This is to certify that the

dissertation entitled

POSITION AND SPEED SENSORLESS CONTROL OF PERMANENT MAGNET SYNCHRONOUS MOTORS

presented by

Ali Khurram

has been accepted towards fulfillment of the requirements for

Ph.D degree in Electrical Engineering

Major professor

Date 2 March 2001

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

Position and Speed Sensorless Control of Permanent Magnet Synchronous Motors

By

Ali Khurram

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering

2001

i
f
l
c
c
t
p
ac
pr
be
co
fo
Th

ABSTRACT

Position and Speed Sensorless Control of Permanent Magnet Synchronous Motors

By

Ali Khurram

Advancements in magnetic materials, semiconductor switching devices, and control strategies continue to enhance the popularity of permanent magnet synchronous motors (PMSMs) in drive applications. Their emergence as the actuators of choice in servo systems stems from their desirable features: compact structure, high airgap flux density, high torque capability for a given frame size, high torque-to-inertia ratio, low maintenance cost, mechanical simplicity and ruggedness. They are electronically commutated and can be made to achieve conventional DC motor characteristics, without the high maintenance cost of DC motors. But their control is more complicated: they need position sensors to synchronize the stator magnetic field with the rotor position. Also, the application of vector control techniques in AC drives demands accurate position and speed feedback information. The use of such shaft sensors present several disadvantages: cost, reliability, motor size, and weight. There has been extensive research on elimination of rotor-mounted position sensors in PMSM control. This dissertation presents an improved position and speed observer suitable for use with a surface-mounted, sinusoidal EMF PMSM as a software transducer. The proposed scheme works in a closed loop fashion. It enhances the allowable initial

position error. It is computationally less intensive; avoids integration of the speed estimate to get the position estimate; can be used without any physical modification; does not rely on rotor saliency and requires no knowledge of the load. The observer is developed from the *dq* model of the motor. Estimation of position and speed is done using differences between estimates of the current derivatives in the *dq* frame, each calculated two different ways: first using high-gain observers, and then using the motor model. The estimator equations are derived. Convergence of the observer dynamics is proved. Results from numerical simulations and practical implementation are presented to validate the proposed scheme.

To My Parents

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Professor Elias Strangas for being my advisor and guidance committee chairperson. His encouragement and unique ideas have been instrumental to the completion of this dissertation.

I am also deeply indebted to Professor Hassan Khalil for his guidance throughout my stay at Michigan State University. I owe him a great deal for his contribution to the analysis part of this work. My sincere appreciation also goes to Professors Robert Schlueter and Byron Drachman for the valuable time and effort they spent being on my guidance committee.

I owe special thanks to my parents, who encouraged me to pursue my dream and supported me with their prayers. My sincere appreciation goes to my wife Tahira for her love and support that enabled me to focus on this work. I cannot thank enough my brother Sajjad for his patient listening ear and sage advice. I thank Almighty for Humza and Fatima, my wonderful children, whose love made it all worthwhile. I would also like to extend my heartfelt thanks to my sisters Annie and Sughra, and all members of my extended family for their love, support and wishes.

I was lucky to have the company of excellent friends at the Electrical Machines and Drives Laboratory, specially Fida and Wes. Their constructive comments and sincere friendship are deeply appreciated.

LIS

LIS

1

2

3

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Introduction

Traditionally, electric motors have been used as actuators. However, they may also be used as sensors of the motion they actuate. This simultaneous operation as an actuator and a sensor can be beneficial when additional motion sensors are too expensive, large, unreliable, or otherwise undesirable. Advancement of the state of the art of such a software transducer for the permanent magnet synchronous motor (PMSM) is the motif of this dissertation.

This chapter first introduces the PMSM vis-à-vis the different types of actuators commonly employed. It continues with brief description of its structure, defines its niche, and presents the pros and cons of sensors. It concludes with a preview of the succeeding chapters.

1.2 Actuators

DC motors have been the most widespread actuators of choice in high performance systems. The principal reason for their popularity is their highly desirable characteristic: the ability to control their torque and flux independently. The trade-off is their less rugged construction, owing to the presence of brushes and commutators which tend to wear out, thus increasing the maintenance cost.

AC motors are more rugged, but they have traditionally been unsuitable for variable speed applications, because their torque and flux are coupled: any change in either one will cause a corresponding reaction in the other. This has changed now because of the emergence of new effective control techniques, mainly vector control. This control technique allows independent control of torque and flux of an AC motor, hence achieving linear torque characteristics resembling those of DC motors. Vector control regulates both the instantaneous magnitude and phase of the current, or the voltage, hence its name. This technique requires extensive processing power in order to achieve effective results and has become possible only with recent developments in affordable computing power.

Permanent magnet synchronous motors (PMSMs) have emerged as viable candidates for high-performance servo drive applications. The remarkable advances in semiconductor switching devices, as well as the recent availability of high energy-density permanent-magnet materials at competitive prices has opened up new possibilities for large-scale application of PMSMs. Furthermore, due to the continuing breakthroughs and reduction in cost of powerful microprocessors, the real-time implementation of sophisticated control schemes is becoming feasible, thus resulting in the possibility of achieving impressive performance.

The popularity of PMSMs stems form their desirable features:

- high efficiency,
- high torque to inertia ratio,
- high torque to volume ratio,
- high air gap flux density,

- high power to inertia ratio,
- high power factor,
- high acceleration and deceleration rates,
- lower maintenance cost,
- simplicity and ruggedness,
- compact structure,
- linear response.

However, the higher initial cost, operating temperature limitations, and danger of demagnetization can be restrictive for some applications.

The rotor of a PMSM has a permanent magnet mounted on it. Traditionally, a position sensor is also present. The signals to the phase windings of the PMSM stator are synchronized with the output from the position sensor to provide electronic commutation. By energizing specific windings in the stator, based on the position of the rotor, a revolving magnetic field is generated. Currents are switched in a predetermined sequence and hence the permanent magnets on the rotor are made to follow the revolving magnetic field. Since the switching frequency is derived from the rotor, the motor cannot lose synchronism. The current is always switched before the permanent magnets catch up. Therefore the speed of the motor is directly proportional to the current switching rate.

PMSMs are electronically commutated and can be made to achieve conventional DC motor characteristics: speed proportional to the supply voltage, torque proportional to the armature current, and start/stall torque higher than the running torque. They have all the advantages traditionally associated with conventional DC motors

such as better efficiency, response and linearity, without the high maintenance cost of their DC counterparts.

Elimination of brushes and commutators also solves the problems associated with contacts: brush noise, sparking and associated radio frequency interference from brush arcing. This enables the PMSM to be used in hostile and explosive environments. Commutation is achieved through reliable solid-state circuit components. Hence, their speed is not limited by the frictional components of mechanical commutation, but by the voltage limit of the control circuit and motor windings.

PMSMs have certain advantages over both induction motors (IMs) and the conventional wound-rotor synchronous motors (WRSMs): since there is no field winding on the rotor, there are no attendant rotor Copper losses, and the losses are mainly due to the stator current. Moreover, the power winding is on the stator where heat can be removed more easily.

A PMSM is more efficient than a comparable IM: in the steady state, the PMSM always operates at synchronous speed; thus it does not have the slip losses inherent in IM operation. Also, the stator current of an IM contains magnetizing as well as torque-producing components. The use of permanent magnet in the rotor of a PMSM makes it unnecessary to supply magnetizing current through the stator for constant air-gap flux; the stator current need only be torque-producing. Hence for the same output, the PMSM will operate at a higher power factor and will be more efficient than the IM. Finally, since the magnetization is provided from the rotor circuit instead of the stator, the motor can be built with a larger airgap without degraded performance. For these reasons, PMSM has a higher efficiency, torque per ampere, effective power factor and power density when compared with an IM. These factors combine to keep the torque/inertia ratio high in small motors which makes it preferable for certain high-performance applications such as robotics and aerospace actuators.

The smaller the motor, the more sense it makes to use permanent magnets for excitation. There is no single 'breakpoint', below which PMSMs outperform induction motors, but it is in the 1-10 kW range. Above this size the induction motor improves rapidly, while the cost of magnets works against the PMSM.

Compared with a conventional synchronous motor, elimination of the field coil, DC supply, and slip rings results in a much simpler motor. In a PMSM, there is no provision for rotor side excitation control. The control is done entirely through the stator excitation control. Field weakening is possible by applying a negative direct axis current to oppose the rotor magnet flux.

This elimination of the need for separate field excitation results in smaller overall size: for a given field strength, the PM assembly is considerably smaller in diameter than its wound field counterpart, providing substantial savings in both size and weight.

1.3 Sensors

The application of vector control techniques in AC drives demands accurate position information. The instantaneous angular position of the rotating field flux vector must be known for accurate vector control. The technique involves orienting the stator current vector orthogonally to the rotor flux vector, so as to maintain an appropriate space angle between the stator and rotor fields, a condition needed to maximize torque per ampere.

Since PMSMs are constructed with a fixed rotor field, supplied by rotor mounted magnets, the rotor position provides the required magnet flux position [29]. Rotor position information has been traditionally provided by shaft mounted optical position encoders, resolvers, or Hall-effect devices.

These devices tend to be expensive, although schemes for improving the data produced by cheaper, less accurate sensors have been suggested [12]. Apart from

higher cost and limited resolution, there are other drawbacks to using position sensors:

- lowered ruggedness and reliability,
- susceptibility to environmental conditions, such as temperature, humidity and vibration,
- increased number of connections,
- increased size and weight,
- added moment of inertia,
- added static and dynamic friction,
- interference problems,
- both ends of the shaft occupied.

For speed control, the speed signal is also needed. For robotics and machine-tools applications, the speed must be accurately controlled, in spite of load torque variations.

The conventional method of speed computation from position data has good performance in the ordinary speed range, but has the disadvantage that the method gives only the average speed during any detection interval; that is to say that the detected motor speed is not the instantaneous motor speed but the average over the last detection interval. In the low-speed region, the detection time becomes large due to the low frequency of the encoder pulse and the speed detection delay increases rapidly as the motor speed decreases. Another problem is the presence of an inherent lag in the estimation. Quantization noise from both the transducer and the estimation scheme is also introduced. Digital filters can be applied to address this problem, but the use of older samples compromises the instantaneous accuracy of the speed estimated.

Thus, transducer based algorithms do not provide an accurate speed estimate with high resolution.

As a consequence, there has been increasing interest in techniques for eliminating the rotor position and speed sensors, and hence bringing about improved reliability and reduced cost of the drive.

The idea behind many such methods is to manipulate the motor equations in order to express position and speed as functions of the terminal quantities. In a sense, the motor is used as a sensor of its motion, because this motion affects the voltages and currents in it. Thus, the voltages and currents possess information concerning the motion. What is needed, then, is a means for extracting this information to estimate the desired motion. Some form of signal processing is necessary, and this places additional demands on the control electronics because of the on-line computation. However, as digital processors continue to become faster and less expensive, this additional signal processing becomes less of a burden.

1.4 Organization

The rest of the chapters in this dissertation are organized as follows:

- Chapter 2 gives a review of the research conducted in this area over the last couple of decades.
- Chapter 3 presents the proposed method. It begins with the problem statement, continues with details of the proposed scheme and concludes with derivations of all the relevant equations.
- Chapter 4 describes and discusses the results obtained from numerical simulations of the proposed ideas.

- Chapter 5 presents mathematical analysis of the scheme and proves stability of the observer dynamics.

- Chapter 6 describes the experimental setup built to implement the scheme. It outlines the software developed and concludes with presentation of experimental results which validate the proposed scheme.

- Chapter 7 summarizes the whole dissertation and proposes avenues for further work.

- Appendix A contains a listing of the Assembly language code developed to implement the scheme.

CHAPTER 2

Literature Review

2.1 Introduction

In this chapter, we present a broad classification of the various techniques proposed to estimate the position and speed of the permanent magnet synchronous motor (PMSM). We begin with a taxonomy of the PMSMs and then develop their mathematical model, to make the literature review self-contained. The chapter concludes with a review of high-gain observers.

2.2 A Taxonomy of PMSMs

The PMSM rotor has magnets mounted either on the surface of the rotor, the so-called surface permanent-magnet motor (SPM), or there can be magnets buried inside the rotor, hence the name interior permanent-magnet motor (IPM). The SPMs have a smooth air gap, whereas saliency arises in the IPMs.

The IPMs are more robust and allow operation at higher speeds. Another advantage of an IPM is that a more efficient design can be obtained, since the electromagnetic torque contains both the magnet component and the reluctance component. The reluctance component arises because of the saliency characteristic of IPMs, thus al-

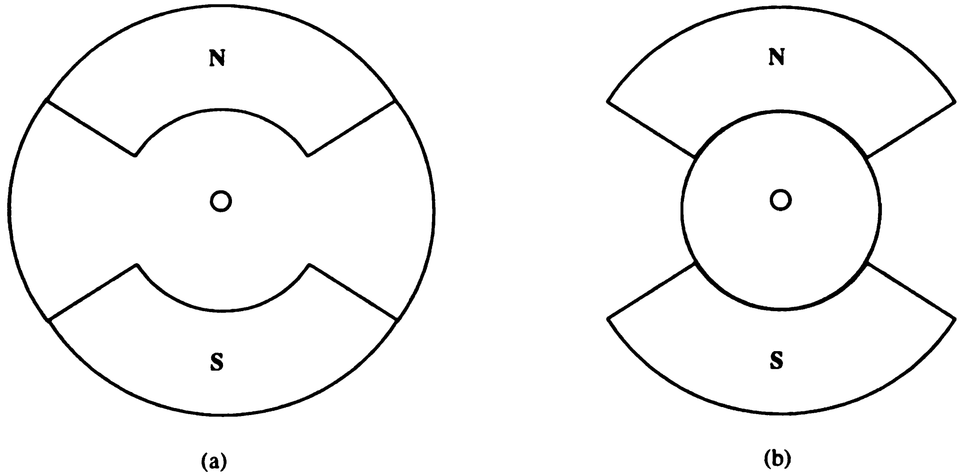

Figure 2.1. Magnet placement: (a) interior (b) surface.

lowing for a reduction in the quantity of magnet needed in comparison with SPMs for a fixed rated torque. The trade-off is the higher manufacturing cost associated with IPMs. Also, their narrower air gap means that armature reaction [31] is significant in IPMs.

In SPMs, because of the large airgap, the armature reaction effect on pole flux is insignificant. Therefore the variation of airgap flux under stator current change is minimized. This provides ease in flux control.

Another classification for PMSMs is based on the flux distribution [26], which translates into the shape of the back EMF generated: trapezoidal and sinusoidal. The waveform of the open-circuit voltages induced in the stator windings due to the permanent magnets, when the machine is run as a generator, determines this characteristic of the back EMF.

For a PMSM with trapezoidal flux distribution, also called brushless DC motor, only two of the three stator phases are excited at any instant of time, so that constant current flows into one of the excited windings and out of the other. The stator currents of the motor are square waves with 120 electrical-degree conduction periods. These

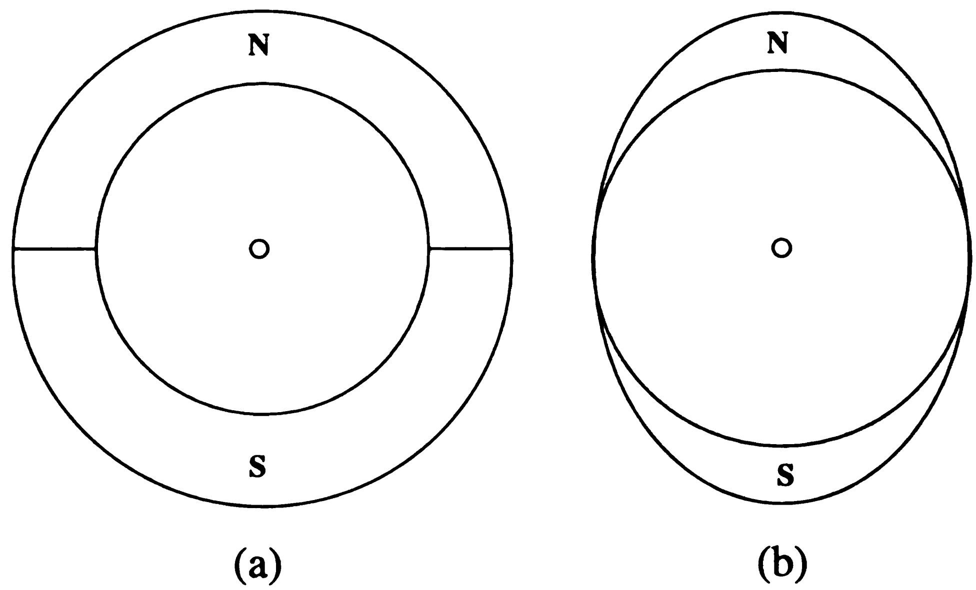

Figure 2.2. Flux distribution: (a) trapezoidal (b) sinusoidal.

rectangular current-fed motors have concentrated windings on the stator, and the induced voltage in the windings is square or trapezoidal. These motors cost less and are normally used in low-power drives.

For a PMSM with sinusoidal flux distribution, all three stator phases are excited. The stator currents are quasi-sinusoidal. The sensorless techniques for this type of motor are more involved. However, this motor is more suited for high performance applications. Also, the torque per ampere is higher because all three phases are excited simultaneously. The sinusoidal current-fed motors have distributed windings on the stator, provide smoother torque and are normally used in high-power applications.

2.3 The Mathematical Model

In this section, the equations of the surface-mounted, sinusoidal-EMF PMSM are presented.

The stator of a PMSM [31, 32] has windings similar to those of the conven-

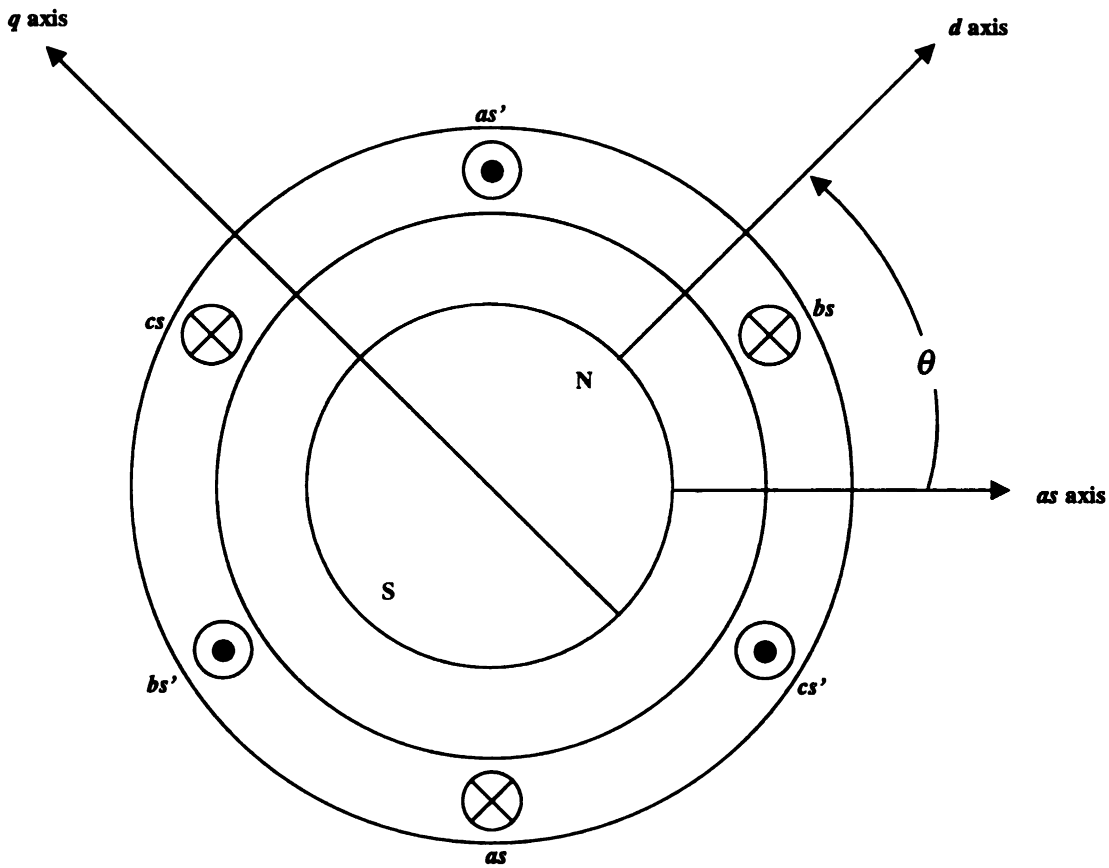

Figure 2.3. Two-pole three-phase PMSM.

tional wound-rotor synchronous motor: three-phase, Y-connected and sinusoidally distributed. The permanent magnets used in the PMSM have high resistivity, so induced currents in the rotor are negligible. Also, for a sinusoidal-EMF PMSM, there is no difference between the back EMF produced by a permanent magnet and that produced by an excited coil. Hence the mathematical model of a PMSM is similar to that of a wound rotor synchronous motor.

The space vector [24] form of the stator voltage equation in the stationary frame of reference is:

$$\overline{v} = R\overline{i} + \frac{d\overline{\psi}}{dt} \tag{2.1}$$

where,

- R is the resistance of the stator winding;

- $\overline{v}$, $\overline{i}$, and $\overline{\psi}$ are the complex space vectors of the three phase stator voltages, currents and flux linkages, all expressed in the stationary reference frame fixed to the stator. They are defined as:

$$\begin{aligned} \overline{v} &= \frac{2}{3}[v_a(t) + av_b(t) + a^2v_c(t)] \\ \overline{i} &= \frac{2}{3}[i_a(t) + ai_b(t) + a^2i_c(t)] \\ \overline{\psi} &= \frac{2}{3}[\psi_a(t) + a\psi_b(t) + a^2\psi_c(t)] \end{aligned} \tag{2.2}$$

where omitting the arguments for simplicity,

- a, a^2 are spatial operators: $a = e^{j2\pi/3}$, $a^2 = e^{j4\pi/3}$;

- v_a ,v_b, and v_c are the instantaneous values of the stator phase voltages;

- i_a, i_b, and i_c are the instantaneous values of the stator phase currents;

- ψ_a, ψ_b, and ψ_c are the instantaneous values of the stator flux linkages and are given by:

$$\begin{aligned} \psi_a &= L_si_a + Mi_b + Mi_c + \psi_r\cos(\theta) \\ \psi_b &= Mi_a + L_si_b + Mi_c + \psi_r\cos(\theta - \frac{2\pi}{3}) \\ \psi_c &= Mi_a + Mi_b + L_si_c + \psi_r\cos(\theta + \frac{2\pi}{3}) \end{aligned} \tag{2.3}$$

where,

- $L_s = L_{sl} + L_{sm}$: L_s, L_{sl}, and L_{sm} are the stator self, leakage and magnetizing inductances;

- M is the mutual inductance between the stator windings, and has a value [22]

$$\begin{aligned} M &= L_{sm}\cos(\frac{2\pi}{3}) \\ &= -\frac{1}{2}L_{sm} \end{aligned}$$

- ψ_r is the amplitude of the flux linkages established in the stator phase windings, by the permanent magnet. It is often referred to as the electromotive force constant, K_e;

- θ: the rotor position: the angle between the as-axis and the north pole of the permanent magnet.

If the windings are star-connected, $i_a + i_b + i_c = 0$, and defining

$$\begin{aligned} L &= L_s - M \\ &= L_{sl} + \frac{3}{2}L_{sm} \end{aligned}$$

the following phase-variable equations are obtained,

$$\begin{aligned} v_a &= Ri_a + L\frac{di_a}{dt} + \frac{d[\psi_r\cos(\theta)]}{dt} \\ v_b &= Ri_b + L\frac{di_b}{dt} + \frac{d[\psi_r\cos(\theta - 2\pi/3)]}{dt} \\ v_c &= Ri_c + L\frac{di_c}{dt} + \frac{d[\psi_r\cos(\theta + 2\pi/3)]}{dt} \end{aligned} \tag{2.4}$$

which yield,

$$\begin{aligned} v_a &= Ri_a + Lpi_a - \psi_r\omega\sin(\theta) \\ v_b &= Ri_b + Lpi_b - \psi_r\omega\sin(\theta - 2\pi/3) \\ v_c &= Ri_c + Lpi_c - \psi_r\omega\sin(\theta + 2\pi/3) \end{aligned} \tag{2.5}$$

where

- p is the differential operator, $p = d/dt$
- ω is the rotor speed, $\omega = d\theta/dt$

All the above formulation can be factored into the stator flux linkage space vector by defining it as:

$$\overline{\psi} = L\overline{i} + \psi_r e^{j\theta} \tag{2.6}$$

thus:

$$\overline{v} = R\overline{i} + L\frac{d\overline{i}}{dt} + \frac{d(\psi_r e^{j\theta})}{dt} \tag{2.7}$$

In the stationary frame of reference fixed to the stator, the three space vectors defined above can be expressed in terms of their real and imaginary components as:

$$\begin{aligned} \overline{v} &= v_D + jv_Q \\ \overline{i} &= i_D + ji_Q \\ \overline{\psi} &= \psi_D + j\psi_Q \end{aligned}$$

As we can see from the equations, the state-space model of a PMSM constitutes a highly coupled and nonlinear dynamic system. The nonlinearity can be significantly simplified through the use of the *dq* transformation, where the target reference frame is fixed to the rotor. We shall see the effect of this transformation in Section 3.5.

2.4 Approaches to Position and Speed Sensorless Operation

Most of the techniques are based on the voltage equations of the PMSM and the information of the terminal quantities, such as line voltage and phase current. Using this information, the rotor angle and speed are estimated directly or indirectly. In addition, there are waveform detection methods, which attempt to identify specific events, such as peaks or zero crossings in the electrical waveforms that are the result of the speed voltage of the motor. They have been successfully demonstrated for commutation needs and may be implemented using inexpensive electronics, which is advantageous from manufacturing viewpoint. However, they are not as accurate as possible because they do not utilize all the information present in the waveform.

French and Acarnley [10] have presented an exhaustive review of the various techniques proposed for sensorless operation of the PMSM. Their work forms the basis of what follows.

