offd, indb, indc, indd;
     offb=(3*npts)/4; offc=npts/2; offd=npts/4;
     for (i=0;i<npts;i++) {
          indb=(i+offb)%npts; indc=(i+offc)%npts; indd=(i+offd)%npts;
          tort[i]=tor[i]+tor[indb]+tor[indc]+tor[indd];
     }
     tort[npts]=tort[0];
     for (i=0,tav=0.0;i<npts/2;i++)
          tav+=tort[2*i]+4*tort[2*i+1]+tort[2*i+2];
     tav/=3.0*npts;
     printf("tav=%.5e\n",tav);
}

#define PHASE(on,off,an) ((off>=on) ?
     (an>=on && an<=off):(an>=on || an<=off))
int ph=0;
double v=0;

mot(t, y, m)
double t, y[], m[];
{
     double theta, i, r;
     theta=fmod(y[1],an60);
     i=current(theta,y[0]);
     if (PHASE(on,off,theta)) {
          if (ph) {
               if (i>ichop) {v=-0.7; r=0.18;}
               else if (t-tlast>tchop) {v=vs; r=0.36;}}
          else {ph=1; v=vs; r=0.36;}}
     else {
          ph=0;
          if (y[0]>0) {v=-(vs+1.4); r=0.0;} else {v=r=0.0;} }
     m[0]=-(0.8+r)*i+v;
     m[1]=w;
}

double current(theta,lambda)
double theta, lambda;
{
     int c1, c2;
     double a[5], lpow, i, cm, *bp;
     theta*=6.0;
     for (c1=0;c1<nr;c1++) a[c1]=0.0;
     for (c1=0;c1<nc;c1++) {
          bp=&b[nr*c1]; cm=cos((nc-c1-1)*theta);
          for (c2=0;c2<nr;c2++) a[c2]+=bp[c2]*cm; }
     for (c1=nr-1,lpow=lambda,i=0.0;c1>=0;c1--,lpow*=lambda) i+=a[c1]*lpow;
     return(i);
}

double torque(theta,lambda)
double theta, lambda;
{
     int c1, c2;
     double a[5], lpow, torque, sm, *bp;
     theta*=6.0;
     for (c1=0;c1<nr;c1++) a[c1]=0.0;
     for (c1=0;c1<nc;c1++) {
          bp=&b[nr*c1]; sm=(nc-c1-1)*sin((nc-c1-1)*theta);
          for (c2=0;c2<nr;c2++) a[c2]+=bp[c2]*sm; }
     lpow=lambda*lambda; torque=0.0;
     for (c1=nr-1,c2=2;c1>=0;c1--,c2++,lpow*=lambda)
          torque+=a[c1]*lpow/c2;
     return(6.0*torque);
}

memalloc()
{
     tout=lam=angle=cur=area=tor=tort=rf=NULL;
     if ((tout=(double far *)_fmalloc(npts*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          return(1); }
     if ((lam=(double far *)_fmalloc(npts*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree(); return(1); }
     if ((angle=(double far *)_fmalloc(npts*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree(); return(1); }
     if ((cur=(double far *)_fmalloc(npts*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree(); return(1); }
     if ((area=(double far *)_fmalloc(npts*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree(); return(1); }
     if ((tor=(double far *)_fmalloc(npts*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree(); return(1); }
     if ((tort=(double far *)_fmalloc((npts+1)*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree(); return(1); }
     if ((rf=(double far *)_fmalloc(npts*sizeof(double)))==NULL) {
          printf("insufficient memory available\n");
          memfree(); return(1); }
     return(0);
}

memfree()
{
     _ffree(tout);
     _ffree(lam);
     _ffree(angle);
     _ffree(cur);
     _ffree(area);
     _ffree(tor);
     _ffree(tort);
     _ffree(rf);
}

quit(fp)
FILE *fp;
{
     fclose(fp);
     exit(1);
}

typedef struct {
     long type;      /* type */
     long mrows;     /* row dimension */
     long ncols;     /* column dimension */
     long imagf;     /* flag indicating imag part */
     long namlen;    /* name length (including NULL) */
} Fmatrix;

\* The procedure LOADM is used to read a matrix variable from a MATLAB
   binary datafile. */

loadm(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;            /* file pointer */
char *pname;         /* pointer to matrix name */
size_t *mrows;       /* row dimension */
size_t *ncols;       /* column dimension */
size_t *imagf;       /* imaginary flag */
double far *xreal;   /* array of real data */
double far *ximag;   /* array of imaginary data */
{
     char mname[20];
```

```
Fmatrix x;
size_t i, j, nr, nc, namlen, fread(), found;
double buf;
found=0;
while (fread((char *)&x, sizeof(Fmatrix), 1, fp) == 1) {
     if (x.type != 0) {
          printf("bad variable type\n");
          return(1); }
     *mrows = nr = x.mrows;
     *ncols = nc = x.ncols;
     *imagf = x.imagf;
     namlen = x.namlen;
     if (fread(mname, sizeof(char), namlen, fp) != namlen) {
          printf("error reading variable name\n");
          return(1); }
     if (strcmp(mname,pname) == 0) { found=1; break; }
     for (j=0;j<nc;j++)
          for (i=0;i<nr;i++)
               if (fread(&buf, sizeof(double), 1, fp) != 1) {
               printf("error reading real data\n");
               return(1); }
     if (x.imagf) {
          for (j=0;j<nc;j++)
               for (i=0;i<nr;i++)
                    if (fread(&buf, sizeof(double), 1, fp) != 1) {
                    printf("error reading imaginary data\n");
                    return(1); } } }
if (found == 0) return(1);
for (j=0;j<nc;j++)
     for (i=0;i<nr;i++) {
          if (fread(&buf, sizeof(double), 1, fp) != 1) {
               printf("error reading real data\n");
               return(1); }
          *xreal=buf;
          xreal++; }
if (x.imagf) {
     for (j=0;j<nc;j++)
          for (i=0;i<nr;i++) {
               if (fread(&buf, sizeof(double), 1, fp) != 1) {
                    printf("error reading imaginary data\n");
                    return(1); }
               *ximag=buf;
               ximag++; } }
rewind(fp);
return(0);
```

```
}

\* The procedure SAVEM is used to store a matrix variable in a MATLAB
   binary datafile. */

savem(fp, pname, mrows, ncols, imagf, xreal, ximag)
FILE *fp;            /* File pointer */
char *pname;         /* pointer to matrix name */
size_t mrows;        /* row dimension */
size_t ncols;        /* column dimension */
size_t imagf;        /* imaginary flag */
double far *xreal;   /* pointer to real data */
double far *ximag;   /* pointer to imag data */
{
     Fmatrix x;
     size_t i, j, fwrite();
     double buf;
     x.type = 0;
     x.mrows = mrows;
     x.ncols = ncols;
     x.imagf = imagf;
     x.namlen = strlen(pname) + 1;
     if (fwrite(&x, sizeof(Fmatrix), 1, fp) != 1) return(1);
     if (fwrite(pname, sizeof(char), (size_t)x.namlen, fp) != x.namlen)
          return(1);
     for (j=0;j<ncols;j++)
          for (i=0;i<mrows;i++) {
               buf=*xreal;
               xreal++;
               if (fwrite(&buf, sizeof(double), 1, fp) != 1) return(1); }
     if (imagf)
          for (j=0;j<ncols;j++)
               for (i=0;i<mrows;i++) {
                    buf=*ximag;
                    ximag++;
                    if (fwrite(&buf,sizeof(double),1,fp) != 1) return(1); }
     return(0);
}
```