2.4.1 Back Electromotive Force

Back electromotive force (EMF) information can be used in brushless DC drive systems, where at any one time only two of the three phases are conducting. The direct back EMF detection method uses measurements of the instantaneous voltage across the third, non-conducting, phase. If the drive is designed such that the non-conducting phase current reduces rapidly, then the back EMF can be measured directly across the non-conducting phase. The zero crossing of the phase voltage can then be used to generate commutation data. Iizuka, Uzuhashi, Kano, Endo, and Mohri [16] have explained the principles of this approach, while Bahlmann [3] has given details of a commercially available integrated circuit which operates on these principles.

The predominant problem with the EMF approach to position estimation is that at low speeds the EMF approaches zero. This is so because the amplitude of the back EMF is proportional to speed: at zero speed, the magnets do not induce any voltage and, also, the voltage and current signals are quite noisy because of the Pulsewidth Modulation (PWM) [14] operation of the power stage. In the low-speed range, the voltage on the motor terminal can hardly be detected because of the small back-EMF

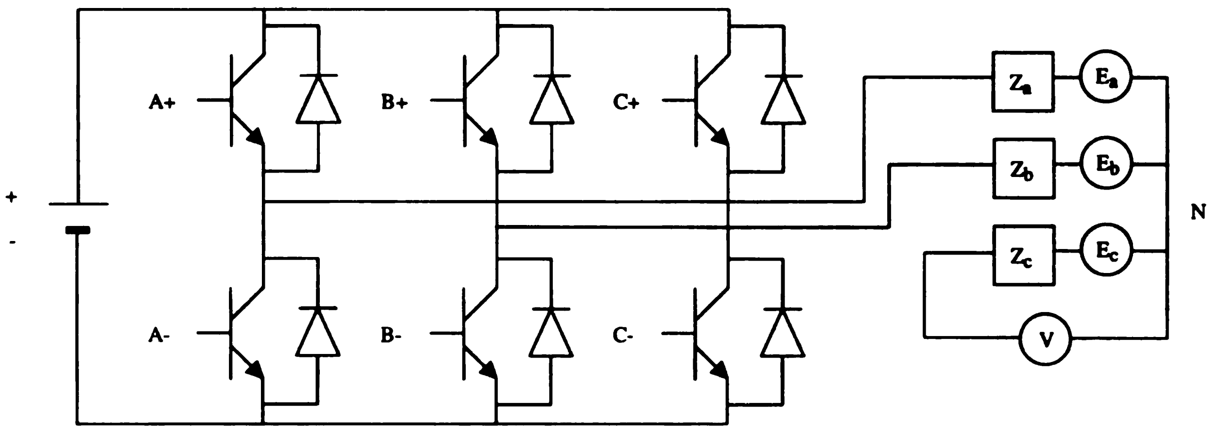

Figure 2.4. The back EMF method.

of the motor and the system noise. Considering the precision of the terminal voltage in PWM operation and the system noise produced by the nonlinear characteristics of the switching devices, the controllable lower speed range of the conventional sensorless drives is generally limited to the value of around 100 RPM.

Another problem to be considered is the starting capability. At standstill, the EMF is zero, so, some special starting algorithm or initial position detection algorithm must be used. Therefore, the scheme is restricted to applications where low-speed performance is not important, and where an alternative open-loop excitation scheme is acceptable to start the motor from standstill.

2.4.2 Excitation Monitoring

Excitation monitoring schemes involve monitoring the conduction paths of the current through the inverter. One method is to monitor the conduction state of the inverter's anti-parallel free-wheeling diodes. Ogasawara and Akagi [28] have done pioneering work with this method. In a brushless DC motor, at any instant, only two phases are conducting, with commutation occurring every 60 electrical degrees. The system operates by chopping one switching signal and leaving the other device

on continuously. The back EMF voltage of the non-switching phase is then measured and when this voltage crosses zero, a commutation position is detected. Like other EMF methods, this approach suffers from poor resolution at low speeds.

Another method, proposed for sinusoidally excited motors, by Arefeen, Ehsani and Lipo [2], monitors the zero crossings of the phase currents with the aid of a modified switching technique. The system estimates when the zero crossing point will occur, switches off the relevant phase, and then monitors the induced voltages in the open phase. The induced voltages can be used to accurately predict the zero crossing. The scheme is a variation on a scheme, proposed for reluctance motors, by Ehsani and Husain [8], and relies on the motor having a high degree of saliency.

2.4.3 Motor Modification

Motor modifications can ease the task of obtaining position information by using embedded search coils in the stator. For example, one scheme proposed by Binns, Al-Aubidy, and Simmin [4], uses three embedded search coils on the stator teeth. One is excited with a high-frequency low-voltage signal; the voltages induced by mutual effects in the two search coils are then processed to produce a position signal. The motion-induced effects are cancelled out with the aid of the two receiving sensors.

2.4.4 Magnetic Saliency

Variable inductance has also been proposed as a technique for position detection. One approach, proposed by Acarnley, Hill, and Hooper [1], is to monitor the phase current waveform, because the rate of change of current is a function of the incremental inductance of the phase circuit, and since phase inductance is a function of electrical position, the rotor position can be estimated.

Kulkarni and Ehsani [23] have proposed a related technique, with the phase inductance being calculated in real time. By measuring the phase currents and voltages, and then using a lookup table of position-inductance data, the position can be determined.

If the switching frequency is high, for instance greater than $10kHz$, then the variation of inductance with the rotor position can be neglected during one switching period. With this assumption, the following instantaneous voltage equation can be used for the phase a of an IPM:

$$v_a = Ri_a + Lpi_a + E_a \tag{2.8}$$

where, as discussed in Section 2.3,

$$L = L_s - M$$

For the phase a of an IPM,

$$L_s = L_{sl} + L_{sm} + L_g \cos(2\theta)$$

and

$$M = -\frac{1}{2}L_{sm} + L_g \cos(2\theta - 2\pi/3)$$

With the assumption that the motor back EMF E_a remains constant during a switching period, the instantaneous value of E_a is evaluated with the knowledge of the previous two instants:

$$\begin{aligned} E_a &= \psi_r \omega \\ &= \psi_r \frac{d\theta}{dt} \\ &= \psi_r \frac{\theta_i - \theta_j}{t_i - t_j} \end{aligned}$$

where θ_i and θ_j are the positions at two instants of time t_i and t_j, respectively.

Rewriting (2.8),

$$L = \frac{v_a - Ri_a - E_a}{pi_a}$$

where,

$$pi_a = \frac{i(j) - i(k)}{t(j) - t(k)}$$

The calculated phase inductance is then used to estimate the rotor position, using a set of stored data that relates the phase inductance with the rotor position.

Harris and Lang [13] have proposed to inject diagnostic voltage pulses into the non-conducting phase. The resulting currents are evaluated to measure the phase inductances. From these inductances, instantaneous motor position is estimated. Ehsani and Husain [8] suggested an alternative analog phase inductance method, which monitors the mutually induced voltages in unused adjacent phases. The induced voltages can be used to estimate inductance and hence position. The disadvantage of signal injection is thus avoided.

The variable inductance approach has the limitation that it works only with anisotropic rotors and also when the variation of the inductance with the rotor position is both sufficient and accurately known. Its principal advantage is that the zero speed is handled more easily with this method than the EMF approach. As with other methods described, errors can occur if assumed values of motor parameters are incorrect, for example if the resistance is inaccurate, due to thermal effects, then as the current increases, the error in estimated position also increases.

2.4.5 Observers

One way to extract all the required information is to model the dynamics of the motor, drive this motor model with the same input as is used to drive the real motor, and somehow ensure that errors between the modeled motor and real motor are minimized.

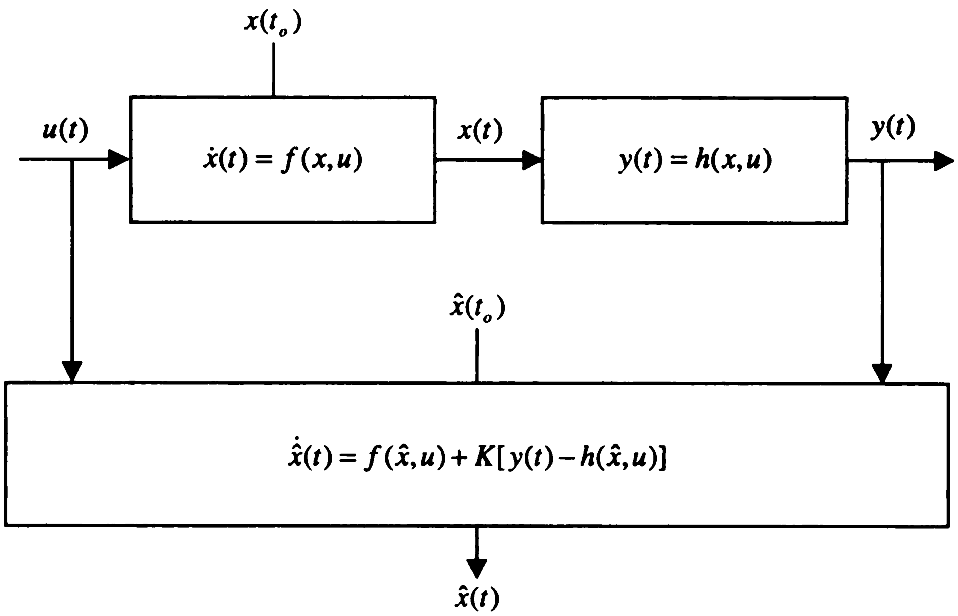

Figure 2.5. The structure of an observer.

If this can be done, the states of the modeled motor will effectively summarize all the information in the waveforms up to the present time, and the model will accurately reflect the behavior of the real motor. A state observer extends this idea. Here, an output is defined as a function of the states, and this output is compared with the equivalent measured output of the real motor. Any error between the two signals is then used to correct the state trajectory of the observer. This processing is shown in Figure 2.5.

An observer is often implemented to reconstruct the inaccessible states in a system. It is driven by the available system inputs and outputs and it may be implemented using hardware or software.

The models for PMSMs are nonlinear, and estimation theory for nonlinear systems is not as well developed as the wealth of knowledge available for linear systems such as DC motors.

Linear Observers

Kim and Sul [21] proposed a Luenberger observer based technique where the rotor angle and the motor speed information are obtained by transforming the terminal voltages to the stationary reference frame. A linear observer is built assuming that the motor speed is nearly constant during the processing time. An angle compensation algorithm to reject the system noise is also utilized. The method uses the resistance and inductance parameters to perform the calculations. Any change in these parameters causes the estimations to detune.

Nonlinear Observers

Observer-based systems allow the controller to access states which are not directly measurable. For example, flux linkage is not a directly measurable quantity. So, to obtain the flux linkage, the motor's phase voltages and currents are measured. These can be used, together with the phase resistance, to obtain an estimate of flux-linkage via integration. One example of this approach applied to a surface mounted PMSM is described by Wu and Slemon [33], where EMF is estimated from the measurement of terminal voltages and stator currents and is then processed to produce the stator flux linkage space vector. The angle of this vector is used to produce the stator current command signals. The system also uses the rate of change of the flux linkage angle to obtain a speed signal. Consoli, Musumeci, Raciti and Testa [5] have proposed a similar technique for interior permanent magnet motor. Due to the integration process by which flux linkage is obtained, these techniques suffer from the effects of integrator drift, which can be compensated either by analog electronics or software techniques as proposed by Hu and Wu [15].

The systems described above use the open loop flux linkage estimated values. In an observer system, this quantity is used to estimate a measurable quantity so that the system can update its motor model and, in the case of flux linkage estimation, compensate for integrator drift. With the availability of cost-effective powerful signal processors within the last few years, the sophistication of such observers schemes has increased.

One such technique, based on measuring both input and output parameters to generate a corrected model was presented by Matsui and Shigyo [25]. Their scheme uses the estimated flux linkage value, together with measured current, to estimate the phase voltage. The error between the measured and estimated voltages is used to correct the flux-linkage. This correction to flux-linkage translates into correction to the speed estimate. In this work, Matsui and Shigyo have presented the following equation for speed estimate:

$$\hat{\omega} = \frac{\hat{v}_q - (R + Lp)\hat{i}_q}{\psi_r + L\hat{i}_d} + (K_p \Delta\hat{v}_d + K_i \int \Delta\hat{v}_d dt) sign(\hat{\omega})$$

where,

$$\Delta\hat{v}_d = \hat{v}_d - \hat{v}_{dm}$$

and

$$\hat{v}_{dm} = (R + Lp)\hat{i}_d - L\hat{\omega}\hat{i}_q$$

The position estimate is found by integration of the speed estimate.

Extended Kalman Filter

Another class of observers uses the extended Kalman filter (EKF) to estimate both position and speed in real time. The work done by Dhaouadi, Mohan, and Norumu [7] illustrates the principles. The EKF is an optimal recursive estimation algorithm. It is based on the state-space model of the motor, together with a statistical description of the uncertainties involved. The uncertainties are modeled by three covariance matrices

for meas

The

where th

x_o and co

$x(t_o)$ and

The a

where $v($

It has a

The

estimate

The E

two steps

measurem

Step 1.

state covar

numerical

matrices: $P(t)$ for the system state vector, $Q(t)$ for the model uncertainty, and $R(t)$ for measurement uncertainty.

The EKF describes the system of interest by the nonlinear state space model:

$$\dot{x}(t) = f[x(t), u(t), t] + w(t)$$

where the initial state vector $x(t_o)$ is modeled as a Gaussian random vector with mean x_o and covariance P_o, while $w(t)$ is a zero-mean white Gaussian noise independent of $x(t_o)$ and with a covariance matrix $Q(t)$.

The available measurements are modeled as:

$$y(t_i) = h[x(t_i), t_i] + v(t_i)$$

where $v(t_i)$ is a zero-mean white Gaussian noise that is independent of $x(t_o)$ and $w(t)$. It has a covariance matrix $R(t_i)$.

The optimal state estimate, $\hat{x}(t)$, generated by the EKF is a minimum-variance estimate of $x(t)$, and is computed in a recursive manner.

The EKF has a predictor-corrector structure, and can be described in the following two steps, where the superscripts $-$ and $+$ refer to the time before and after the measurements have been processed.

Step 1. Prediction (from t_{i-1}^+ to t_i^-): The optimal state estimate, $\hat{x}(t)$, and the state covariance matrix, $P(t)$, are propagated from measurement time t_{i-1}^+ to t_i^-, by numerical integration of the following equations, where $t \in [t_{i-1}^+, t_i^-]$:

$$\begin{aligned} \dot{\hat{x}}(t) &= f[\hat{x}(t), u(t), t] \\ \dot{P}(t) &= F^T P(t) + P(t)F + Q(t) \end{aligned}$$

where

$$F = \frac{\partial f(\hat{x}, u, t)}{\partial \hat{x}}$$

evaluated at $\hat{x} = \hat{x}(t_{i-1}^{+})$.

Step 2. Correction (from t_i^- to t_i^+:) By comparing the measurement vector, $y(t_i)$, to the estimated one, $\hat{y}(t_i)$, a correction factor is obtained and is used to correct both the state vector and the covariance matrix. The following equations describe this step:

$$\begin{aligned} \hat{x}(t_i^+) &= \hat{x}(t_i^-) + K(t_i)\{y(t_i) - h[\hat{x}(t_i^-), t_i]\} \\ P(t_i^+) &= P(t_i^-) - K(t_i)HP(t_i^-) \end{aligned}$$

where the filter gain matrix $K(t_i)$ is defined as

$$K(t_i) = P(t_i^-)H^T[HP(t_i^-)H^T + R(t_i)]^{-1}$$

and

$$H = \frac{\partial h(\hat{x}, t_i)}{\partial \hat{x}}$$

evaluated at $\hat{x} = \hat{x}(t_i^-)$.

The algorithm is computationally intensive, and the accuracy also depends on the model parameters used. A critical part of the design is to use correct initial values for the various covariance matrices. These have important effects on the filter stability and convergence times. In principle, these need to be obtained by considering the stochastic properties of the corresponding noises. However since these are usually not known, trial-and-error is used for the initial estimates of the elements of these matrices to get the best tradeoff between filter stability and convergence time.

Implementing the algorithm requires powerful signal processing technology. Also, there is no apriori performance or stability guarantee, i.e., in general if incorrect initial values are used, the EKF algorithm may not converge to the correct values.

2.5 High-Gain Observers

High-gain observers have played a pivotal role in this dissertation. They have been used in estimating derivatives in the proposed scheme and also in proving stability of the position and speed dynamics. This section reviews the concept and provides references for further investigation.

The use of high-gain observers in robust estimation of output derivatives was first introduced by Esfandiari and Khalil [9], and since then has been used by many other researchers [20]. To illustrate a simple version of this concept [6], consider the case of a single-input, single-output nonlinear system which has a uniform relative degree [17] equal to the dimension of the state vector. Such a system has no zero dynamics [19] and can be transformed into the normal form:

$$\begin{aligned} \dot{x} &= Ax + B[a(x)u + b(x)] \\ y &= Cx \end{aligned}$$

where,

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \cdots & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & \cdots & \cdots & 1 & 0 \\ 0 & 0 & \cdots & \cdots & 0 & 1 \\ 0 & 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}_{n\times n}, \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}_{n\times 1}$$

$$C = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}_{1\times n}$$

and $a(x) \neq 0$.

Suppose that there exists a state feedback stabilizing control $u = \psi(x)$, where, $\psi(x)$ is a locally Lipschitz [19] function.

To estimate the state x, we use the observer,

$$\dot{\hat{x}} = A\hat{x} + B[a_o(\hat{x})\psi(\hat{x}) + b_o(\hat{x})] + H(y - C\hat{x}) \tag{2.9}$$

where $a_o(x)$ and $b_o(x)$, the nominal models of the nonlinear functions $a(x)$ and $b(x)$, respectively, and the $n \times 1$ matrix H is the observer gain. The estimation error, $e = x - \hat{x}$, satisfies the equation

$$\dot{e} = (A - HC)e + B\Delta(x, e, t)$$

where

$$\Delta(.) = [a(x) - a_o(\hat{x})]\psi(\hat{x}) + b(x) - b_o(\hat{x})$$

The observer gain, H, is taken as

$$H^T = \begin{bmatrix} \frac{\alpha_1}{\epsilon} & \frac{\alpha_2}{\epsilon^2} & \cdots & \cdots & \frac{\alpha_n}{\epsilon^n} \end{bmatrix}$$

where ϵ is a small positive parameter and the positive constants α_i are chosen such that the roots of

$$s^n + \alpha_1 s^{n-1} + \cdots + \alpha_{n-1} s + \alpha_n = 0 \tag{2.10}$$

have negative real parts. This choice of H assigns the eigenvalues of $(A - HC)$ at $1/\epsilon$ times the roots of (2.10). Using singular perturbation analysis, it is shown in [9] that the estimation error will decay to $O(\epsilon)$ values after a short transient period of the form $[0, T_1(\epsilon)]$ where $\lim_{\epsilon \to 0} T_1(\epsilon) = 0$.

The observer equation (2.9) is nonlinear due to the terms $a_o\psi$ and b_o. Choosing the nominal functions $a_o(x)$ and $b_o(x)$ to be zero results in the linear form of the observer,

$$\dot{\hat{x}} = A\hat{x} + H(y - C\hat{x})$$

It is important to notice that this linear version of the high-gain observer is an approximate differentiator. This can be seen by examining the transfer function from y to $\hat{x}$, given by:

$$G(s) = (sI - A + HC)^{-1}H$$

For n=2, $G(s)$ is given by

$$G(s) = \frac{1}{\epsilon^2 s^2 + \alpha_1 \epsilon s + \alpha_2} \begin{bmatrix} \epsilon s \alpha_1 + \alpha_2 \\ \alpha_2 s \end{bmatrix}$$

It is clear that $G_2(s)$, the transfer function from y to x_2, approaches s as ϵ tends to zero, which shows that $\hat{x}_2$ approximates the derivative $\dot{y}$.

CHAPTER 3

The Proposed Method

3.1 Introduction

In this chapter, the specifics of the dissertation are being presented. It begins with the motivation for such a scheme. It continues with the problem statement, and culminates in the derivation of the proposed estimation scheme. The last section presents the detailed derivations carried out.

3.2 Motivation

In the previous chapters, we have seen that the PMSM has emerged as an actuator of choice in servo systems. It owes its niche to its desirable characteristics as well as the latest advancements in the areas of magnetic materials, semiconductor switches and control strategies.

Also the need for position and speed sensors for an effective implementation of the vector control techniques has been described. One such technique commonly used to control the speed of a PMSM is shown in Figure 3.1. Here, the actual speed, ω, is measured and is fed back. The difference between the reference speed, ω_r, and the actual speed, ω, is calculated. The resulting speed error, $\Delta\omega$, is applied as input to a

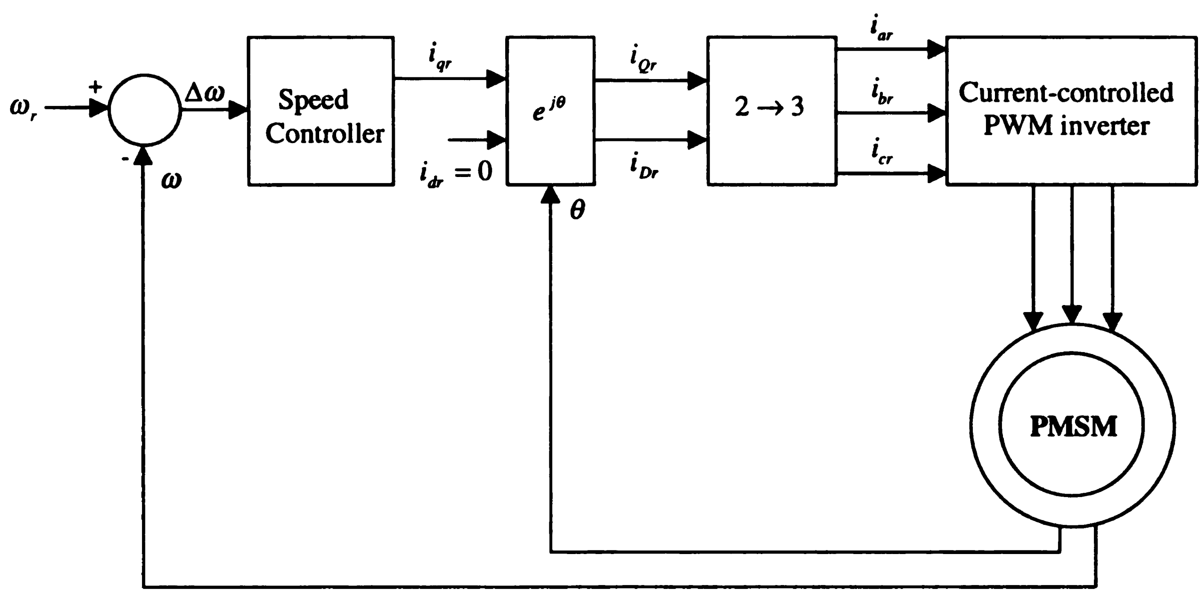

Figure 3.1. Traditional vector control of PMSM.

PI controller. The output of the PI controller is i_{qr}, the torque producing component of the stator reference current, transformed to the rotor frame of reference. The other component, i_{dr}, is the flux producing component. Its reference value is kept at zero, unless the motor is to be driven at a speed which exceeds the base value, determined by the DC bus voltage. In that case, the value of i_{dr} is set to a negative number. The details of this mode of operation will be given in Section 5.3.

The role of position sensing comes into the picture at this point: the reference values i_{qr} and i_{dr} are transformed into the stator frame of reference currents i_{ar}, i_{br}, and i_{cr}, using the measured position θ. A current controlled PWM inverter is used to ensure that the actual currents established in the PMSM follow the reference currents.

Since the primary focus of this dissertation is the estimation of position and speed, we shall limit the control part to using the above classical PI controller scheme throughout this work.

In the last chapter, the disadvantages associated with the use of mechanical sensors have been highlighted, most important of which are cost and reliability. Hence the move towards eliminating these sensors.

The techniques proposed by most researchers rely on using electrical or magnetic quantities that vary with position. Back EMF-based schemes have proved to be popular, but they exhibit uncertainties at low speed. Also, they have been mostly used with the motors with trapezoidal EMF, the so-called brushless DC motors, hence are restricted in their application.

Schemes that exploit inductance variation avoid the low-speed problem associated with the back EMF-based techniques. However, these schemes are effective only when there is large and accurately known variation of phase inductance with position, thus limiting the scope to motors with interior permanent magnets and their attendant saliency.

Schemes based on motor modification suffer from a similar drawback: they cannot be applied to off-the-shelf motors, or to the ones which are already being used in industry.

The flux linkage based methods have the advantage that they could be used in both sinusoidal and rectangular types of motors, although most proposed schemes have relied on open loop flux estimations and thus suffer from integrator drift.

The above synopsis highlights the importance of techniques based on observers. Many schemes along these lines have been proposed, as discussed in Chapter 2. The proposed idea builds upon the work done with such schemes.

3.3 Assumptions

In this work, the following assumptions are made regarding the PMSM:

- the rotor has surface mounted permanent magnets;
- the induced EMF is sinusoidal;

- magnetic saturation is neglected;
- stator windings are star connected.

3.4 Problem Statement

The primary objective of this dissertation is to formulate a scheme which estimates the position of the rotor, with the knowledge of only three motor parameters: resistance R, inductance L, and rotor flux linkage ψ_r. The input to the scheme must consist of only the terminal quantities: voltages and currents. The scheme must not rely on rotor saliency or any physical modification.

It must be able to correct errors in the position estimate. These errors might result from a lack of knowledge of the initial rotor position, or might arise from any other causes.

The secondary objective is, that under the same constraints, to come up with the speed estimate and a formulation to correct it, should this estimate be erroneous.

Once such scheme is formulated, it must be validated using numerical simulations and experimental implementation. Mathematical analysis must also be performed to prove stability of the position and speed dynamics.

3.5 The Proposed Scheme

This scheme was inspired by the work done by Matsui and Shigyo [25], as described in Section 2.4.5. The proposed scheme takes the approach of using the difference of current derivatives to estimate position and speed.

The motivation for the use of current derivatives to do this estimation comes from the work of Khalil, Strangas and Miller [18]. They use a high-gain observer to approximate the derivative of $\hat{i}_q$, the torque producing component of stator current,

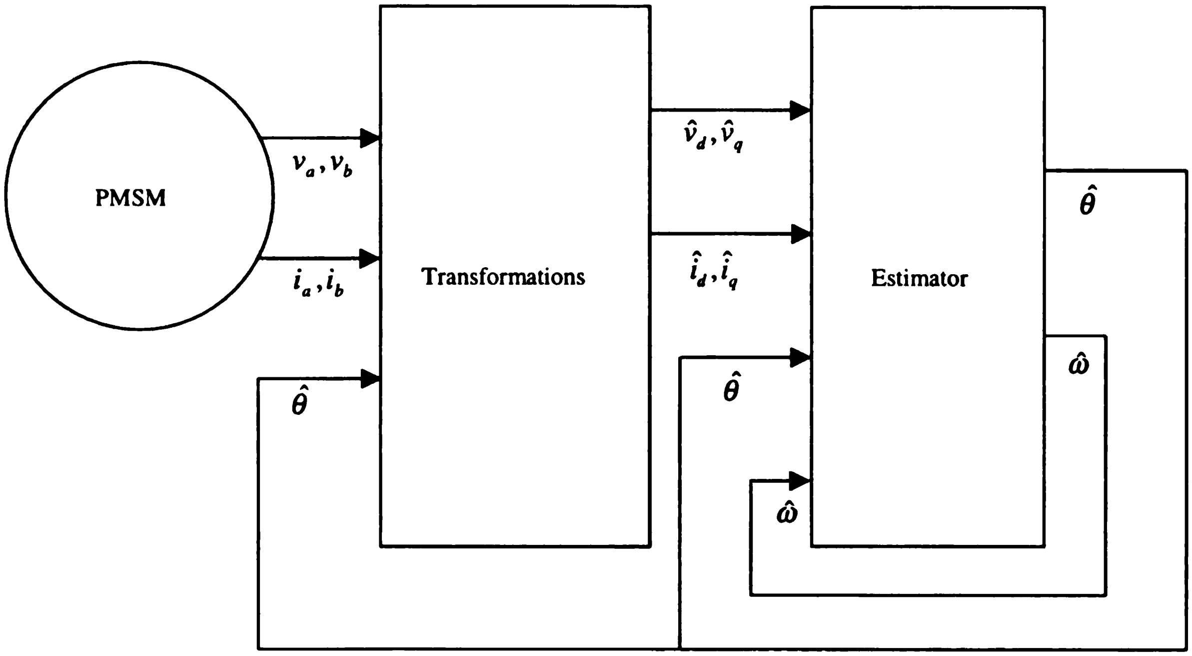

Figure 3.2. Overview of the proposed scheme.

transformed to the estimate of the rotor flux. This derivative is then used to estimate the speed of an induction motor.

The underlying steps in the proposed scheme are:

- measure the motor currents and voltages;
- transform these variables to the rotor frame of reference using $\hat{\theta}$, the estimate of θ;
- calculate the derivatives of each of the currents $\hat{\imath}_d$ and $\hat{\imath}_q$, two different ways;
- use the error between the current derivatives to drive the observer.

Figure 3.2 gives an overview of the proposed scheme. The details of the scheme follow.

The voltage model of the motor, as given in Equation (2.5) is the starting point.

The equation is rewritten here, for convenience:

$$\begin{aligned}
v_a &= Ri_a + Lpi_a - \psi_r\omega\sin(\theta) \\
v_b &= Ri_b + Lpi_b - \psi_r\omega\sin(\theta - 2\pi/3) \\
v_c &= Ri_c + Lpi_c - \psi_r\omega\sin(\theta + 2\pi/3)
\end{aligned} \tag{3.1}$$

If the rotor position θ is known, the following matrix transforms the variables of the stator reference frame to the one fixed to the rotor:

$$\begin{bmatrix} f_d \\ f_q \end{bmatrix} = \frac{2}{3}\begin{bmatrix} \cos\theta & \cos(\theta - 2\pi/3) & \cos(\theta + 2\pi/3) \\ -\sin\theta & -\sin(\theta - 2\pi/3) & -\sin(\theta + 2\pi/3) \end{bmatrix}\begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} \tag{3.2}$$

where, f represents either a voltage or a current.

Since we are developing a sensorless scheme, θ is unknown, we use its estimate $\hat{\theta}$. This modifies (3.2) to:

$$\begin{bmatrix} \hat{f}_d \\ \hat{f}_q \end{bmatrix} = \frac{2}{3}\begin{bmatrix} \cos\hat{\theta} & \cos(\hat{\theta} - 2\pi/3) & \cos(\hat{\theta} + 2\pi/3) \\ -\sin\hat{\theta} & -\sin(\hat{\theta} - 2\pi/3) & -\sin(\hat{\theta} + 2\pi/3) \end{bmatrix}\begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix} \tag{3.3}$$

Carrying out the transformation, (3.1) and (3.2) yield[1]:

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} R + Lp & -L\omega \\ L\omega & R + Lp \end{bmatrix}\begin{bmatrix} i_d \\ i_q \end{bmatrix} + \psi_r\omega\begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.4}$$

[1] Equations (3.4) and (3.5) are derived in Section 3.7.

while, (3.1) and (3.3) result in:

$$\begin{bmatrix} \hat{v}_d \\ \hat{v}_q \end{bmatrix} = \begin{bmatrix} R+Lp & -L\hat{\omega} \\ L\hat{\omega} & R+Lp \end{bmatrix} \begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} + \psi_r \omega \begin{bmatrix} -\sin\Delta\theta \\ \cos\Delta\theta \end{bmatrix} \tag{3.5}$$

where, ω is the actual speed, while $\hat{\omega}$ is its estimate, and

$$\Delta\theta = \theta - \hat{\theta} \tag{3.6}$$

Rearranging (3.5), we get:

$$p\begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} = \frac{1}{L}\left\{ \begin{bmatrix} \hat{v}_d \\ \hat{v}_q \end{bmatrix} - \begin{bmatrix} R & -L\hat{\omega} \\ L\hat{\omega} & R \end{bmatrix} \begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} - \psi_r \omega \begin{bmatrix} -\sin\Delta\theta \\ \cos\Delta\theta \end{bmatrix} \right\} \tag{3.7}$$

The right-hand side of (3.7) contains the variables we are estimating: ω and θ. The variables on the left-hand side, $p\hat{i}_d$ and $p\hat{i}_q$, can be calculated by finding the derivatives of the signals known to us: $\hat{i}_d$ and $\hat{i}_q$. High-gain observers [9] will be used to find these derivatives.

At this point, the objective is to come up with expressions that relate the unknown with the known: the errors in our estimates $\theta - \hat{\theta}$ and $\omega - \hat{\omega}$ with errors in computable signals. To this end, we define a new set of variables, replicating (3.7) but assuming $\Delta\theta = 0$ and $\hat{\omega} = \omega$:

$$p\begin{bmatrix} \hat{i}_{dm} \\ \hat{i}_{qm} \end{bmatrix} = \frac{1}{L}\left\{ \begin{bmatrix} \hat{v}_d \\ \hat{v}_q \end{bmatrix} - \begin{bmatrix} R & -L\hat{\omega} \\ L\hat{\omega} & R \end{bmatrix} \begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} - \psi_r \hat{\omega} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \tag{3.8}$$

Further defining:

$$\begin{bmatrix} \Delta p\hat{i}_d \\ \Delta p\hat{i}_q \end{bmatrix} = p\begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} - p\begin{bmatrix} \hat{i}_{dm} \\ \hat{i}_{qm} \end{bmatrix} \tag{3.9}$$

(3.7) - (3.8) yields the error in current derivatives:

$$\begin{bmatrix} \Delta p\hat{i}_d \\ \Delta p\hat{i}_q \end{bmatrix} = \frac{\psi_r}{L} \begin{bmatrix} \omega \sin \Delta\theta \\ -\omega \cos \Delta\theta + \hat{\omega} \end{bmatrix} \tag{3.10}$$

Assuming that $\Delta\theta$ is small, the following simplification is used:

$$\sin \Delta\theta \approx \Delta\theta$$

$$\cos \Delta\theta \approx 1$$

Thus,

$$\begin{bmatrix} \Delta p\hat{i}_d \\ \Delta p\hat{i}_q \end{bmatrix} = \frac{\psi_r}{L} \begin{bmatrix} \omega\Delta\theta \\ -\Delta\omega \end{bmatrix} \tag{3.11}$$

where,

$$\Delta\omega = \omega - \hat{\omega} \tag{3.12}$$

Equation (3.11) states that an error in the position estimate manifests itself in the signal $\Delta p\hat{i}_d$. Similarly, if there is an error in the speed estimate, it reflects itself in the signal $\Delta p\hat{i}_q$.

Rearranging (3.11), we get:

$$\begin{aligned} \Delta\omega &= -\frac{L}{\psi_r}\Delta p\hat{i}_q \\ \Delta\theta &= \frac{L}{\psi_r \omega}\Delta p\hat{i}_d \end{aligned}$$

Defining $k = L/\psi_r$ and substituting the definitions of $\Delta\omega$ and $\Delta\theta$ from (3.6) and (3.12), the above set of equations can be re-written as:

$$\omega = \hat{\omega} - k\Delta p\hat{i}_q \tag{3.13}$$

$$\theta = \hat{\theta} + k\frac{\Delta p\hat{i}_d}{\omega} \tag{3.14}$$

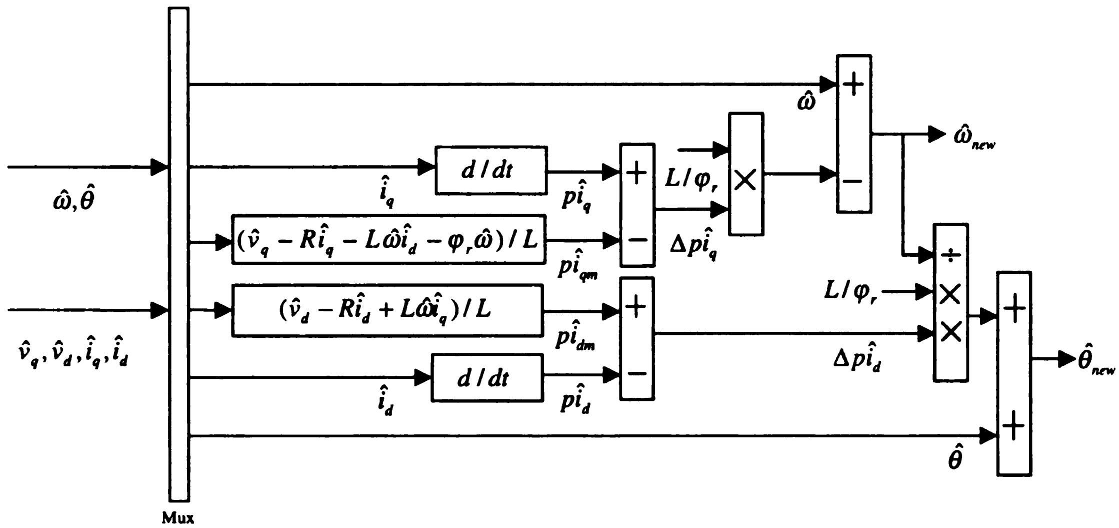

Figure 3.3. Details of the proposed scheme.

For the purpose of implementation, (3.13) and (3.14) can be expressed as:

$$\hat{\omega}_{new} = \hat{\omega} - k\Delta p\hat{i}_q \tag{3.15}$$

$$\hat{\theta}_{new} = \hat{\theta} + k\frac{\Delta p\hat{i}_d}{\hat{\omega}_{new}} \tag{3.16}$$

Figure 3.3 shows the above algorithm graphically.

3.6 Characteristics of the Proposed Scheme

There are several advantages to the proposed scheme. First, the scheme works in a closed-loop fashion meaning that it has an inherent correction mechanism. For instance, if there is an error in the position estimate, the scheme has a way to fix it.

Another advantage relates to computational burden. Since there are just two terms in each expression: the previous value of the estimate and the correction term, the scheme cuts down on the number of calculations. This contrasts with the more traditional observer equations, which have an estimation term, often replicating the plant dynamics, in addition to the correction term.

The third salient feature is that that the scheme does not estimate position by the commonly employed technique of integrating the speed estimate. Instead it rather uses the d-axis current derivatives, thus avoiding drift problems associated with the integration process.

Another advantage relates to the scope of the scheme: it does not rely on some special motor feature, for instance, saliency of poles. It can be used in the existing motors without any physical modification, such as placement of search coils. Also, it does not require knowledge of the mechanical load attached.

Finally, the scheme does not require the availability of the neutral point for voltage computation. Infact, it can be implemented with just two current sensors: the use of voltage sensors can be avoided using knowledge of the PWM pattern.

The major limitation of the scheme is that it requires knowledge of three motor parameters: resistance R, inductance L and rotor flux linkage ψ_r: parameter mismatch can influence the accuracy of the estimates. This will be further investigated in the next chapters.

3.7 Derivations

3.7.1 Derivation of Equation (3.4)

We begin the derivation with the general form of Equation (3.2):

$$v_{dqo} = M v_{abc} \tag{3.17}$$

In expanded form, this equation is written as:

$$\begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos\theta & \cos(\theta - 2\pi/3) & \cos(\theta + 2\pi/3) \\ -\sin\theta & -\sin(\theta - 2\pi/3) & -\sin(\theta + 2\pi/3) \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} \tag{3.18}$$

Here the third component v_o is not associated with the rotor reference frame. Instead it is related arithmetically to the *abc* variables, independent of θ. When the three-phase system is symmetrical and the voltages form a balanced three-phase set of *abc* sequence, the sum of the set is zero, hence the v_o variable is zero.

The derivation that follows also uses the relationship between voltages, currents and flux linkages in the *abc* frame of reference:

$$v_{abc} = Ri_{abc} + p\psi_{abc} \tag{3.19}$$

Similar to Equation (3.17), the flux linkages across the frames are related as:

$$\psi_{dqo} = M\psi_{abc} \tag{3.20}$$

This translates to:

$$\psi_{abc} = M^{-1}\psi_{dqo} \tag{3.21}$$

Putting Equations (3.17), (3.19), and (3.21) together, we have:

$$\begin{aligned} v_{dqo} &= Mv_{abc} \\ &= M\{Ri_{abc} + p\psi_{abc}\} \\ &= MRi_{abc} + Mp\psi_{abc} \\ &= Ri_{dqo} + Mp\{M^{-1}\psi_{dqo}\} \end{aligned}$$

$$
\begin{aligned}
&= Ri_{dqo} + M\{[p(M^{-1})]\psi_{dqo} + M^{-1}p\psi_{dqo}\} \\
&= Ri_{dqo} + M[p(M^{-1})]\psi_{dqo} + MM^{-1}p\psi_{dqo} \\
&= Ri_{dqo} + M[p(M^{-1})]\psi_{dqo} + p\psi_{dqo} \qquad (3.22)
\end{aligned}
$$

Further manipulation leads us to:

$$
M[p(M^{-1})] = \omega \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad (3.23)
$$

Equation (3.23) is derived in Section (3.7.2). Equations (3.22) and (3.23) yield the following result, written in expanded form:

$$
\begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} = R \begin{bmatrix} i_d \\ i_q \\ i_o \end{bmatrix} + \omega \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \psi_d \\ \psi_q \\ \psi_o \end{bmatrix} + p \begin{bmatrix} \psi_d \\ \psi_q \\ \psi_o \end{bmatrix} \qquad (3.24)
$$

This leads us to the relation:

$$
\begin{aligned}
v_d &= Ri_d - \omega\psi_q + p\psi_d \\
&= Ri_d - \omega L i_q + Lpi_d \\
&= (R + Lp)i_d - \omega L i_q
\end{aligned}
$$

where, we have used the definitions of ψ_d and ψ_q

$$
\begin{aligned}
\psi_d &= Li_d + \psi_r \\
\psi_q &= Li_q
\end{aligned}
$$

Similarly,

$$\begin{aligned} v_q &= Ri_q + p\psi_q + \omega\psi_d \\ &= Ri_q + Lpi_q + \omega L i_d + \omega\psi_r \\ &= (R + Lp)i_q + \omega(Li_d + \psi_r) \end{aligned}$$

This concludes the derivation of Equation (3.4).

3.7.2 Derivation of Equation (3.23)

In this section, the details leading to Equation (3.23) are being presented. We begin with the transformation matrix M:

$$M = \frac{2}{3}\begin{bmatrix} \cos\theta & \cos(\theta - 2\pi/3) & \cos(\theta + 2\pi/3) \\ -\sin\theta & -\sin(\theta - 2\pi/3) & -\sin(\theta + 2\pi/3) \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \tag{3.25}$$

This matrix has the inverse M^{-1}:

$$M^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 1 \\ \cos(\theta - 2\pi/3) & -\sin(\theta - 2\pi/3) & 1 \\ \cos(\theta + 2\pi/3) & -\sin(\theta + 2\pi/3) & 1 \end{bmatrix} \tag{3.26}$$

Taking the derivative of Equation (3.26), we get:

$$p[M^{-1}] = \omega\begin{bmatrix} -\sin\theta & -\cos\theta & 0 \\ -\sin(\theta - 2\pi/3) & -\cos(\theta - 2\pi/3) & 0 \\ -\sin(\theta + 2\pi/3) & -\cos(\theta + 2\pi/3) & 0 \end{bmatrix} \tag{3.27}$$

Multiplying Equations (3.25) and (3.27) together, we get:

$$Mp[M^{-1}] = \omega\frac{2}{3}\begin{bmatrix} M_{11} & M_{12} & 0 \\ M_{21} & M_{21} & 0 \\ M_{31} & M_{32} & 0 \end{bmatrix} \tag{3.28}$$

where, the elements of the matrix are written and manipulated below:

$$\begin{aligned}
M_{11} &= -\cos\theta\sin\theta - \cos(\theta - 2\pi/3)\sin(\theta - 2\pi/3) \\
&\quad -\cos(\theta + 2\pi/3)\sin(\theta + 2\pi/3) \\
&= 0 \\
M_{12} &= -\cos^2\theta - \cos^2(\theta - 2\pi/3) - \cos^2(\theta + 2\pi/3) \\
&= -\frac{3}{2} \\
M_{21} &= \sin^2\theta + \sin^2(\theta - 2\pi/3) + \sin^2(\theta + 2\pi/3) \\
&= \frac{3}{2} \\
M_{22} &= \sin\theta\cos\theta + \sin(\theta - 2\pi/3)\cos(\theta - 2\pi/3) \\
&\quad +\sin(\theta + 2\pi/3)\cos(\theta + 2\pi/3) \\
&= 0 \\
M_{31} &= -\frac{1}{2}\{\sin\theta + \sin(\theta - 2\pi/3) + \sin(\theta + 2\pi/3)\} \\
&= 0 \\
M_{32} &= -\frac{1}{2}\{\cos\theta + \cos(\theta - 2\pi/3) + \cos(\theta + 2\pi/3)\} \\
&= 0
\end{aligned}$$

Putting all the elements back into Equation (3.28), we get:

$$Mp[M^{-1}] = \omega\frac{2}{3}\begin{bmatrix} 0 & -\frac{3}{2} & 0 \\ \frac{3}{2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

i.e.,

$$Mp[M^{-1}] = \omega \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

This concludes the derivation of Equation (3.23).

3.7.3 Derivation of Equation (3.5)

The first first row of the matrix equation (3.5), is derived as follows:

$$\begin{aligned}
\hat{v}_d &= \frac{2}{3}\left[v_a \cos\hat{\theta} + v_b \cos(\hat{\theta} - 2\pi/3) + v_c \cos(\hat{\theta} + 2\pi/3)\right] \\
&= \frac{2}{3}[\{Ri_a + Lpi_a - \psi_r \omega \sin\theta\} \cos\hat{\theta} \\
&\quad + \{Ri_b + Lpi_b - \psi_r \omega \sin(\theta - 2\pi/3)\} \cos(\hat{\theta} - 2\pi/3) \\
&\quad + \{Ri_c + Lpi_c - \psi_r \omega \sin(\theta + 2\pi/3)\} \cos(\hat{\theta} + 2\pi/3)] \\
&= R[\frac{2}{3}\{i_a \cos\hat{\theta} + i_b \cos(\hat{\theta} - 2\pi/3) + i_c \cos(\hat{\theta} + 2\pi/3)\}] \\
&\quad + L[\frac{2}{3}\{p[i_a \cos\hat{\theta}] + p[i_b \cos(\hat{\theta} - 2\pi/3)] + p[i_c \cos(\hat{\theta} + 2\pi/3)]\}] \\
&\quad - \psi_r \omega[\frac{2}{3}\{\sin\theta \cos\hat{\theta} + \sin(\theta - 2\pi/3)\cos(\hat{\theta} - 2\pi/3) \\
&\quad + \sin(\theta + 2\pi/3)\cos(\hat{\theta} + 2\pi/3)\}] \\
&= R\hat{i}_d + L[\frac{2}{3}\{\cos\hat{\theta} pi_a + \cos(\hat{\theta} - 2\pi/3) pi_b + \cos(\hat{\theta} + 2\pi/3) pi_c\}] \\
&\quad + L[\frac{2}{3}\{-i_a \hat{\omega} \sin\hat{\theta} - i_b \hat{\omega} \sin(\hat{\theta} - 2\pi/3) - i_c \hat{\omega} \sin(\hat{\theta} + 2\pi/3)\}] \\
&\quad - \psi_r \omega[\frac{2}{3}\{\frac{3}{2}\sin(\theta - \hat{\theta})\}] \\
&= R\hat{i}_d + Lp\hat{i}_d - L\hat{\omega}\hat{i}_q - \psi_r \omega \sin(\theta - \hat{\theta}) \\
\hat{v}_d &= (R + Lp)\hat{i}_d - L\hat{\omega}\hat{i}_q - \psi_r \omega \sin\Delta\theta
\end{aligned}$$

This derivation makes use of the following trigonometric identity:

$$\sin x \cos y + \sin(x - \frac{2\pi}{3}) \cos(y - \frac{2\pi}{3}) + \sin(x + \frac{2\pi}{3}) \cos(y + \frac{2\pi}{3}) = \frac{3}{2} \sin(x - y)$$

Along similar lines, the second row of the matrix equation (3.5), is derived as follows:

$$\begin{aligned}
\hat{v}_q &= -\frac{2}{3} \left[v_a \sin\hat{\theta} + v_b \sin(\hat{\theta} - 2\pi/3) + v_c \sin(\hat{\theta} + 2\pi/3) \right] \\
&= -\frac{2}{3} [\{Ri_a + Lpi_a - \psi_r \omega \sin\theta\} \sin\hat{\theta} \\
&\quad + \{Ri_b + Lpi_b - \psi_r \omega \sin(\theta - 2\pi/3)\} \sin(\hat{\theta} - 2\pi/3) \\
&\quad + \{Ri_c + Lpi_c - \psi_r \omega \sin(\theta + 2\pi/3)\} \sin(\hat{\theta} + 2\pi/3)] \\
&= R[-\frac{2}{3}\{i_a \sin\hat{\theta} + i_b \sin(\hat{\theta} - 2\pi/3) + i_c \sin(\hat{\theta} + 2\pi/3)\}] \\
&\quad + L[-\frac{2}{3}\{p[i_a \sin\hat{\theta}] + p[i_b \sin(\hat{\theta} - 2\pi/3)] + p[i_c \sin(\hat{\theta} + 2\pi/3)]\}] \\
&\quad + \psi_r \omega[-\frac{2}{3}\{\sin\theta \sin\hat{\theta} + \sin(\theta - 2\pi/3) \sin(\hat{\theta} - 2\pi/3) \\
&\quad + \sin(\theta + 2\pi/3) \sin(\hat{\theta} + 2\pi/3)\}] \\
&= R\hat{i}_q + L[\frac{2}{3}\{\sin\hat{\theta} pi_a + \sin(\hat{\theta} - 2\pi/3) pi_b + \sin(\hat{\theta} + 2\pi/3) pi_c\}] \\
&\quad + L[\frac{2}{3}\{i_a \hat{\omega} \cos\hat{\theta} + i_b \hat{\omega} \cos(\hat{\theta} - 2\pi/3) + i_c \hat{\omega} \cos(\hat{\theta} + 2\pi/3)\}] \\
&\quad + \psi_r \omega[-\frac{2}{3}\{\frac{3}{2} \cos(\theta - \hat{\theta})\}] \\
&= R\hat{i}_q + Lp\hat{i}_q + L\hat{\omega}\hat{i}_d + \psi_r \omega \cos(\theta - \hat{\theta}) \\
\hat{v}_q &= (R + Lp)\hat{i}_q + L\hat{\omega}\hat{i}_d + \psi_r \omega \cos\Delta\theta
\end{aligned}$$

This derivation uses the following trigonometric identity:

$$\sin x \sin y + \sin(x - \frac{2\pi}{3}) \sin(y - \frac{2\pi}{3}) + \sin(x + \frac{2\pi}{3}) \sin(y + \frac{2\pi}{3}) = \frac{3}{2} \cos(x - y)$$

This concludes the derivation of Equation (3.5).

CHAPTER 4

Numerical Simulations

4.1 Introduction

In this chapter, results from the numerical simulations are presented. All the simulations were carried out with *Simulink* using the default variable-step ode45 (Dormand-Prince) algorithm. The maximum step size, relative tolerance and absolute tolerance were all set at 1e-3.

4.2 Simulation Results

Figure 4.1 sets the baseline for the results: it shows that the estimated position, $\hat{\theta}$, converges to the actual one, θ, in less than a tenth of a second. This is despite the fact that the estimate is initially off the actual by 179^o. This ability to converge despite such a wide initial estimation error, $(\theta - \hat{\theta})_o$, is a major strength of the proposed scheme. The speed estimate, $\hat{\omega}$, is working well too, with the estimation error, $\omega - \hat{\omega}$, converging to zero in less than a tenth of a second, as well. In Figure 4.2, the same simulation is carried out with the motor running at about one fifth the speed of the first one, and also with a load attached. No deterioration in performance is observed.

In both these cases, the speed estimation error is seen approaching a maximum of about 3 rad/sec during the initial transient. The simulation shown in Figure 4.3, investigates this error. Here, the initial position error is reduced to 90^o. It is observed that the speed estimation error is cut down significantly to about 0.5 rad/sec. This is expected because the estimated position $\hat{\theta}$ is the most critical element in the whole estimation scheme, as the transformation to the rotor frame of reference relies on the accuracy of $\hat{\theta}$. The speed estimate is very sensitive to the initial position error. For the rest of the simulations, the initial position error is kept at the 90^o level.

The next three simulations investigate the performance of the estimation scheme as the motor is rotated at reduced speeds. Figure 4.4 shows that the estimation scheme is working quite acceptably at speeds of 10, 1 and 0.1 RPM. The estimation converges much slowly as the motor speed drops beneath 1 RPM.

The next four simulations examine the impact of parameter mismatch on the performance of the estimation scheme. The parameter chosen is the resistance of the stator winding. This choice follows from the fact that the resistance is the most sensitive of all the parameters, increasing as the motor heats up. In these simulations, we continue starting the position estimate 90^o off the actual one. We are interested in exploring whether the position estimation error is still converging to zero or not. A load is applied so as to emphasize the impact of the mismatch: in the absence of a load, the motor currents drop to negligible values once the desired speed is reached, thus downplaying the effect of a parameter mismatch. The motor is turned at the low speeds of 10 and 1 RPM, thus further eliminating factors which mask the impact of parameter mismatch: at higher speeds, most of the estimation schemes work well, it is the lower speed interval where most stop converging.

Figures 4.7 and 4.8 show the results of the case when the value of the resistance used by the estimation algorithm is less than the actual. This corresponds to the situation where the motor has heated up. We see that the position estimate is still working as well as in the previous cases, while the speed estimate fails to converge to the correct value. The accuracy of the speed estimate is more important for the speed controller, which is not the main focus of the discussion here. Since transformation to the rotor frame of reference depends on the correct value of the position estimate, its convergence is critical for the estimator to work. The speed estimate is found to be about 10% more than the actual. Part of the problem can be attributed to the fact that the motor has a resistance value of 6-Ω per phase, which is higher than the usual 0.5-1.0Ω range. Thus a 10% mismatch figures much more prominently.

Figures 4.9 and 4.10 present the situation when the resistance value assumed by the estimator is 10% higher than the nominal. This corresponds to a colder than normal motor. The results are comparable to the ones obtained above: the position estimate shows agreement with the actual, while the speed estimate is found to be about 10% less.

This concludes the presentation of the simulation results. We have noticed that the estimation scheme has its strengths and weaknesses: as far the estimation of position, the proposed scheme performs quite well, with estimation error converging to zero even in the face of parameter mismatch, and in the presence of an initial error of as much as 179^o. Whereas, the speed estimate performs well when there is perfect parameter knowledge, but has problems when the assumed resistance value does not match the actual. We shall explain these observations through mathematical analysis in the next chapter.

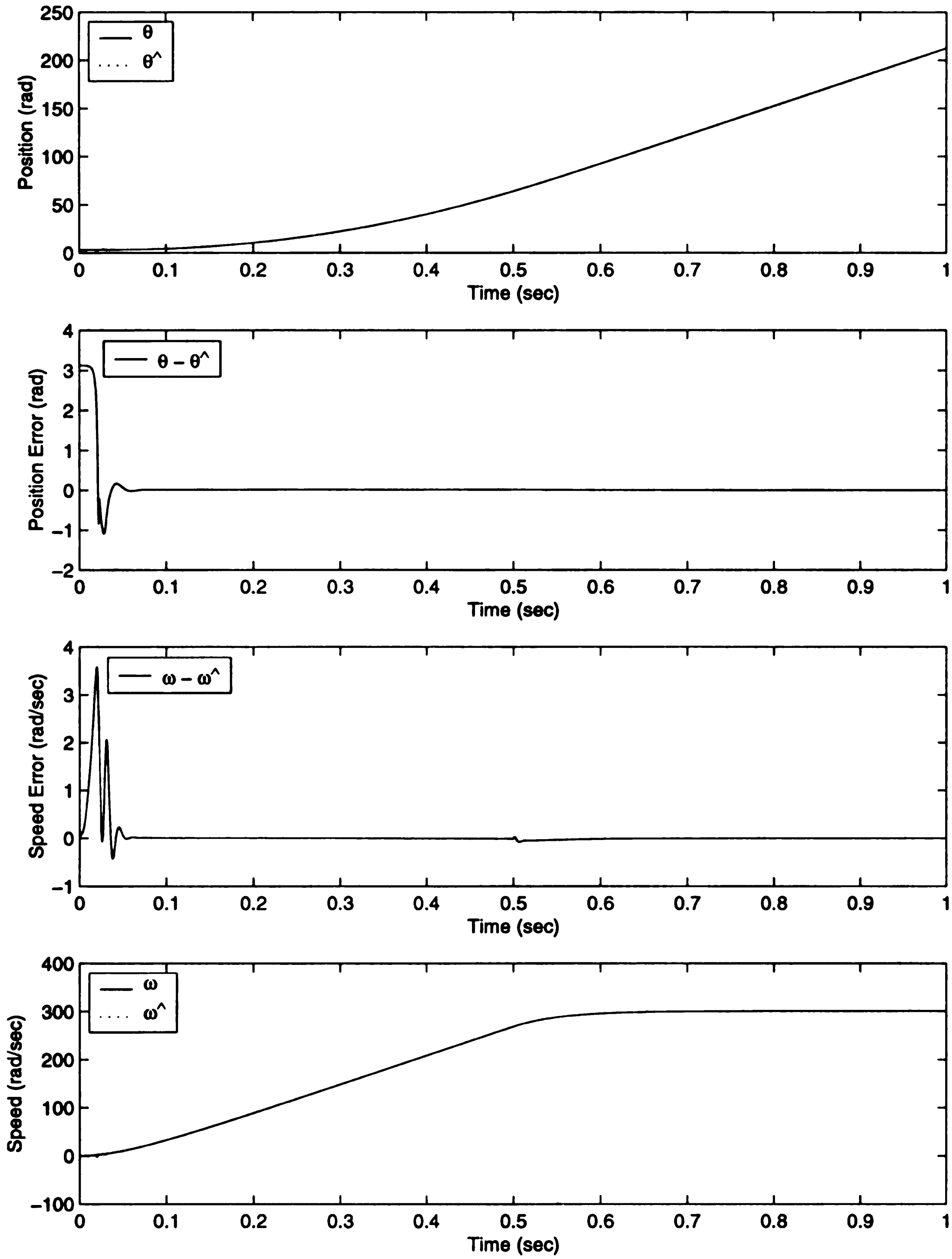

Figure 4.1. Simulation: $\Delta\theta_o$: 179^o, ref speed: 900 RPM.

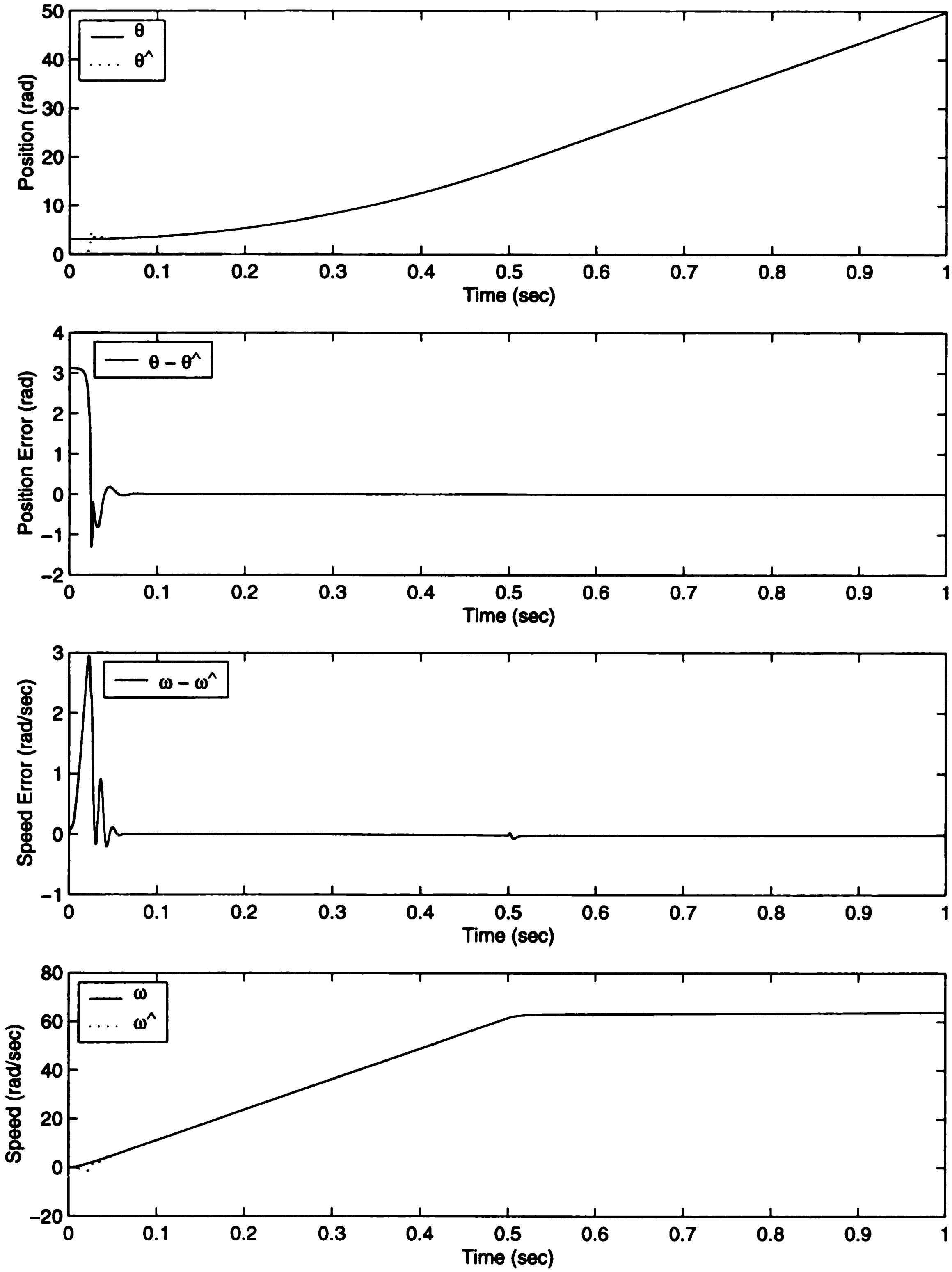

Figure 4.2. Simulation: $\Delta\theta_o$: 179^o, ref speed: 180 RPM.

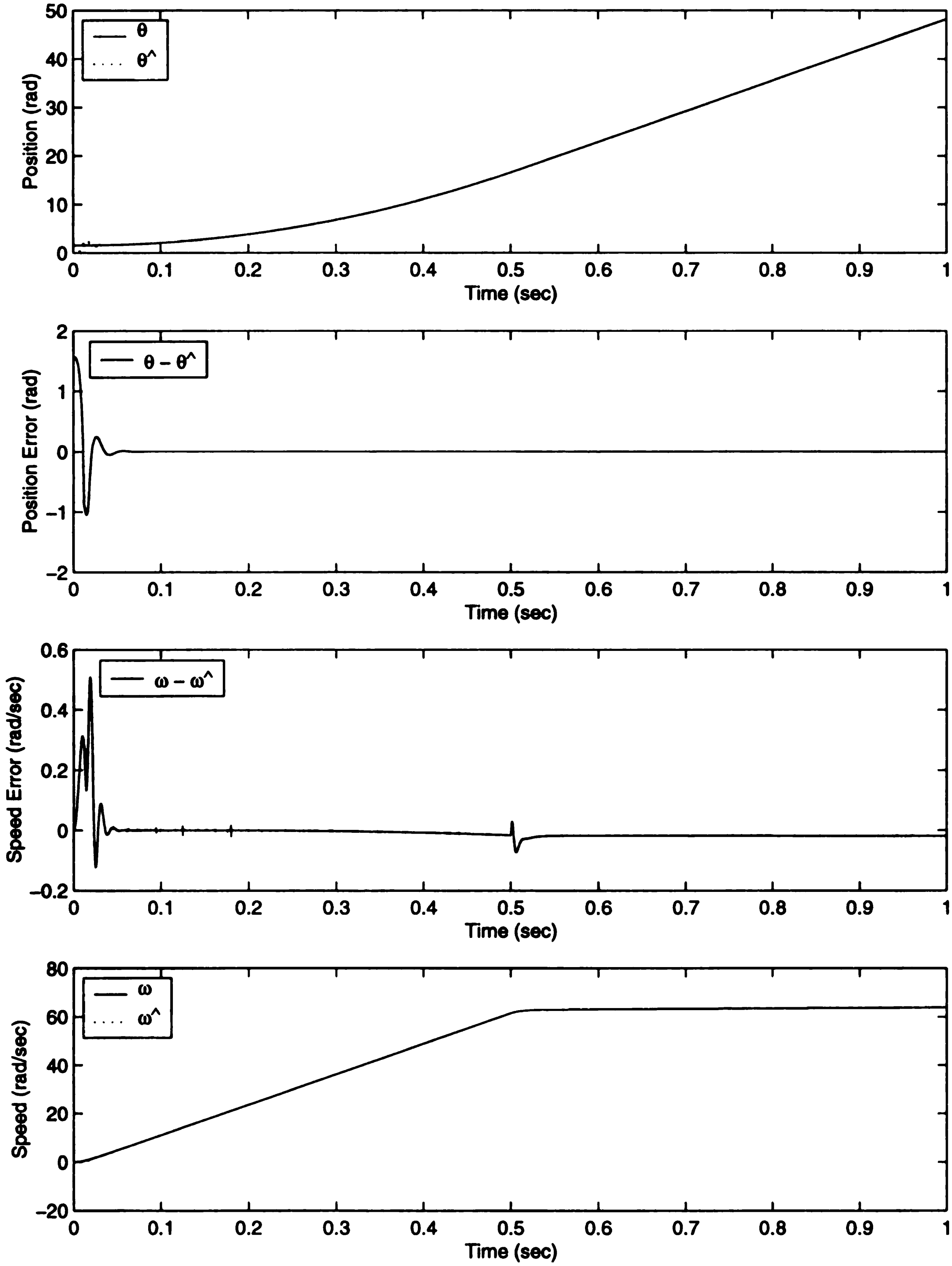

Figure 4.3. Simulation: $\Delta\theta_o$: 90^o, ref speed: 180 RPM.

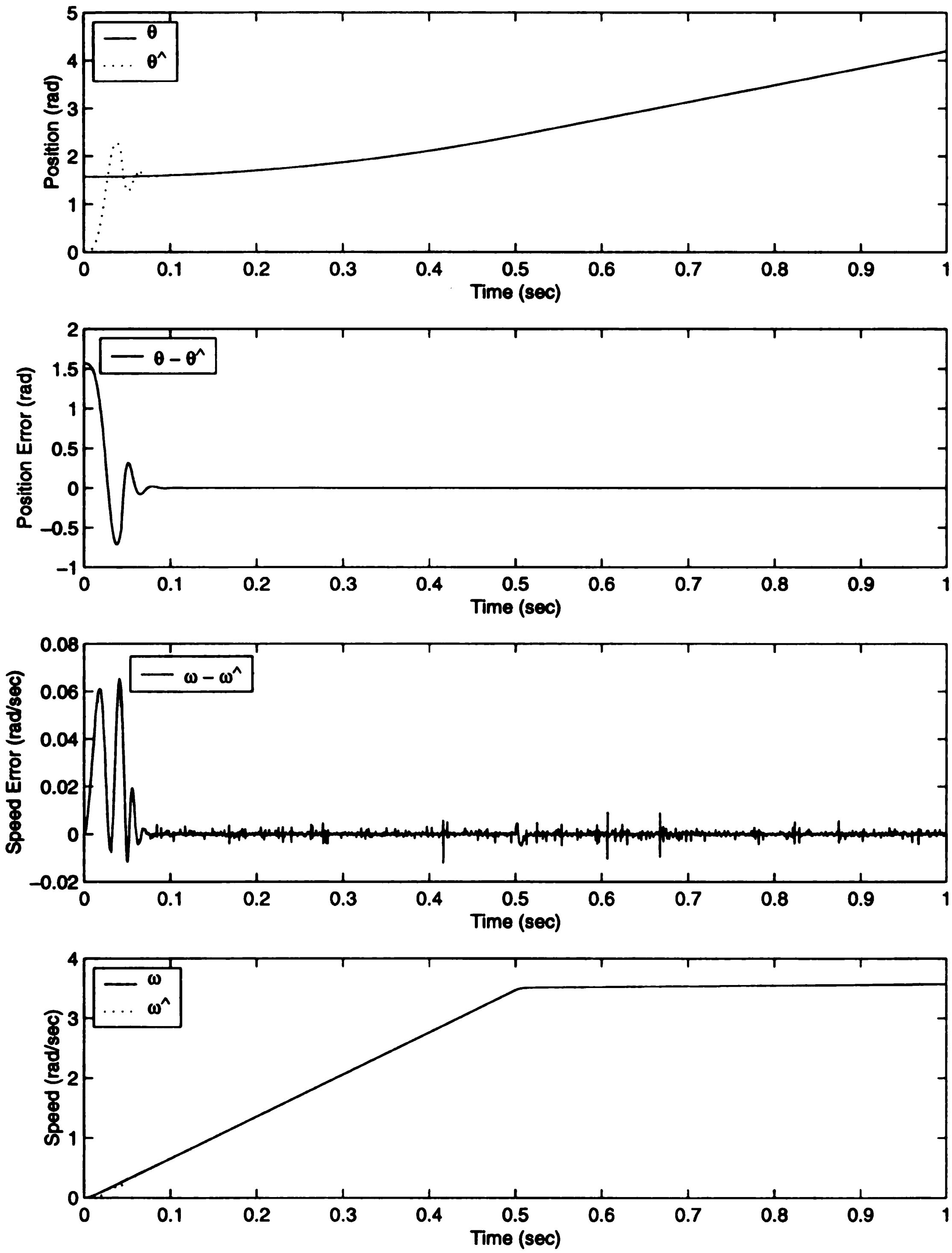

Figure 4.4. Simulation: $\Delta\theta_o : 90^o$, ref speed: 10 RPM.

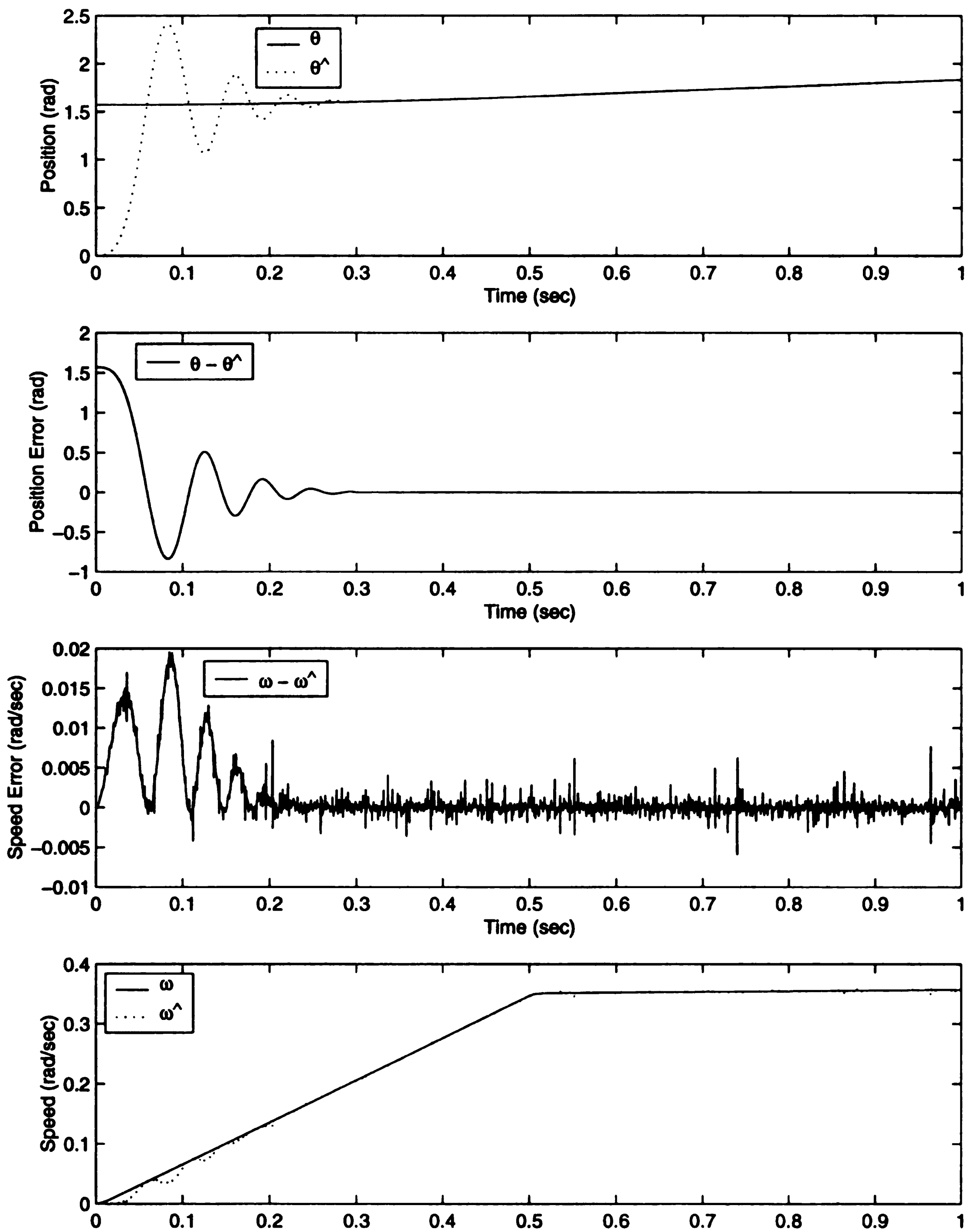

Figure 4.5. Simulation: $\Delta\theta_o$: 90^o, ref speed: 1 RPM.

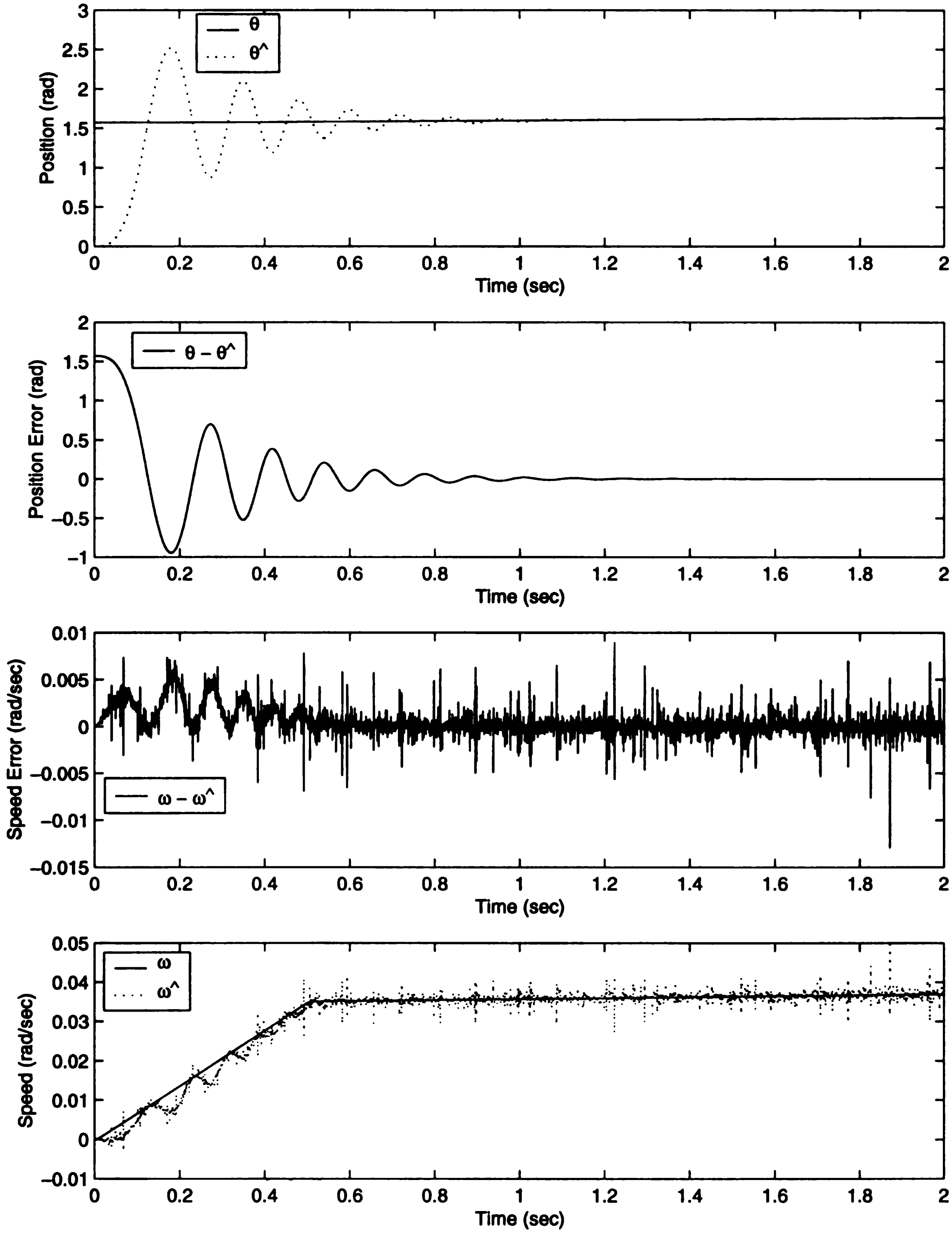

Figure 4.6. Simulation: $\Delta\theta_o$: 90^o, ref speed: 0.1 RPM.

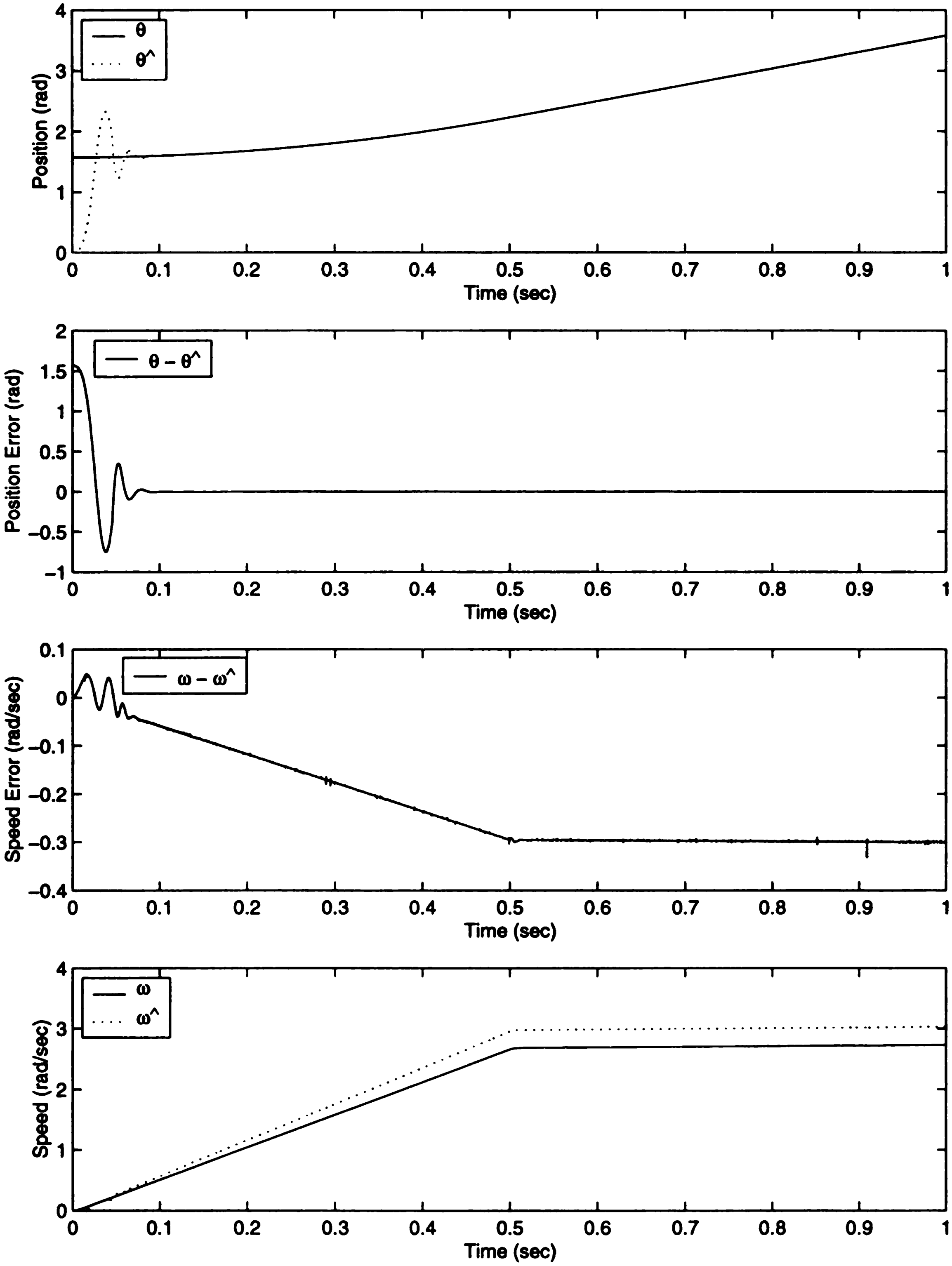

Figure 4.7. Simulation: $\Delta\theta_o : 90^o$, $\hat{R} : 0.9R$, ref speed: 10 RPM.

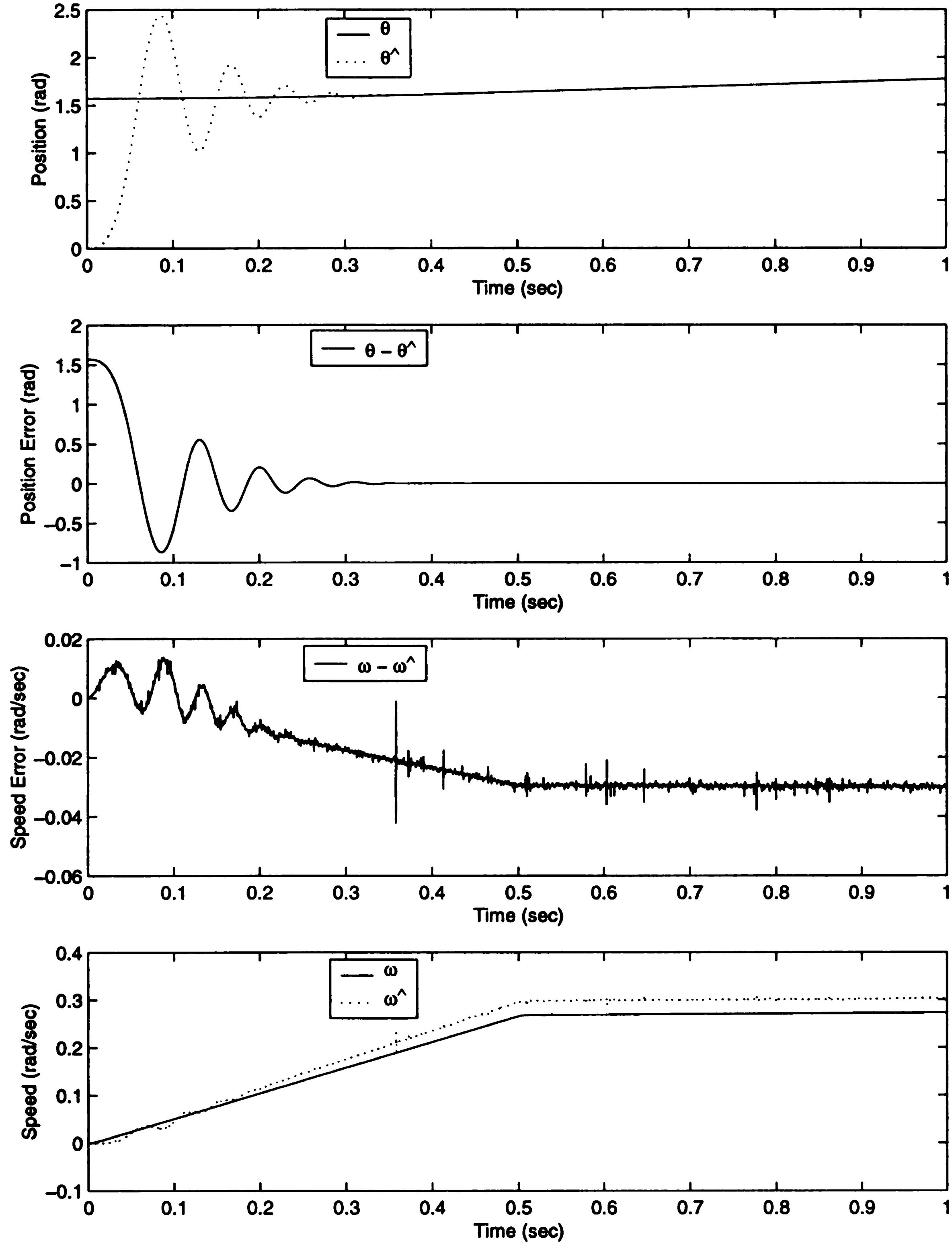

Figure 4.8. Simulation: $\Delta\theta_o : 90^o, \hat{R} : 0.9R$, ref speed: 1 RPM.

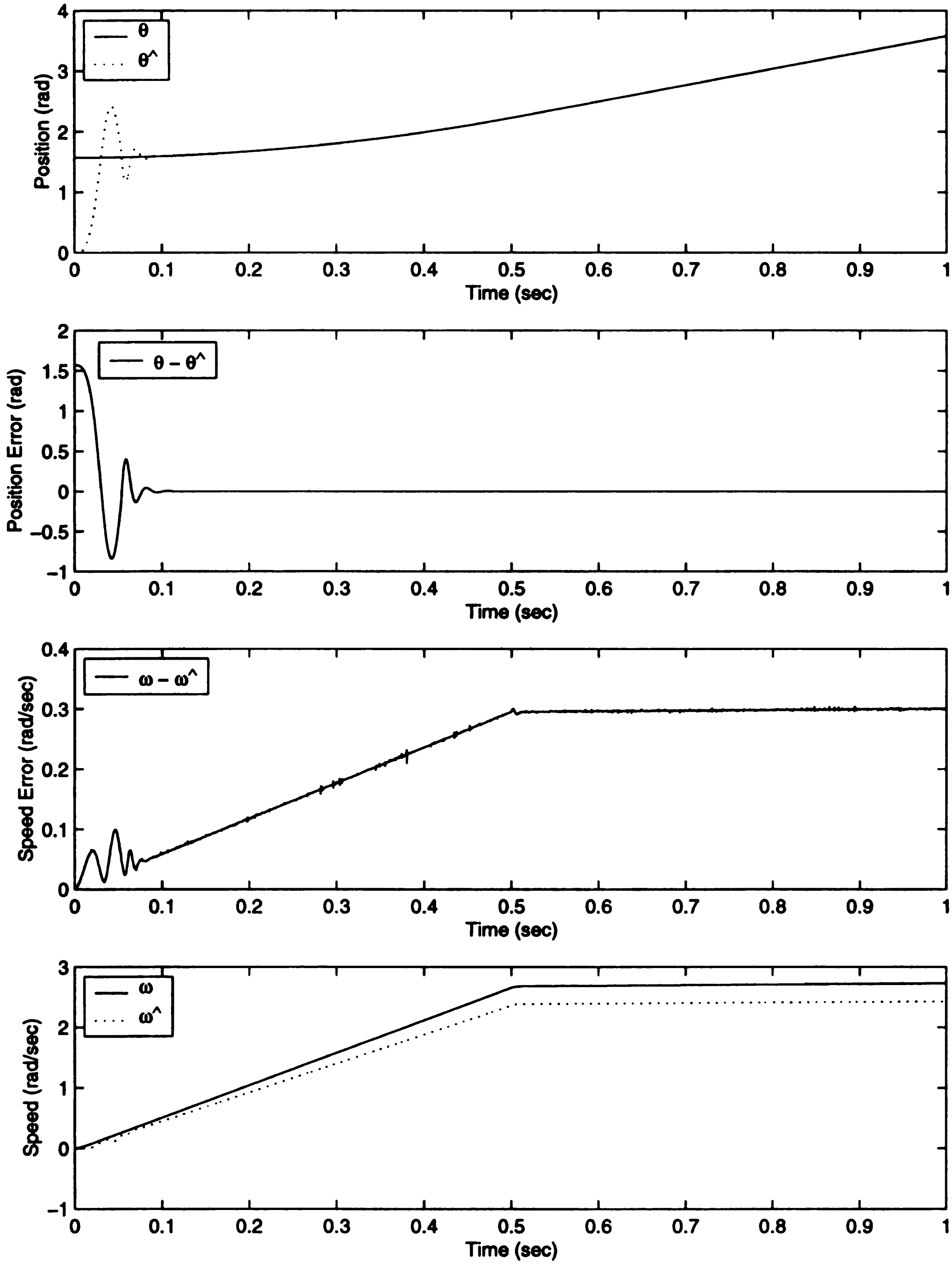

Figure 4.9. Simulation: $\Delta\theta_o : 90^o, \hat{R} : 1.1R$, ref speed: 8 RPM.

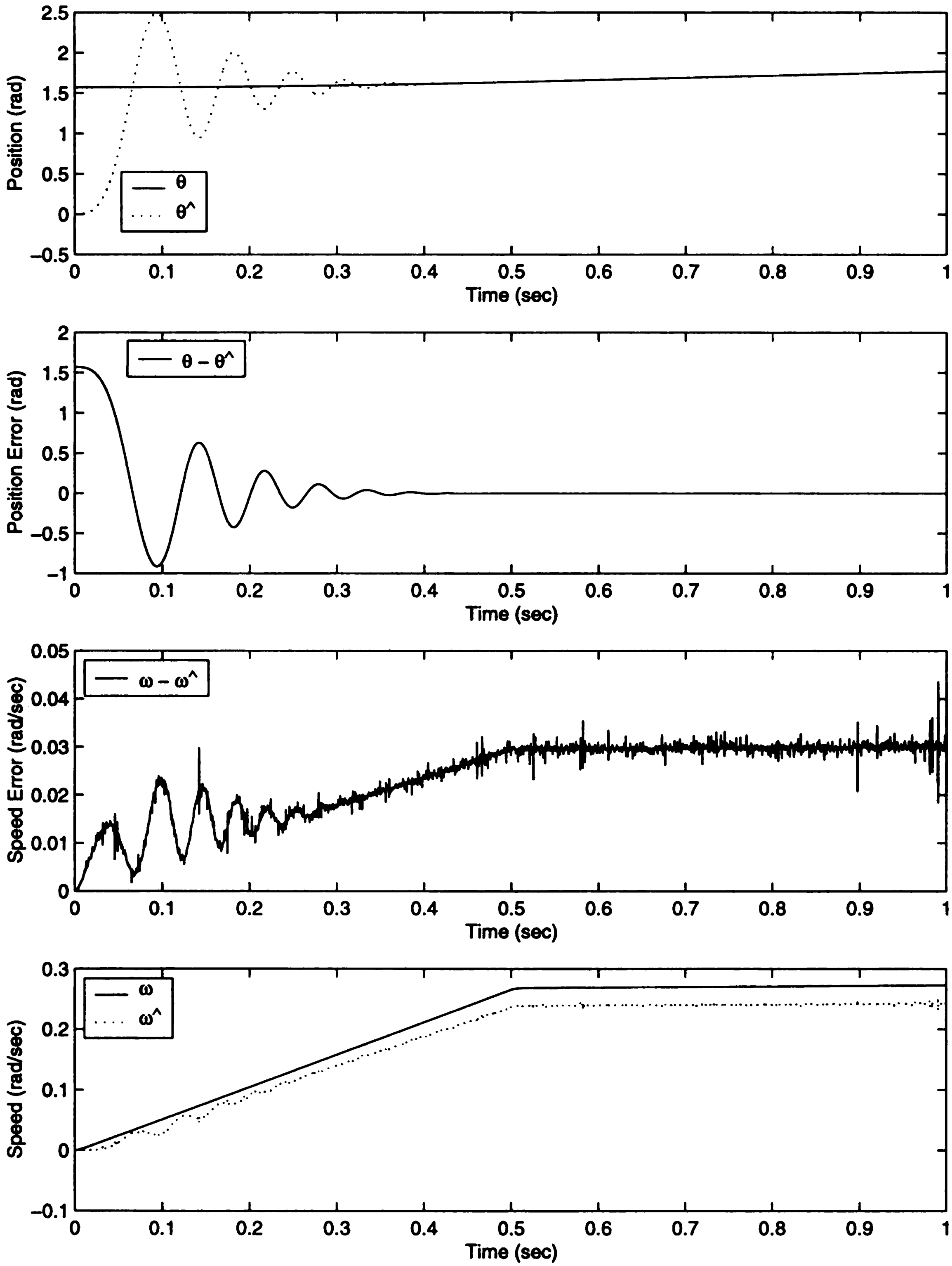

Figure 4.10. Simulation: $\Delta\theta_o : 90^o, \hat{R} : 1.1R$, ref speed: 1 RPM.

CHAPTER 5

Mathematical Analysis

5.1 Introduction

So far, the proposed scheme has been developed on the basis of intuition: starting with the motor model, we transformed the electrical variables to the assumed rotor frame of reference; manipulation of the resulting equations provided us with the means of estimating rotor position and speed. Specifically, we calculated the derivatives of the two rotor-frame currents, $\hat{i}_q$ and $\hat{i}_d$, each two different ways. The two derivatives of $\hat{i}_d$ were called $p\hat{i}_d$ and $p\hat{i}_{dm}$. Similarly, the derivatives of $\hat{i}_q$ were called $p\hat{i}_q$ and $p\hat{i}_{qm}$. We then calculated the difference of the corresponding derivatives. This produced the signals $\Delta p\hat{i}_d$ and $\Delta p\hat{i}_q$. We saw that an error in our position estimate, $\Delta\theta$, manifested itself in the signal $\Delta p\hat{i}_d$, while an error in our speed estimate, $\Delta\omega$, reflected itself in the signal $\Delta p\hat{i}_q$. This observation provided us with the mechanism to correct our estimates. This is the intuitive basis of the estimation scheme.

In this chapter, we shall recast the proposed scheme in the standard controls terminology, in effect we begin to bridge the gap between intuition and rigor. We shall show that the scheme consists of two uncoupled first order high-gain observers [9], one each for position and speed. Each observer is driven by an independent measurement. This lets us study the stability of each observer independently. The high-gain feature analytically explains the rapid convergence of the position and speed estimates to the actual values, as we observed numerically in the simulation results.

This chapter starts with the analysis of the proposed scheme, proceeds with arguing the stability of the speed- and the position-error dynamics, and concludes with an explanation of the effect of parameter mismatch.

5.2 Analysis

We begin the analysis by reviewing the key relationships. The implemented observer equations are:

$$\hat{\omega}_{new} = \hat{\omega}_{old} - k\Delta p\hat{i}_q \tag{5.1}$$

$$\hat{\theta}_{new} = \hat{\theta}_{old} + k\frac{\Delta p\hat{i}_d}{\hat{\omega}_{new}} \tag{5.2}$$

where,

- $\hat{\omega}_{new}$ is the new corrected speed estimate, and can be written as $\hat{\omega}_{k+1}$;
- $\hat{\omega}_{old}$ is the old speed estimate, and can be written as $\hat{\omega}_k$;
- $-k\Delta p\hat{i}_q$ is the speed correction term;
- $k = \frac{L}{\psi_r}$, the ratio of the stator phase inductance to the rotor flux linkage;
- $\Delta p\hat{i}_q = p\hat{i}_q - p\hat{i}_{qm}$, the difference between the two derivatives of $\hat{i}_q$;
- $\hat{\theta}_{new}$ is the new corrected position estimate, and can be written as $\hat{\theta}_{k+1}$;
- $\hat{\theta}_{old}$ is the old position estimate, and can be written as $\hat{\theta}_k$;
- $k\Delta p\hat{i}_d/\hat{\omega}_{new}$ is the position correction term;
- $\Delta p\hat{i}_d = p\hat{i}_d - p\hat{i}_{dm}$, the difference between the two derivatives of $\hat{i}_d$.

Equations (5.1) and (5.2) are based on the following relationship between estimation errors and differences in the current derivatives:

$$\begin{bmatrix} \Delta \hat{pi}_d \\ \Delta \hat{pi}_q \end{bmatrix} = \frac{\psi_r}{L} \begin{bmatrix} \omega \Delta\theta \\ -\Delta\omega \end{bmatrix} \tag{5.3}$$

where,

$$\Delta\theta = \theta - \hat{\theta} \tag{5.4}$$

$$\Delta\omega = \omega - \hat{\omega} \tag{5.5}$$

Having reviewed the key relationships, we proceed to a rigorous treatment of the proposed observer. Our objective is to estimate the position and speed of the motor. So, we begin with the mechanical dynamics of the motor, which can be expressed as:

$$\dot{\theta} = F_1 \tag{5.6}$$

$$\dot{\omega} = F_2 \tag{5.7}$$

where, F_1 represents to the motor speed ω, while F_2 represents its acceleration. The implemented observer equations, (5.1) and (5.2), can be expressed as:

$$\dot{\hat{\theta}} = h_1 \Delta \hat{pi}_d \tag{5.8}$$

$$\dot{\hat{\omega}} = h_2 \Delta \hat{pi}_q \tag{5.9}$$

where h_1 and h_2 are the observer gains whose choices will be justified based on stability considerations.

Equations (5.3), (5.8), and (5.9) yield:

$$\dot{\hat{\theta}} = h_1 \frac{\psi_r}{L} \omega \Delta\theta \tag{5.10}$$

$$\dot{\hat{\omega}} = -h_2 \frac{\psi_r}{L} \Delta\omega \tag{5.11}$$

Differentiating both sides of equations (5.4) and (5.5) with respect to time, we get:

$$(\dot{\Delta\theta}) = \dot{\theta} - \dot{\hat{\theta}} \tag{5.12}$$

$$(\dot{\Delta\omega}) = \dot{\omega} - \dot{\hat{\omega}} \tag{5.13}$$

The task at hand is to show that the intuitive choices made for the observer gains h_1 and h_2 lead to the conclusion that both the position-error dynamics $(\dot{\Delta\theta})$ and the speed-error dynamics $(\dot{\Delta\omega})$ are stable.

Substituting expressions for $\dot{\theta}$, $\dot{\hat{\theta}}$, $\dot{\omega}$, and $\dot{\hat{\omega}}$ from equations (5.6), (5.10), (5.7), and (5.11), respectively, into equations (5.12) and (5.13), we get:

$$\begin{aligned} (\dot{\Delta\theta}) &= -h_1 \frac{\psi_r}{L} \omega \Delta\theta + F_1 \\ (\dot{\Delta\omega}) &= h_2 \frac{\psi_r}{L} \Delta\omega + F_2 \end{aligned}$$

These error dynamics can be put in the matrix form as:

$$\begin{bmatrix} \Delta\dot{\theta} \\ \Delta\dot{\omega} \end{bmatrix} = \begin{bmatrix} -h_1 \frac{\psi_r}{L} \omega & 0 \\ 0 & h_2 \frac{\psi_r}{L} \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta\omega \end{bmatrix} + \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

or more succinctly, as:

$$\dot{e} = Ae + f$$

where,

$$A = \begin{bmatrix} -h_1 \frac{\psi_r}{L}\omega & 0 \\ 0 & h_2 \frac{\psi_r}{L} \end{bmatrix}, \quad e = \begin{bmatrix} \Delta\theta \\ \Delta\omega \end{bmatrix}, \quad f = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

In this analysis, the f will be treated as a bounded disturbance vector and conditions will be introduced to ensure that it does not destroy the stability property of the rest of the system dynamics. The above formulation decouples the position and speed error dynamics, so each one can be studied independent of the other.

5.2.1 Speed-Error Dynamics

The speed-error dynamics are:

$$(\dot{\Delta\omega}) = h_2 \frac{\psi_r}{L} \Delta\omega + F_2 \tag{5.14}$$

Now, we shall show that these dynamics are stable. To that end, the implemented equation for ω_{new} , (5.1), is rewritten and manipulated:

$$\begin{aligned} \hat{\omega}_{new} &= \hat{\omega}_{old} - k\Delta p\hat{i}_q \\ \hat{\omega}_{new} - \hat{\omega}_{old} &= -k\Delta p\hat{i}_q \\ \hat{\omega}_{new} - \hat{\omega}_{old} &= -\frac{L}{\psi_r}\Delta p\hat{i}_q \\ \frac{\hat{\omega}_{new} - \hat{\omega}_{old}}{T} &= -\frac{L}{\psi_r T}\Delta p\hat{i}_q \end{aligned} \tag{5.15}$$

Equation (5.9) is also rewritten:

$$\dot{\hat{\omega}} = h_2 \Delta p\hat{i}_q \tag{5.16}$$

Equations (5.15) and (5.16) show that the speed observer implementation corresponds to choosing h_2 as

$$h_2 = -\frac{L}{\psi_r T}$$

where T is the sampling time period. Substituting this choice for h_2 into the speed-error dynamics equation, (5.14), we get:

$$(\dot{\Delta\omega}) = -\frac{\Delta\omega}{T} + F_2$$

The typical sampling frequency is $10kHz - 20kHz$. This translates into a very high gain for the linear part of the above equation. This high-gain observer formulation ensures that the effect of the disturbance term F_2 can be neglected and hence, the error in speed estimation assumes an infinitesimal value in a short time (less than a tenth of a second in our initial simulations). Neglecting the disturbance term, the above equation can be written as:

$$(\dot{\Delta\omega}) = -\frac{\Delta\omega}{T}$$

This concludes the stability proof of the speed error dynamics. We have shown that the speed error is ultimately bounded [19], with the bound proportional to the sampling period T.

5.2.2 Position-Error Dynamics

The argument for the stability of the position-error dynamics closely mirrors the one presented above for those of the speed-error.

The position-error dynamics are:

$$(\dot{\Delta\theta}) = -h_1 \frac{\psi_r}{L} \omega \Delta\theta + F_1 \tag{5.17}$$

Now, we shall show that these dynamics are stable. To that end, the implemented equation for θ_{new} , (5.2), is rewritten and manipulated:

$$\begin{aligned}
\hat{\theta}_{new} &= \hat{\theta}_{old} + \frac{k}{\hat{\omega}_{new}}\Delta\hat{pi}_d \\
\hat{\theta}_{new} - \hat{\theta}_{old} &= \frac{k}{\hat{\omega}_{new}}\Delta\hat{pi}_d \\
\hat{\theta}_{new} - \hat{\theta}_{old} &= \frac{L}{\psi_r\hat{\omega}_{new}}\Delta\hat{pi}_d \\
\frac{\hat{\theta}_{new} - \hat{\theta}_{old}}{T} &= \frac{L}{\psi_r T\hat{\omega}_{new}}\Delta\hat{pi}_d \qquad (5.18)
\end{aligned}$$

Equation (5.8) is also rewritten:

$$\dot{\hat{\theta}} = h_1\Delta\hat{pi}_d \qquad (5.19)$$

Equations (5.18) and (5.19) show that the position observer implementation corresponds to choosing h_1 as

$$h_1 = \frac{L}{\psi_r T\hat{\omega}_{new}}$$

where T is the sampling time. Substituting this choice for h_1 into the position-error dynamics equation, (5.17), we get:

$$(\dot{\Delta\theta}) = -\frac{\Delta\theta}{T}\frac{\omega}{\hat{\omega}_{new}} + F_1$$

Since, we have already argued the stability of the speed-error dynamics, the factor $\frac{\omega}{\hat{\omega}_{new}} \longrightarrow 1+\epsilon$, with ϵ small, very fast.

Neglecting ϵ, we are left with,

$$(\dot{\Delta\theta}) = -\frac{\Delta\theta}{T} + F_1$$

The comments made regarding the stability of the speed-error dynamics apply here as well: the typical sampling frequency is $10kHz - 20kHz$. This translates into a very high gain for the linear part of the above equation. Similar to what we saw above, this high-gain observer formulation does two things: first, it ensures that the error in position estimation becomes infinitesimal in a short time (less than a tenth of a second in our initial simulations) and second, the effect of disturbance term F_1 can be neglected. Neglecting the disturbance term, the above equation can be written as:

$$(\dot{\Delta\theta}) = -\frac{\Delta\theta}{T}$$

This concludes the stability proof of the position error dynamics.

5.3 Effect of Parameter Mismatch

In this section, we present an analytical explanation of the observer behavior in the presence of parameter mismatch. The parameter of interest is R, the stator resistance.

The resistance mismatch can be modeled by using $\hat{R} = R + \Delta R$ instead of R in our derivation, where R is the nominal stator resistance, while ΔR is the mismatch. This affects the development starting with equation (3.14). All prior equations remain unaffected. Equations (3.13) and (3.14) are rewritten below for convenience:

$$p\begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} = \frac{1}{L}\left\{\begin{bmatrix} \hat{v}_d \\ \hat{v}_q \end{bmatrix} - \begin{bmatrix} R & -L\hat{\omega} \\ L\hat{\omega} & R \end{bmatrix}\begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} - \psi_r\omega\begin{bmatrix} -\sin\Delta\theta \\ \cos\Delta\theta \end{bmatrix}\right\} \tag{5.20}$$

$$p\begin{bmatrix} \hat{i}_{dm} \\ \hat{i}_{qm} \end{bmatrix} = \frac{1}{L}\left\{\begin{bmatrix} \hat{v}_d \\ \hat{v}_q \end{bmatrix} - \begin{bmatrix} R & -L\hat{\omega} \\ L\hat{\omega} & R \end{bmatrix}\begin{bmatrix} \hat{i}_d \\ \hat{i}_q \end{bmatrix} - \psi_r\hat{\omega}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}$$

With the resistance mismatch incorporated, the above equation becomes:

$$p\begin{bmatrix}\hat{i}_{dmr}\\ \hat{i}_{qmr}\end{bmatrix}=\frac{1}{L}\left\{\begin{bmatrix}\hat{v}_d\\ \hat{v}_q\end{bmatrix}-\begin{bmatrix}R+\Delta R & -L\hat{\omega}\\ L\hat{\omega} & R+\Delta R\end{bmatrix}\begin{bmatrix}\hat{i}_d\\ \hat{i}_q\end{bmatrix}-\psi_r\hat{\omega}\begin{bmatrix}0\\ 1\end{bmatrix}\right\} \quad (5.21)$$

Here, including the subscript r in the signal names $p\hat{i}_{dmr}$ and $p\hat{i}_{qmr}$ indicates the presence of resistance mismatch. The same convention will be followed below as well.

Defining:

$$\begin{bmatrix}\Delta p\hat{i}_{dr}\\ \Delta p\hat{i}_{qr}\end{bmatrix}=p\begin{bmatrix}\hat{i}_d\\ \hat{i}_q\end{bmatrix}-p\begin{bmatrix}\hat{i}_{dmr}\\ \hat{i}_{qmr}\end{bmatrix}$$

(5.20) - (5.21) yields the difference in current derivatives:

$$\begin{bmatrix}\Delta p\hat{i}_{dr}\\ \Delta p\hat{i}_{qr}\end{bmatrix}=\begin{bmatrix}\frac{\psi_r}{L}\omega\sin\Delta\theta+\frac{\Delta R}{L}\hat{i}_d\\ \frac{\psi_r}{L}(-\omega\cos\Delta\theta+\hat{\omega})+\frac{\Delta R}{L}\hat{i}_q\end{bmatrix}$$

As before, we assume that $\Delta\theta$ is small, and use the following simplification:

$$\sin\Delta\theta\approx\Delta\theta$$
$$\cos\Delta\theta\approx 1$$

Thus,

$$\begin{bmatrix}\Delta p\hat{i}_{dr}\\ \Delta p\hat{i}_{qr}\end{bmatrix}=\begin{bmatrix}\frac{\psi_r}{L}\omega\Delta\theta+\frac{\Delta R}{L}\hat{i}_d\\ -\frac{\psi_r}{L}\Delta\omega+\frac{\Delta R}{L}\hat{i}_q\end{bmatrix} \quad (5.22)$$

Rearranging (5.22), we get:

$$\begin{aligned}\Delta\omega &= -\frac{L}{\psi_r}\Delta p\hat{i}_{qr}+\frac{\Delta R}{\psi_r}\hat{i}_q\\ \Delta\theta &= \frac{L}{\psi_r\omega}\Delta p\hat{i}_{dr}-\frac{\Delta R}{\psi_r\omega}\hat{i}_d\end{aligned}$$

Again, defining $k=L/\psi_r$ and substituting the definitions of $\Delta\omega$ and $\Delta\theta$ from

(5.4) and (5.5), the above set of equations can be re-written as:

$$\hat{\omega}_{new} = \hat{\omega}_{old} - k\Delta p\hat{i}_{qr} + \frac{\Delta R}{\psi_r}\hat{i}_q \tag{5.23}$$

$$\hat{\theta}_{new} = \hat{\theta}_{old} + k\frac{\Delta p\hat{i}_{dr}}{\hat{\omega}_{new}} - \frac{\Delta R}{\psi_r\hat{\omega}_{new}}\hat{i}_d \tag{5.24}$$

Hence the resistance mismatch results in one additional term each in the expressions for $\hat{\omega}_{new}$ and $\hat{\theta}_{new}$.

Effect on Position Estimate: Our estimation effort is focused in the speed region where there is no field weakening effect. This means that we choose to force $\hat{i}_d$, the flux-producing component of the stator current in the estimated rotor frame of reference, to be zero. This is done to obtain the maximum torque-to-current ratio. This optimal mode of operation is suitable below the base rotor speed, where sufficient voltage is available from the inverter which supplies the stator windings of the motor. The hysteresis current controller ensures this, and the resulting $\hat{i}_d$ that flows is very close to zero. As a result, we can analytically explain what we observed in our simulations: the proposed scheme works well as far the position estimation, even in the presence of resistance mismatch.

The situation is different at speeds above the base speed, in the constant-power range. Since the permanent magnet rotor flux is constant, the induced EMF increases directly with the motor speed. If a given speed is to be reached, the terminal voltage must also be increased to match the increased stator EMF. The increased stator terminal voltage would require an increase in the voltage rating of the inverter used. However, with a given inverter, there is a ceiling voltage which cannot be exceeded. Thus to limit the terminal voltage of the motor to the ceiling voltage of the inverter, field weakening has to be introduced. The effect of field weakening can be obtained by controlling the stator currents in such a way that the stator-current space vector

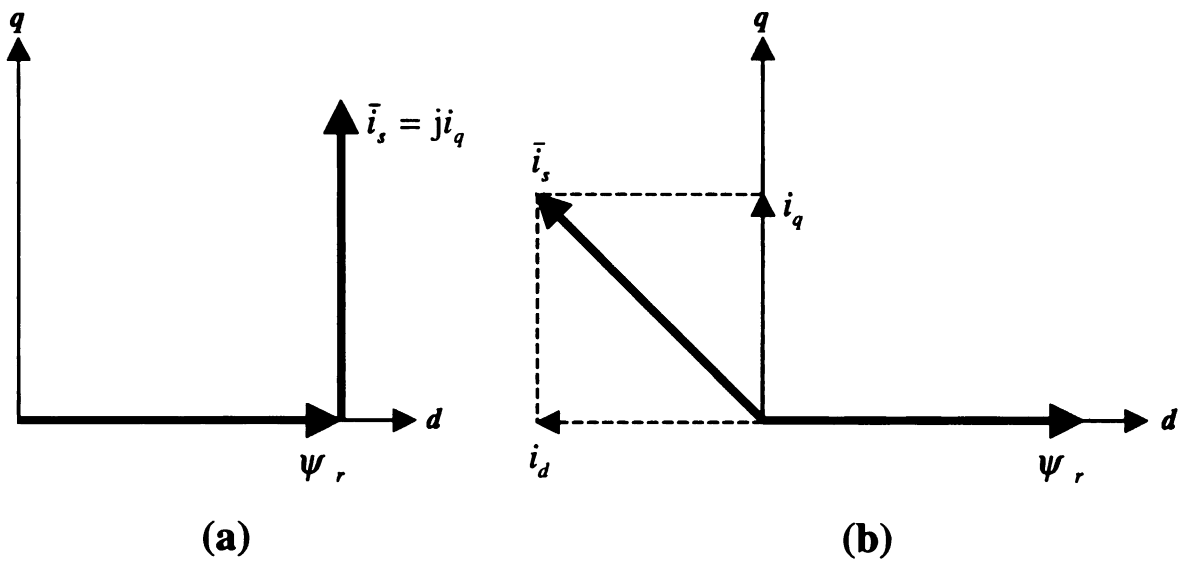

Figure 5.1. Orthogonal orientation of the current space vector vis-à-vis rotor flux linkage : (a) base region (b) field weakening region.

in the rotor reference frame contains a direct-axis component i_d along the negative direct-axis of the rotor reference frame, in addition to the quadrature-axis stator current component i_q. This is shown in Figure 5.1.

The resistance mismatch could have more pronounced effect under field weakening. But in this region the $\hat{\omega}_{new}$ factor, in equation (5.24), in the denominator helps, since, this will act to attenuate the impact of having non-zero $\hat{i}_d$. This explains why the resistance mismatch will have no appreciable effect on the position estimate, even with field weakening in place.

Effect on Speed Estimate: The situation is different in the case of speed estimation: the resistance mismatch results in the term $(\Delta R/\psi_r)\hat{i}_q$, in equation (5.23). Here, the ΔR factor comes multiplied with $\hat{i}_q$, the torque-producing component of the stator current in the estimated rotor frame of reference. The value of $\hat{i}_q$ increases linearly with the torque demand, the electric torque developed being given by the relation:

$$\tau_e = \frac{3}{2} P \psi_r \hat{i}_q$$

where,

- τ_e is the torque developed;
- P is the number of pole pairs in the motor;
- ψ_r is the rotor flux linkage.

If there is no load attached to the motor, the torque demand is there only while starting from rest: once the desired speed is reached, $\hat{i}_q$ drops to a low value just enough to overcome friction and windage. In this situation, the impact of resistance mismatch is going to be minimal. But when the motor is loaded, this being the more typical case, the impact of resistance mismatch is significant. This will depend on the torque demanded. For a constant torque load, the $\hat{i}_q$ will be constant, and the error in speed estimation will be proportional to ΔR, as observed in simulation results.

This concludes our mathematical analysis of the proposed scheme.

CHAPTER 6

Experimental Implementation

6.1 Introduction

To investigate the viability of the proposed scheme, an implementation was carried out. We present details of this implementation in this chapter. It begins with a description of the experimental setup built for this purpose. It continues with the description of the software developed and presentation of results of the various experiments conducted. It concludes with a discussion of these results.

6.2 Experimental Setup

The experimental setup, built to test the scheme in real-time, is shown in Figure 6.1. It consists of the following modules:

- A 3-phase sinusoidal EMF permanent magnet synchronous motor;
- A hysteresis current controlled PWM inverter;
- A digital signal processor board, that implements the estimation algorithm;
- Current and voltage sensors;

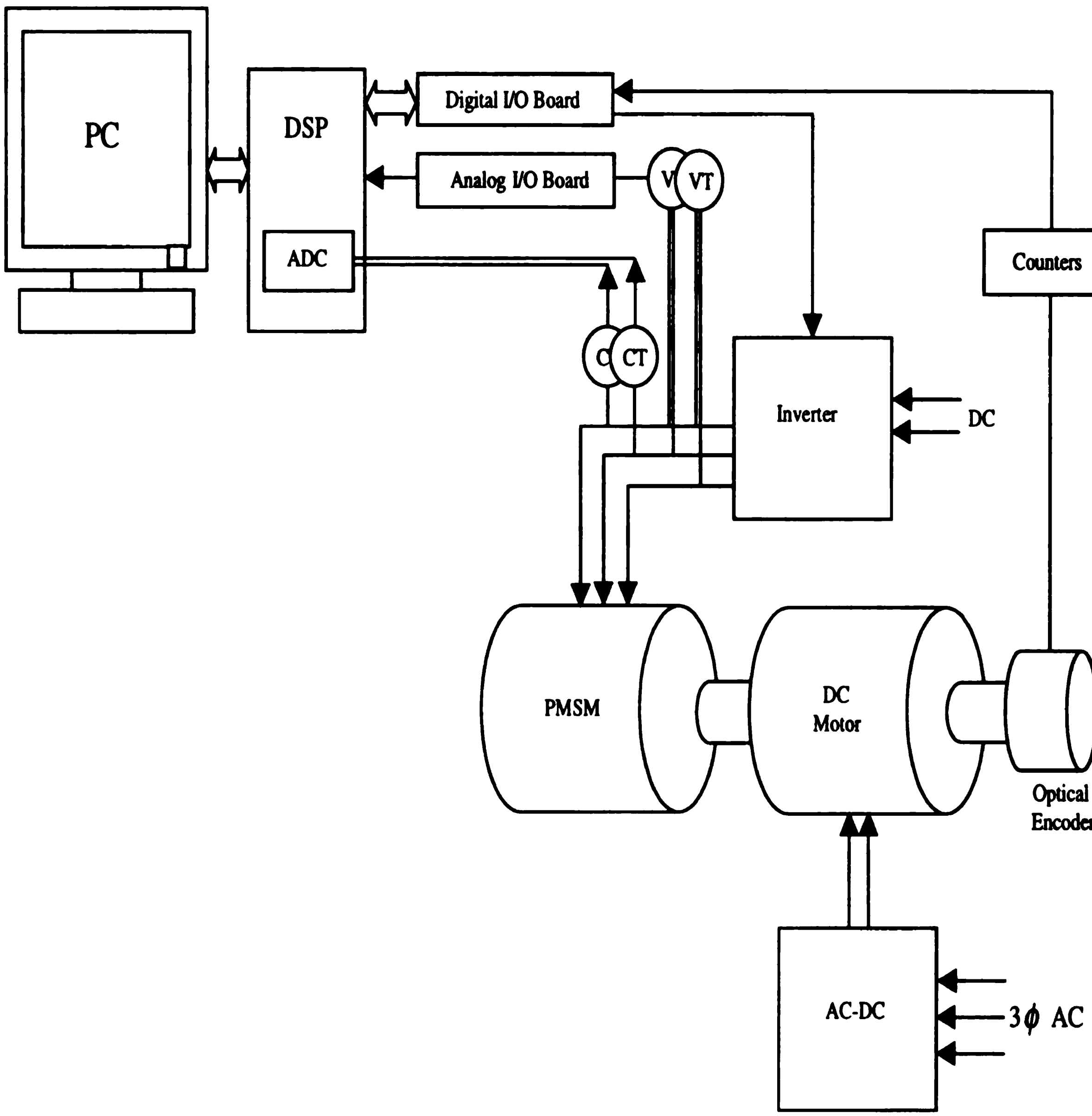

Figure 6.1. Experimental setup.

- An optical pulse encoder to verify the estimated speed and position;
- A load in the form of either a DC brake or a DC motor fed by a 3-phase controlled rectifier / regenerative dynamometer.

6.2.1 Permanent Magnet Synchronous Motor

The PMSM used is part of the high performance servo motor series manufactured by Pacific Scientific. It is a three-phase, six-pole motor and has Neodymium-Iron-Boron (NdFeB) rotor magnets. At room temperature, NdFeB has the highest energy-product of all commercially available magnets. Energy-product is the product of B, the magnetic flux density, and H, the magnetizing force or field intensity. The high remanence and coercivity of NdFeB permit reductions in motor frame-size for the same output compared with motors using ceramic magnets. This translates into high torque-to-inertia ratio. The PMSM used is rated at 320V DC bus voltage. The nominal parameters are given in Table 6.1.

6.2.2 Hysteresis Current Controlled PWM Inverter

The inverter consists of a 3-phase 60 Hz 208V full-wave rectifier section followed by a set of six 20 kHz insulated-gate bipolar transistor (IGBT) switches, rated at 30 A, 60 V. The switches used are part of the *Powerex Intellimod* [30] series. They are controlled by the signals generated by the DSP board. The specific module used is PM30RSF060.

The current-controlled PWM inverter consists of a conventional PWM voltage-source inverter equipped with current-regulating loops to provide a controlled current output. Since the inverter has a high switching frequency, the stator currents of the motor can be rapidly adjusted in magnitude and phase.

Table 6.1. Parameters: Pacific Scientific S21GNNA-RNNM-00 PMSM.

Parameter	Symbol	Value	Units
Number of Pole Pairs	P	3	
Resistance	R	6	Ω
Inductance	L	8	mH
Voltage Constant	ψ_r	0.0572	$V_{peak}/rad/sec$
Rated Speed	W_r	7900	RPM
Rated Torque	T_{cr}	0.44	Nm
Moment of Inertia	J_M	0.042m	kgm^2
Thermal Resistance	R_{TH}	2.2	$deg.C/Watt$
Thermal Time Constant	τ_{TH}	5.0	$min.$
Continuous Stall Torque	T_{CS}	0.5	Nm
Continuous Stall Current	I_{cs}	1.5	A_{rms}
Static Friction (max)	T_f	0.008	Nm
Viscous Damping Coefficient	K_{DV}	0.003	$Nm/kRPM$
Weight (motor only)	W	1.4	kg

The reference current waveform is generated and fed to a comparator, together with the actual measured current of the motor. The comparator error is used to switch the devices in the inverter so as to limit the instantaneous current error. If the motor phase current is more positive than the reference current value, the upper device is turned off, and the lower device is turned on, causing the motor current to decrease, and vice versa. The comparator has a hysteresis band that determines the permitted deviation of the actual phase current from the reference value before an inverter switching is initiated. Thus, the actual current tracks the reference current without significant amplitude error or phase delay. There are three independent current controllers for each inverter phase. Figure 6.2 gives an overview of the controller, whereas Figure 6.3 shows the details.

Typical output current waveforms obtained with hysteresis current control are illustrated in Figure 6.4. Figure 6.5 shows the detailed picture. A small hysteresis band gives a near-sinusoidal motor current with a small current ripple, but requires

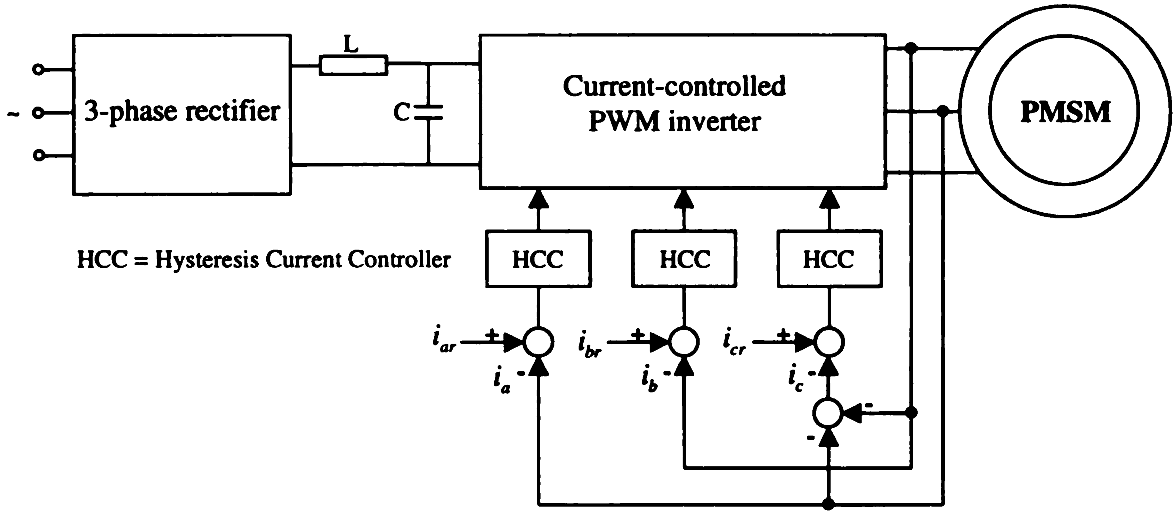

Figure 6.2. Hysteresis current controller: physical overview.

a high switching frequency in the inverter. The switching frequency is not constant for a given hysteresis band, but is modulated by the variations in motor inductance and back emf.

6.2.3 Digital Signal Processor

The digital signal processor board uses the AT&T DPS32C processor. It has the following characteristics:

- A 32-bit floating-point unit;
- A 16-/24-bit fixed-point unit;
- 1536x32 bit words of on-chip memory;
- Parallel and serial interfaces.

The DSP32C processor operates at a clock rate of 50MHz and is capable of performing 25 million floating point computations in a second.

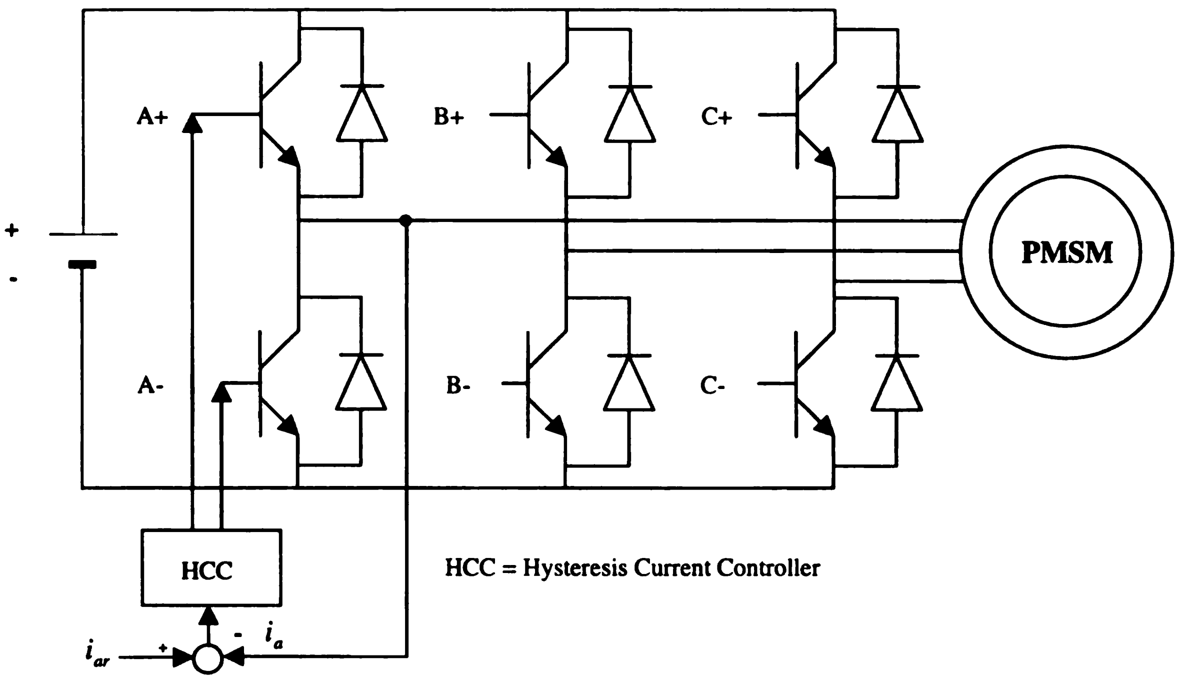

Figure 6.3. Hysteresis current controller: physical details.

The DSP board has two on-board 200kHz, 16-bit analog-to-digital converters. Also included is the DSPLINK parallel expansion bus that allows further expansion of the analog and digital I/O capabilities of the board by connecting expansion boards. The DSP board outputs the control signals, which are fed, via isolated drive control circuitry, to the gate inputs of the IGBTs. These control signals are interfaced via PC/32DIO, a 32 channel digital I/O peripheral board that connects to the DSP via the DSPLINK interface. The 32 channels are arranged as four 8-bit bidirectional ports.

6.2.4 Optical Pulse Encoder

The optical pulse encoder is an incremental device that provides pulsed output waveform. The choice of an incremental, as opposed to an absolute, encoder results in higher resolution at a lower cost, and with fewer output lines. It has a resolution of

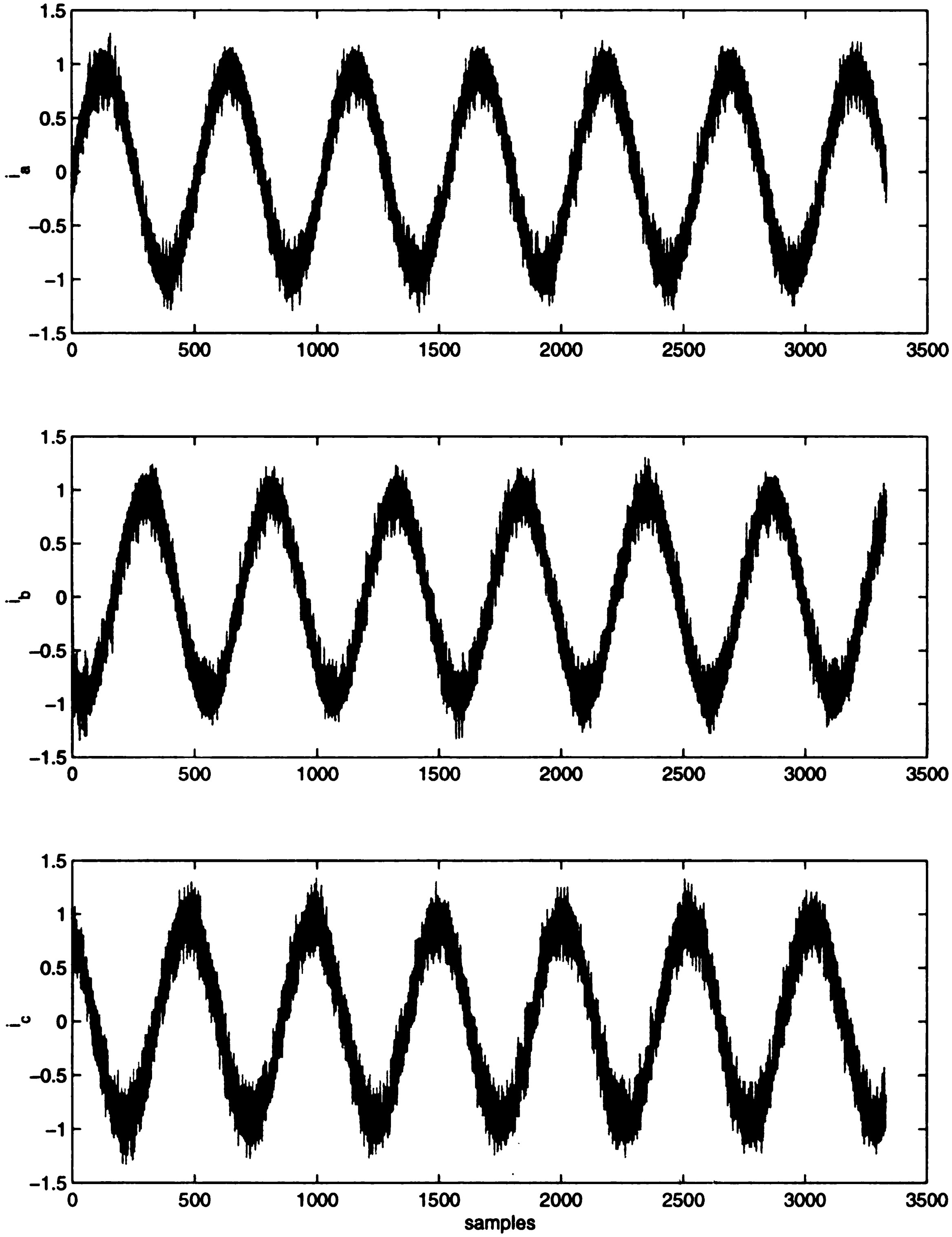

Figure 6.4. Hysteresis current controller: waveform overview.

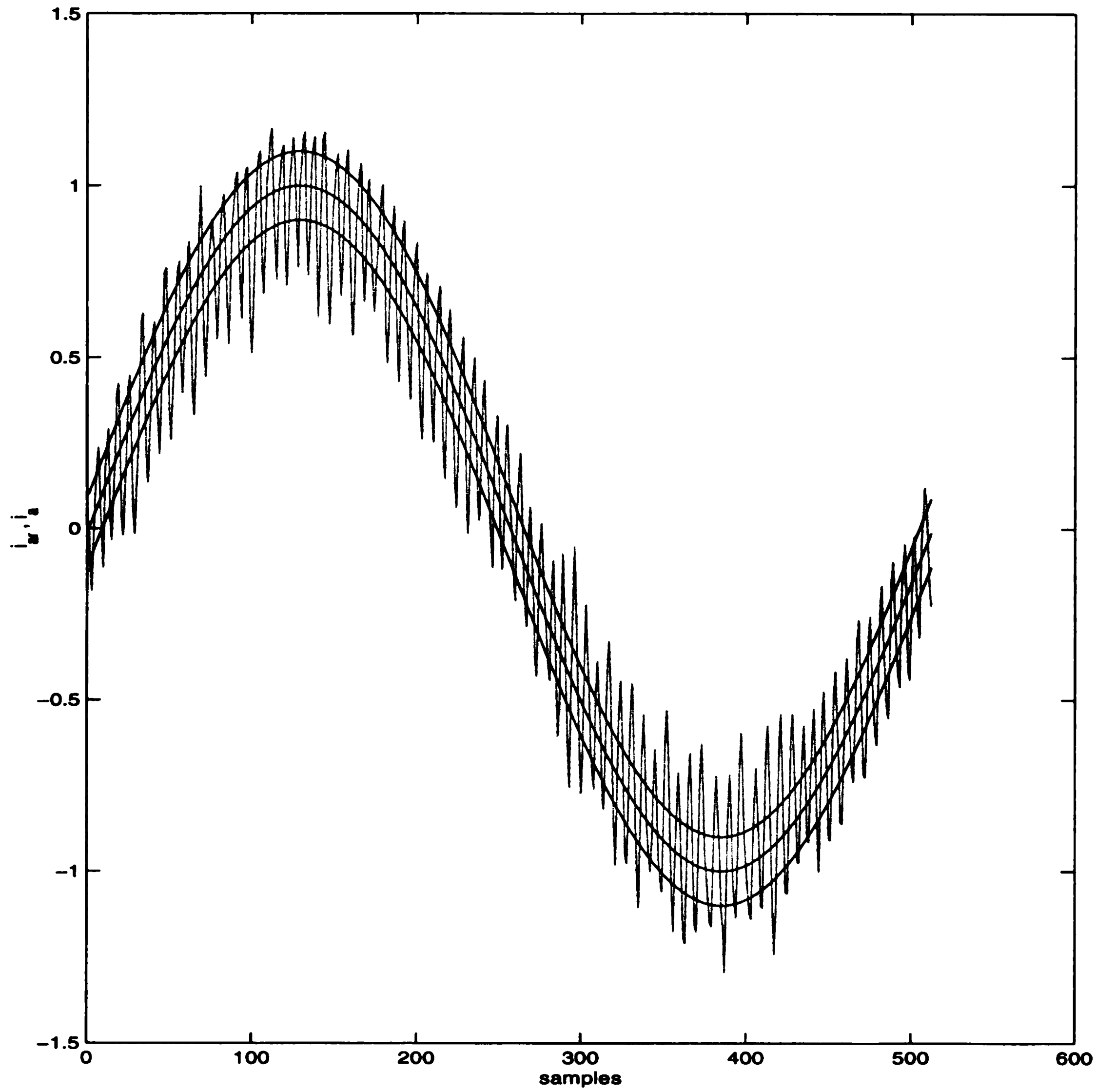

Figure 6.5. Hysteresis current controller: waveform details.

1024 pulses per revolution. It uses two output channels in quadrature for position sensing. By viewing the transition on one channel relative to the state of the other, we can detect the direction of motion as well. The output pulses generated are processed by counters and are then sent to the DSP board via two ports of the digital I/O board.

6.3 Software

To ensure generation of optimized code which executes at the speed required by the algorithm, all programming has been done in Assembly language. A flow-chart of the software developed is shown in Figures 6.6 and 6.7. The code is included in Appendix A.

The initialization step consists of configuring various control registers of the processor and the bidirectional ports of the PC/32DIO board, resetting the counters, and setting up the interrupt frequency.

This is followed by the alignment step, where a current pulse is repeatedly applied to the motor. The objective is to bring the rotor to the zero degree position. When this is accomplished, the north pole of the permanent magnet is coincident with the as-axis, the magnetic axis of the phase-a winding. This processing is shown in Figure 6.8.

The motor is started by introducing a revolving magnetic field. This is done by injecting three-phase sinusoidal currents, with appropriate phases, into the stator. The frequencies of these currents are chosen such that the motor follows a specific speed profile.

Actual currents and voltages are measured. These measured currents are compared with the reference values. Three hysteresis current controllers then calculate the appropriate switching pattern to ensure that the measured currents stay within

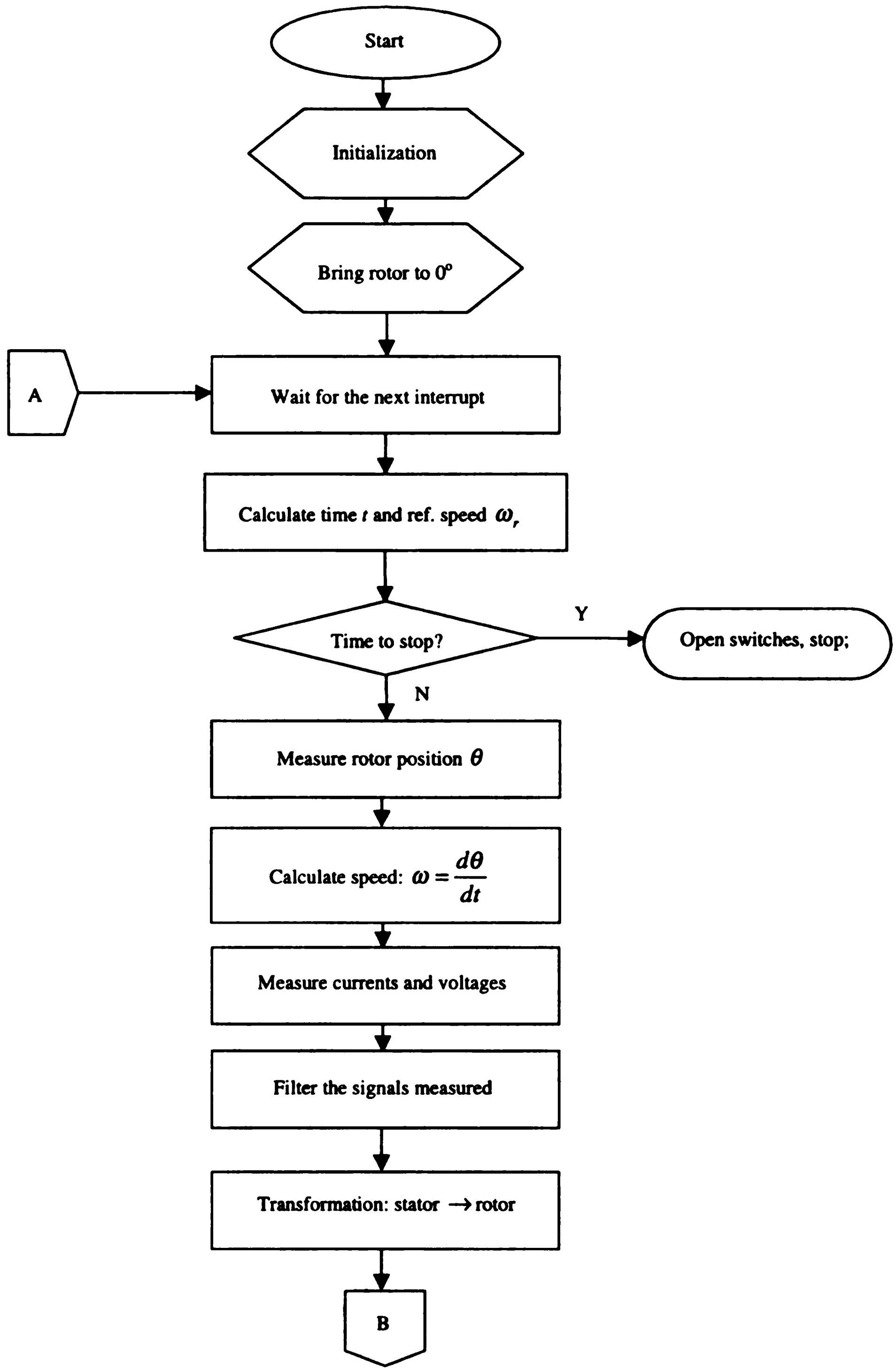

Figure 6.6. Flowchart of the Assembly language program.

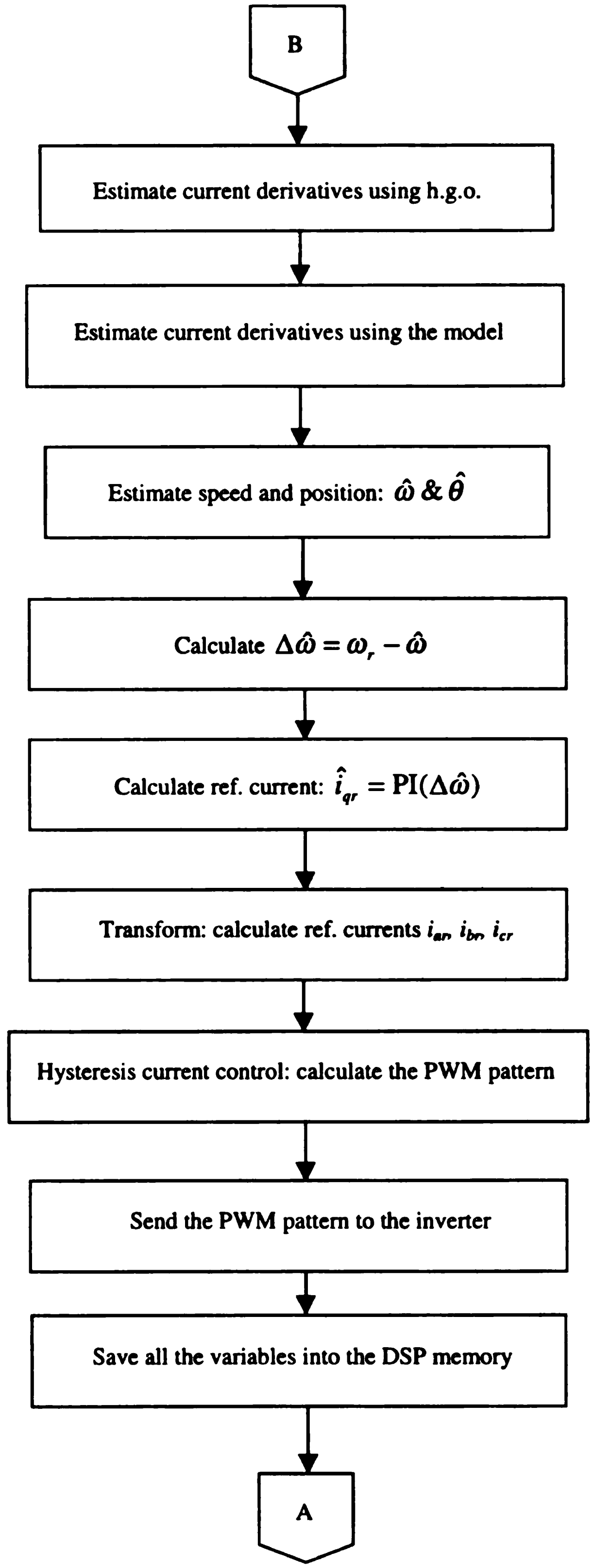

Figure 6.7. Flowchart of the Assembly language program (continued).

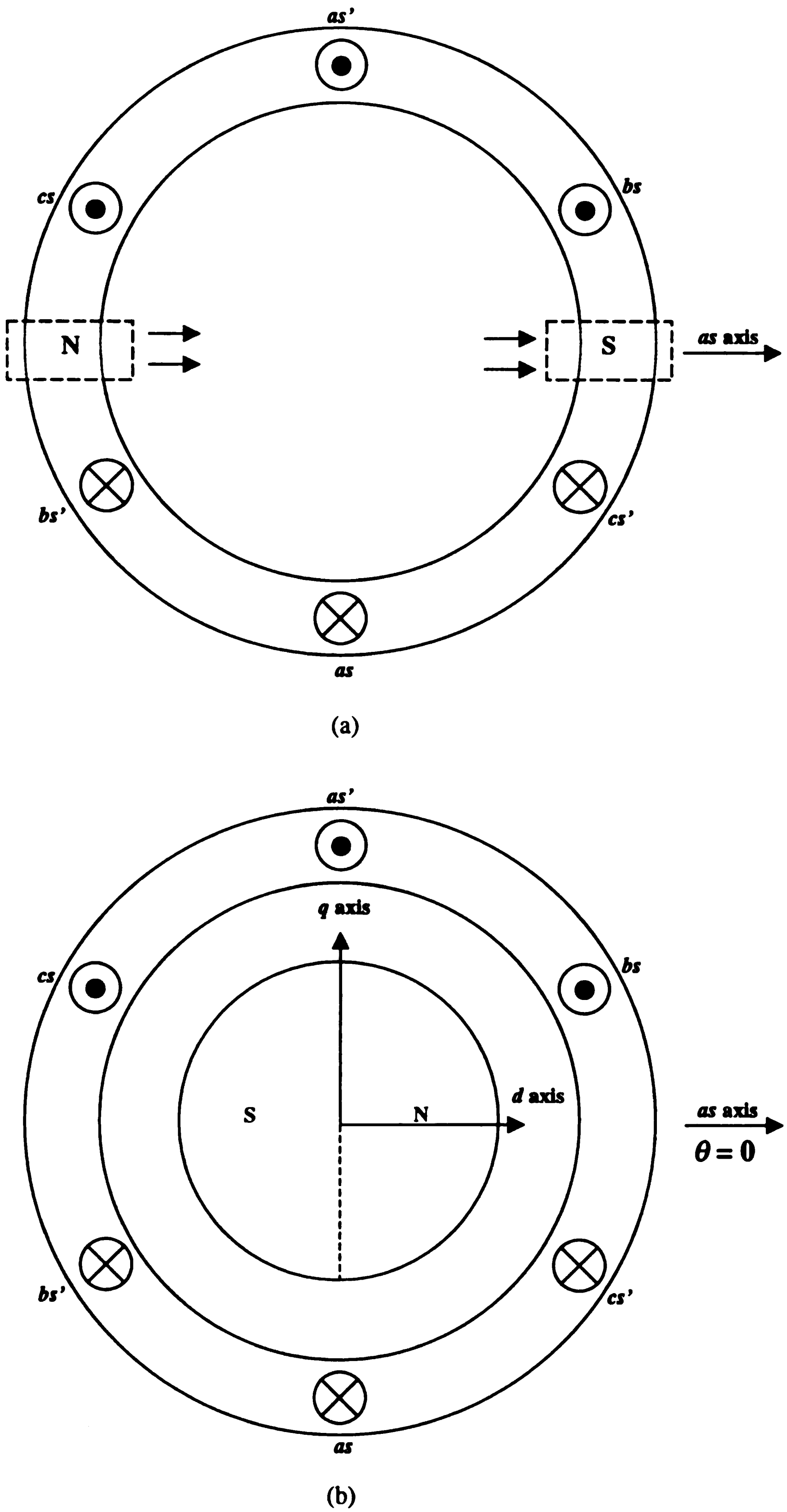

Figure 6.8. Initial rotor alignment: (a) stator magnetic field (b) resulting rotor position.

the hysteresis bandwidth of the reference ones. This pattern is sent to the inverter controller, which translates the switching pattern to actual switches being closed or opened.

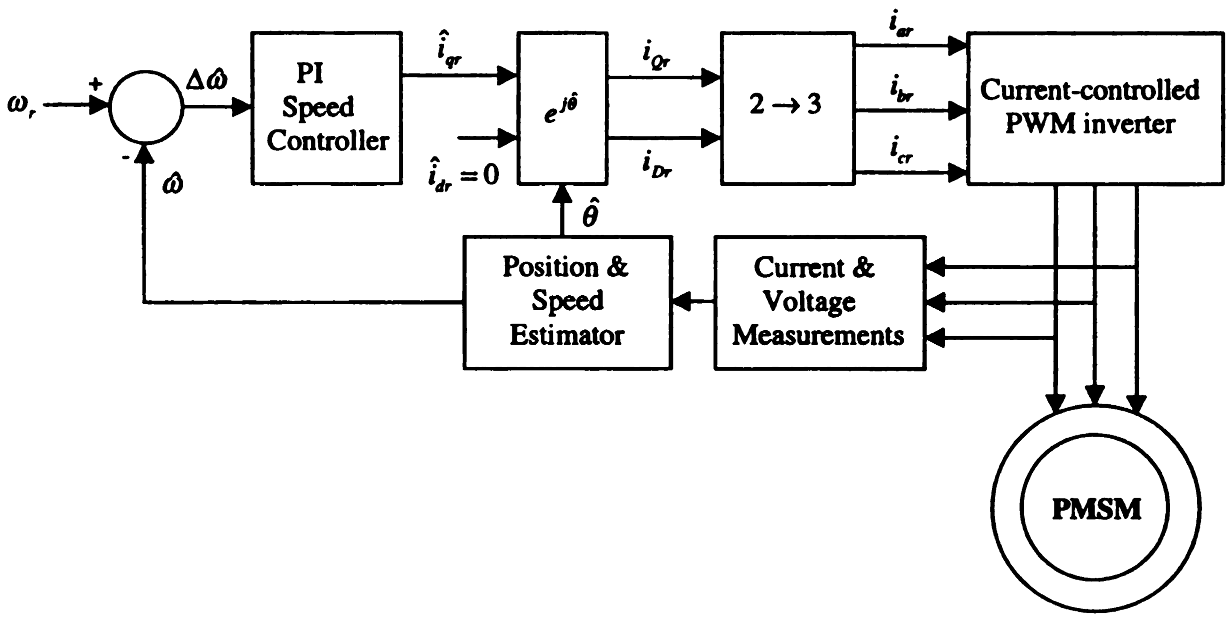

Figure 6.9. Sensorless closed-loop operation of the PMSM.

Torque is produced and the motor starts rotating. The position, θ, is measured and speed, ω, is calculated using a high-gain observer. The details of the high-gain observer were given in Section 2.5.

The measured currents and voltages are filtered using second order Butterworth low-pass filters. This filtering is needed to get rid of the current ripple which does not contribute to the torque produced but can cause problems with the calculation of derivatives using the high-gain observer. The DSP board has its own analog low-pass filters. These filters limit noise and provide anti-aliasing protection.

The filtered currents and voltages are transformed to the rotor frame of reference, using the estimate of position from the previous iteration. This is followed by estimation of derivatives of these transformed currents, using high-gain observers. The derivatives of these transformed currents are then computed from the model of the PMSM, using Equation (3.8). Position and speed are then estimated using Equations (3.15) and (3.16) of the proposed scheme.

For closed loop operation, the motor is started open loop, as described above. When it reaches a certain speed, the loop is closed: speed estimate is fed back to a PI controller, which compares the estimate with the reference speed. The output of the controller is, $\hat{i}_q^r$, the torque-producing component of the stator reference current in the estimated rotor frame of reference. The flux-producing component, $\hat{i}_d^r$, is chosen as zero, since no field weakening is intended. The above two reference currents are transformed to the stator frame of reference, using the position estimate. This generates the three-phase reference currents: i_{ar}, i_{br}, and, i_{cr}. These reference currents are then compared with the actual ones, as explained above. Figure 6.9 describes the processing.

Once the experiment is carried out, the resulting data are retrieved from the DSP board memory using a C language program. The data are analyzed further using *Matlab*'s visualization tools.

6.4 Experimental Results

In this section, we present results from the experiments conducted. The reference speed profile followed is similar to the one used in numerical simulations: the reference speed increases, or decreases in the case of negative speeds, for the first half second and then reaches a steady level. In some experiments, the reference speed kept increasing, while in others, bidirectional profiles are used, going positive initially and then assuming negative values.

In each of these figures, the first plot shows the actual position of the motor while the second one shows the estimated position. The third plot presents the error between the actual position and its estimate. In all these plots, the angles have been mapped to the $[-\pi, \pi]$ domain. This has been done for two reasons. First, plotting the data without this mapping can mask any estimation errors, since the original scale would go from zero to the maximum value of the position. For instance, if the motor is running at 200 radians/sec, then in two seconds, the position reaches 400 radians. On this scale, the actual and estimated position plots can look identical, even if they have some mismatch. Mapping to the $[-\pi, \pi]$ domain prevents it and reveals the two signals in detail. The second reason is related to the limitation of any practical setup. Any processor used to implement the scheme will have a finite word length. If the motor is run for a long time, the overflow and underflow errors can cause the scheme to collapse. So, in practice, the actual and estimated positions are always mapped onto the 2π long domain.

One effect of this mapping is that we notice sharp discontinuities in the position estimates while the experiment is in the initial part of its run. These discontinuities arise each time the estimation error falls outside the $[-\pi, \pi]$ domain.

Figure 6.10 shows the result of an experiment where the motor is kept speeding up to 550 RPM. In this case, the loop is closed at about 0.3 s, and the estimate of position converges to a constant number. It is important to mention here that the indication of success in these experiments is the position estimation error converging either to zero, or to a constant number. The latter case is due to the initial offset in actual position measurement and is explained below.

In simulations, we know where the rotor initially is: this being simply a matter of initializing the particular state of the state space model at the desired value. In actual practice, this cannot hold true. The present setup uses an optical pulse encoder which sends pulses to a pair of counters. As described in Section 6.3, the first part of the experiment involves initial rotor alignment, where we repeatedly apply a current pulse to the motor. The objective is to bring the rotor to a the zero degree position. We then send a pulse to the counters to reset them, so that the count starts from zero. This processing was shown in Figure 6.8. This arrangement is the most effective we can implement, since we are working with a surface-mounted permanent magnet motor. This arrangement cannot guarantee that the zero position reported by the position sensor is infact so. An offset in this initial alignment shows up as the constant number to which the position estimation error converges.

We notice that position estimator does not converge at low speed. To investigate this further, a series of experiments have been conducted, and the motor is rotated at lower speeds. Figures 6.11 - 6.14 give results of experiments, where the motor is rotated at 600, 150, 80 and 45 RPM. The accuracy of the position estimate suffers as the reference speed is decreased. In case of Fig 6.14, the estimator has been re-tuned for the low speed. It appears that depending on the intended application speed, the proposed algorithm might be tuned further to make it work below the 45 RPM threshold. But so far, in the experiments, we have been able to get reasonable results down to this speed only.

Another set of experiments conducted uses negative reference speed profiles. The motor has been made to turn at speeds of -600, -300 and -75 RPM. Figures 6.15 - 6.17 present these results.

A third set of experiments conducted involves investigating the behavior of the proposed scheme under speed reversal. Figures 6.18 and 6.19 show the results. While the estimator converges eventually, it does not show satisfactory performance in the first part of the experiment.

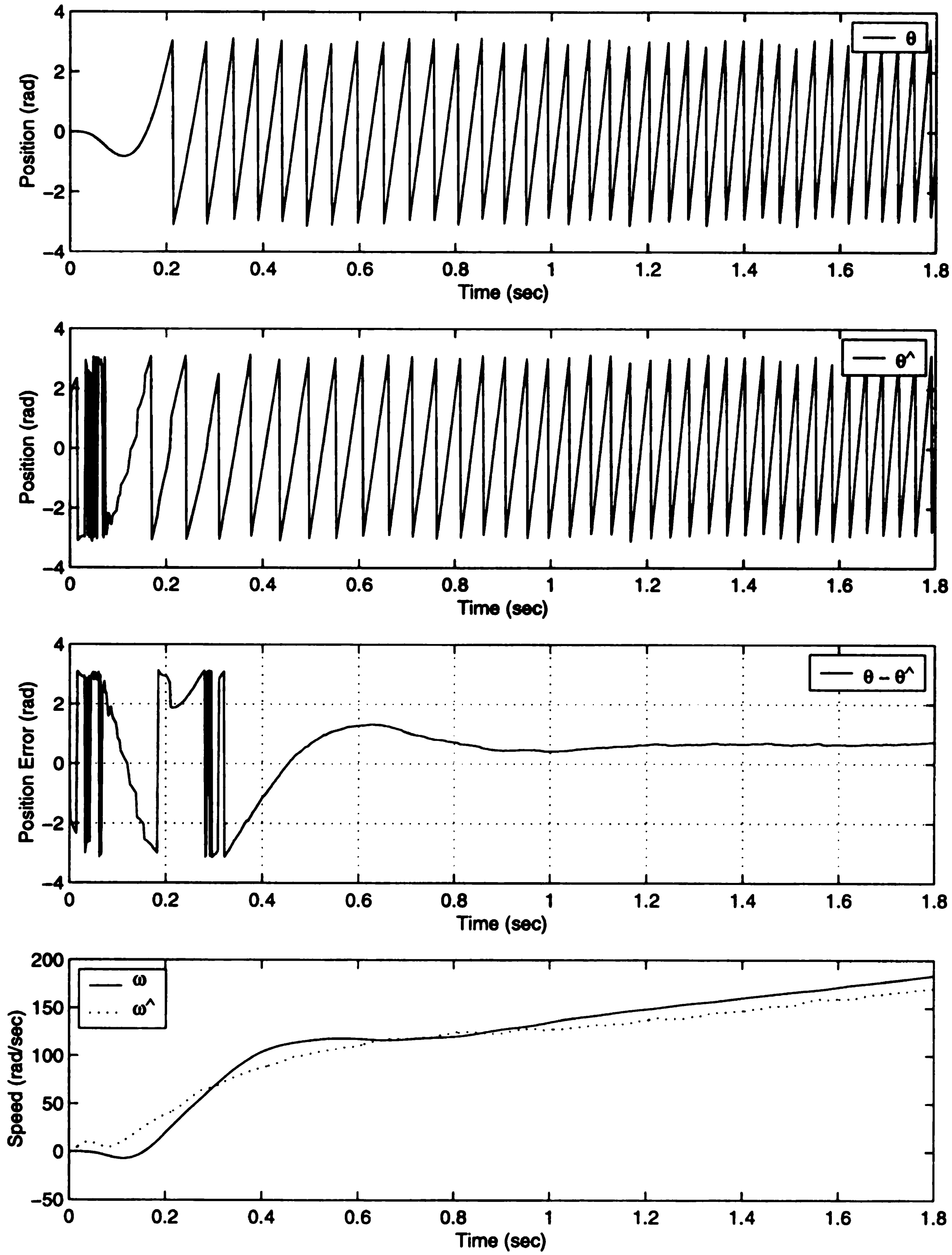

Figure 6.10. Experiment: ref speed increasing to 550 RPM.

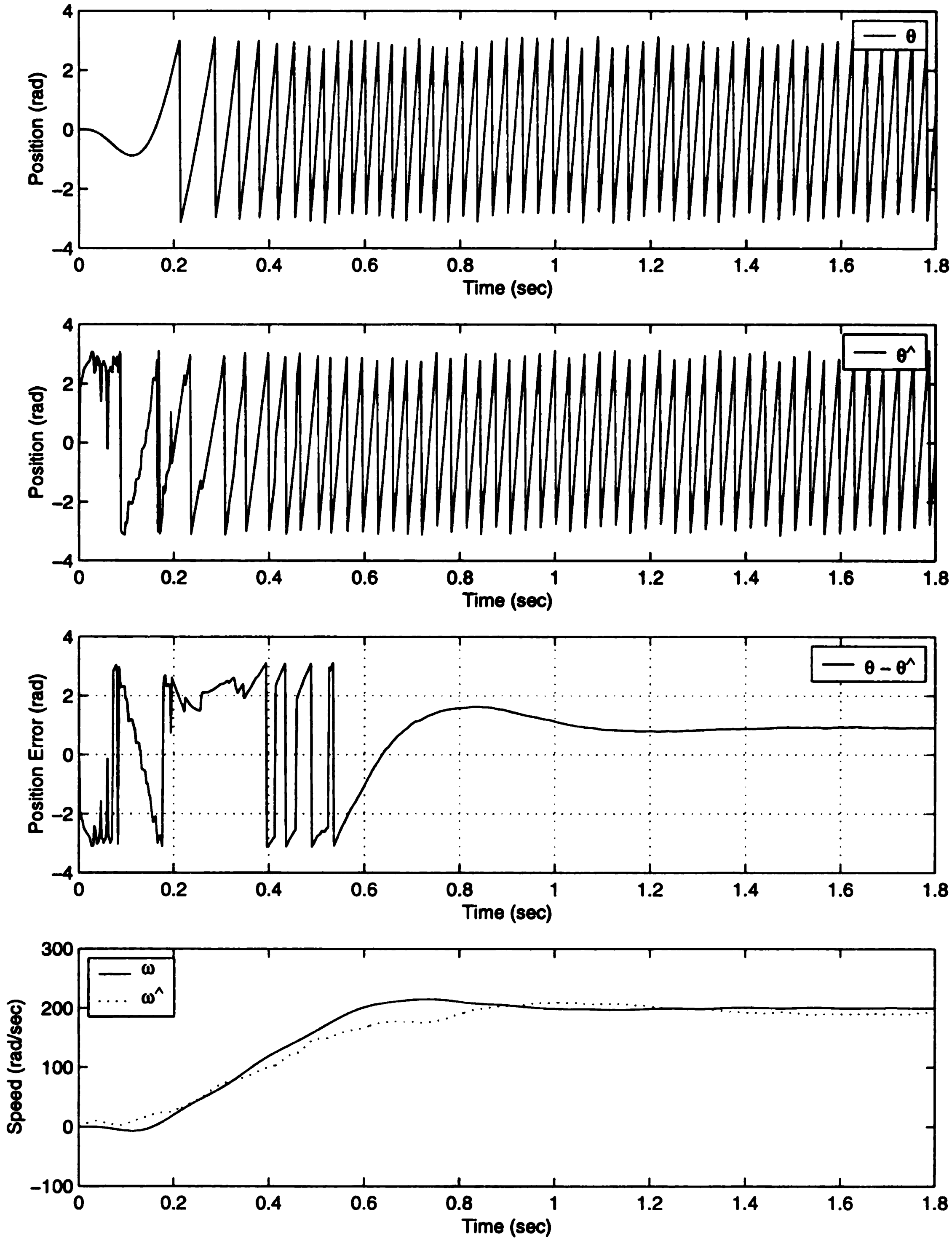

Figure 6.11. Experiment: ref speed: 600 RPM.

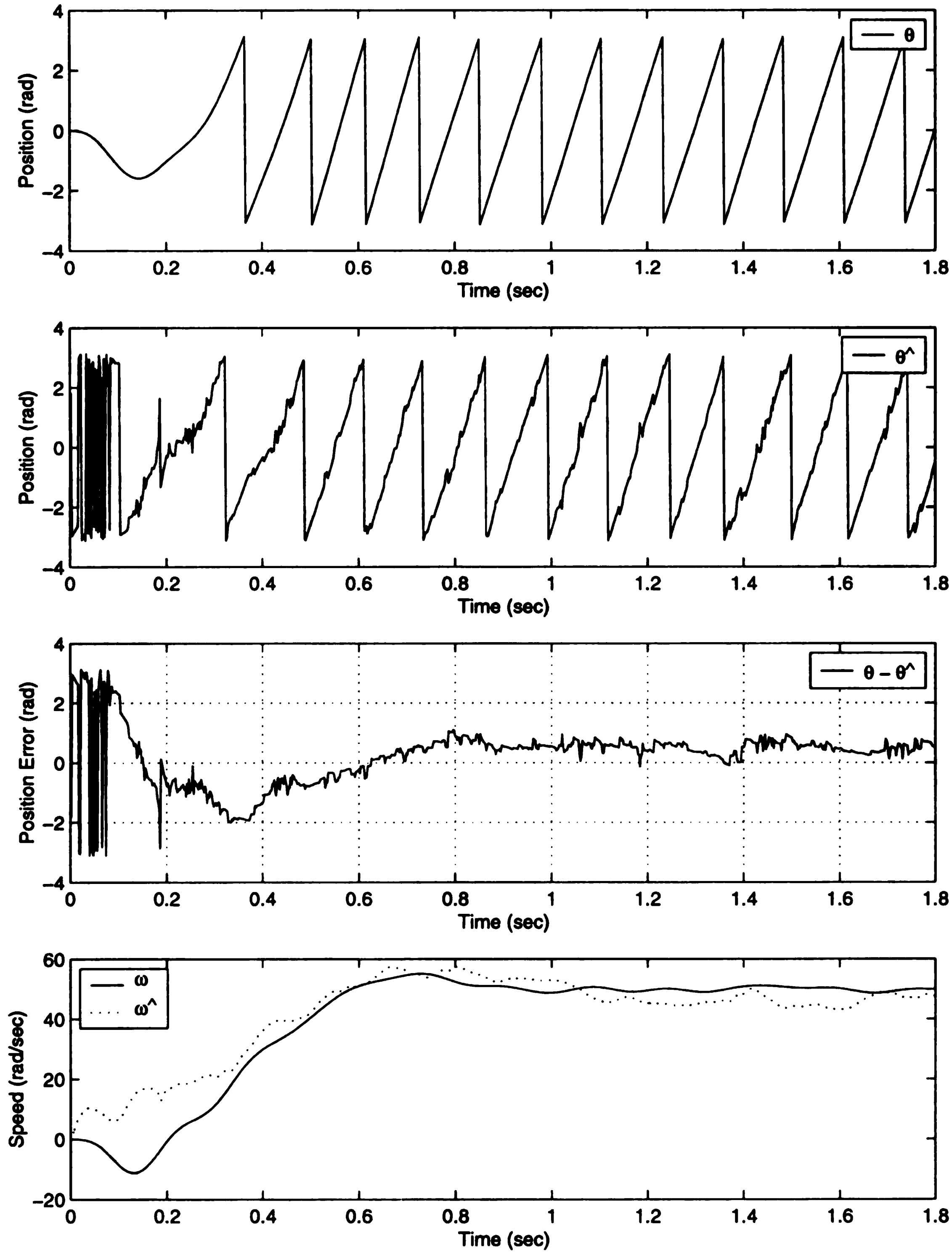

Figure 6.12. Experiment: ref speed: 150 RPM.

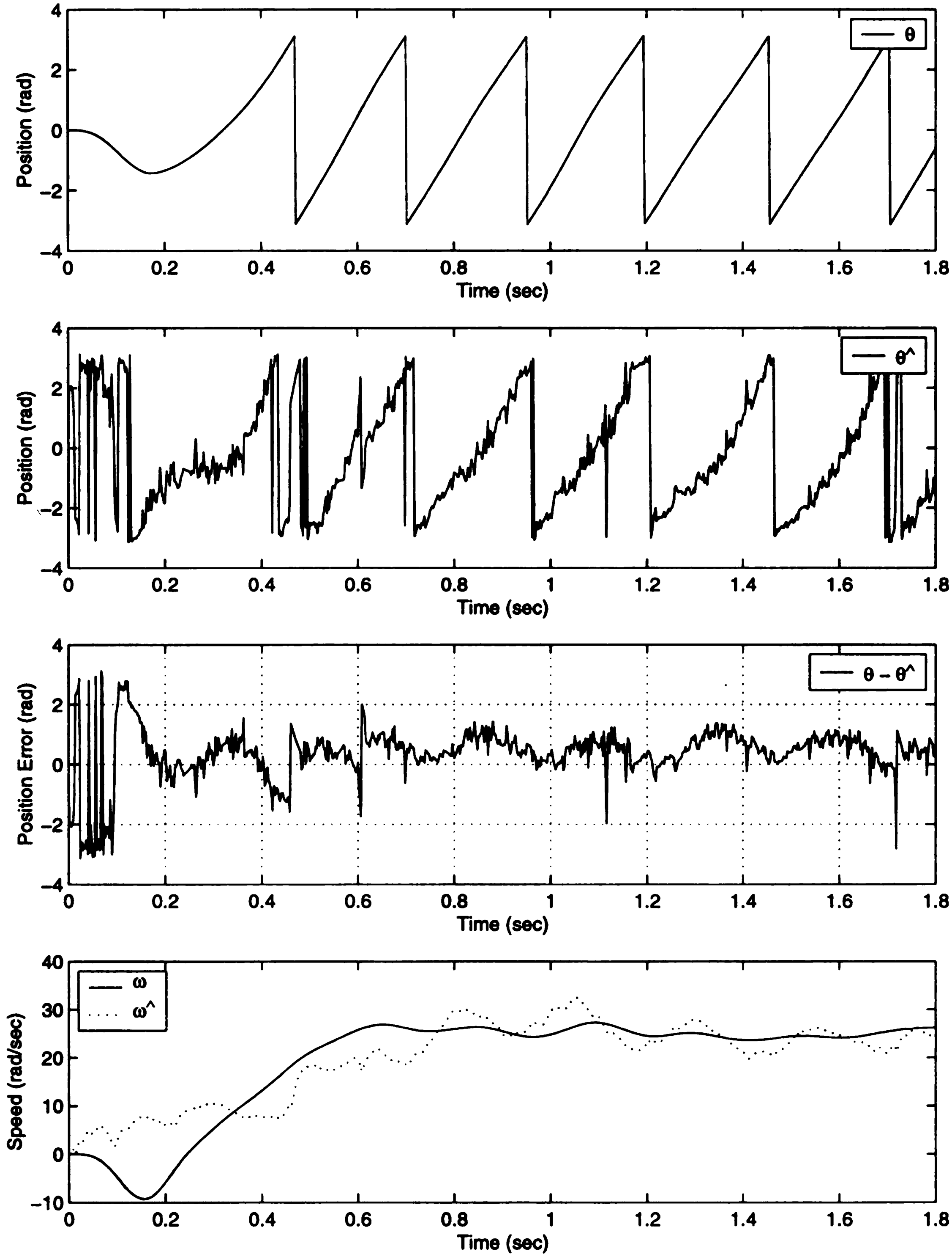

Figure 6.13. Experiment: ref speed: 80 RPM.

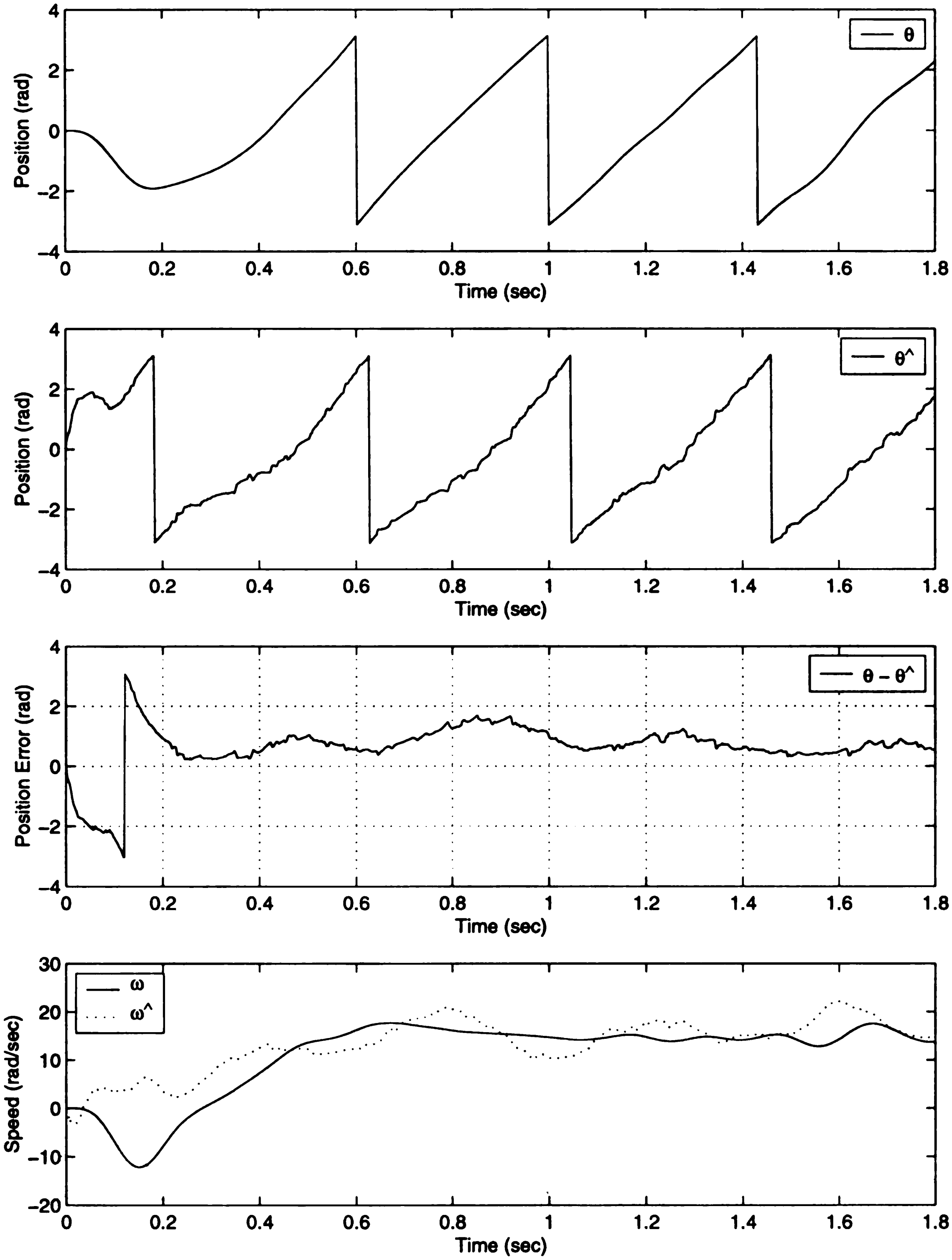

Figure 6.14. Experiment: ref speed: 45 RPM.

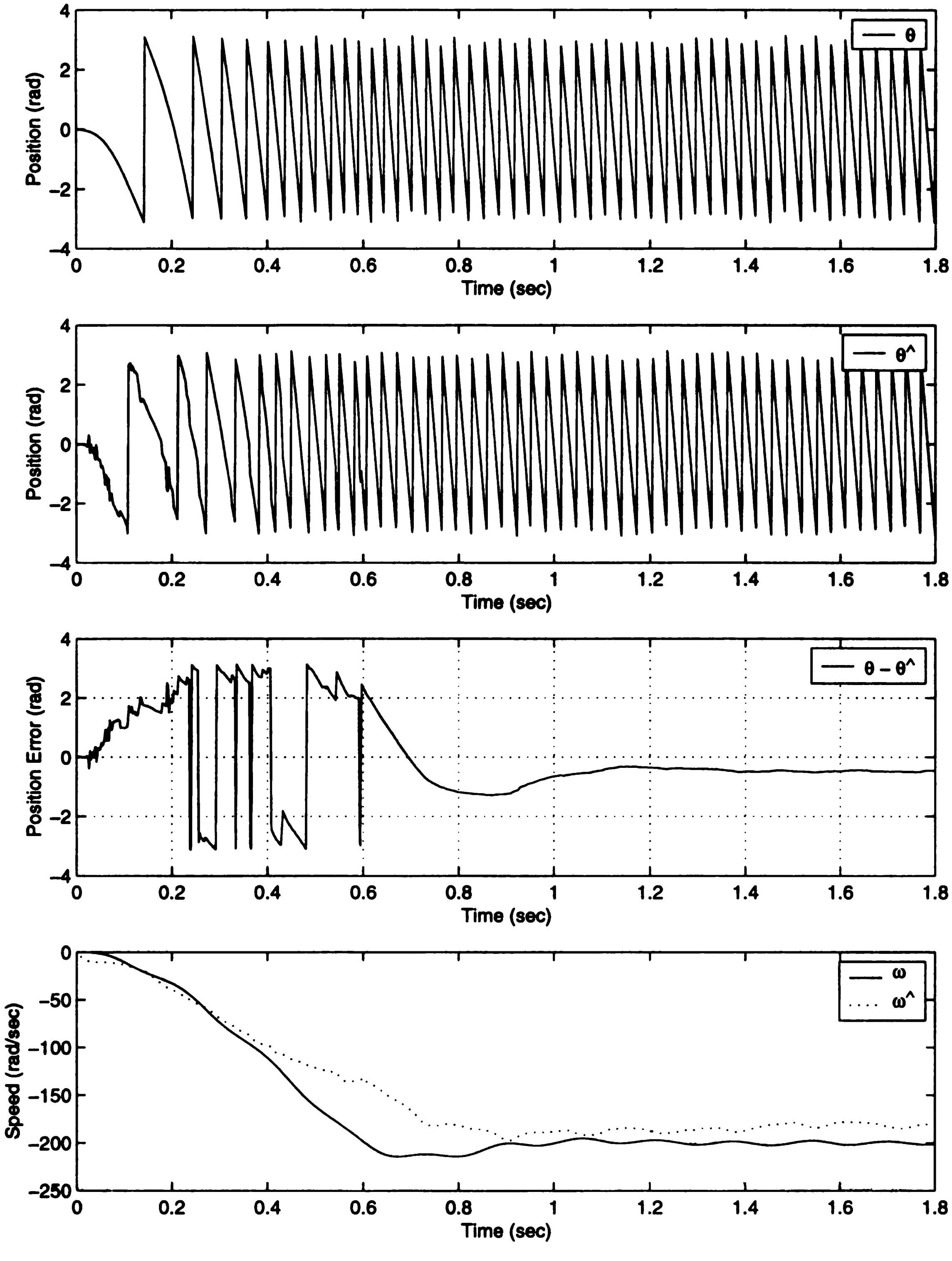

Figure 6.15. Experiment: ref speed: -600 RPM.

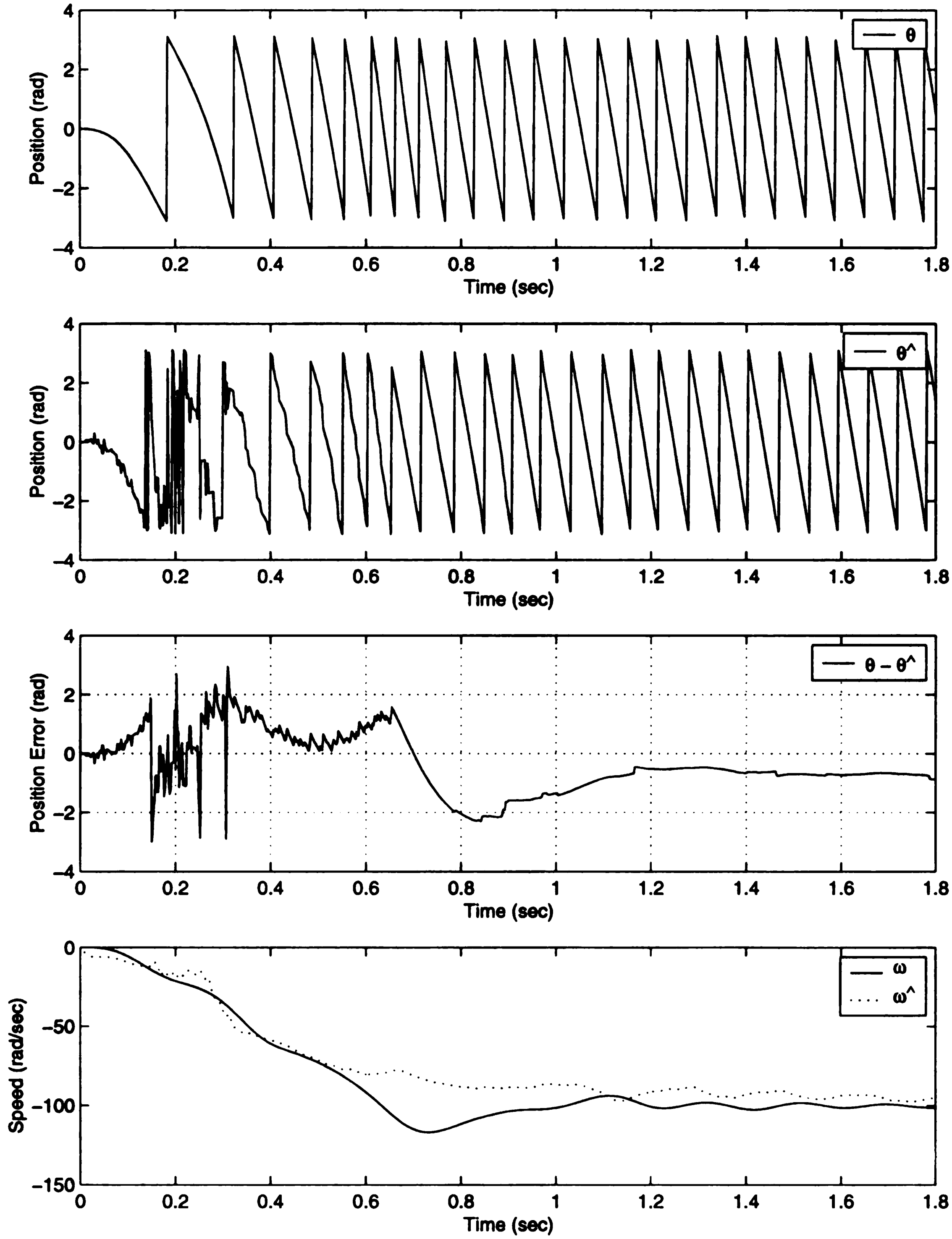

Figure 6.16. Experiment: ref speed: -300 RPM.

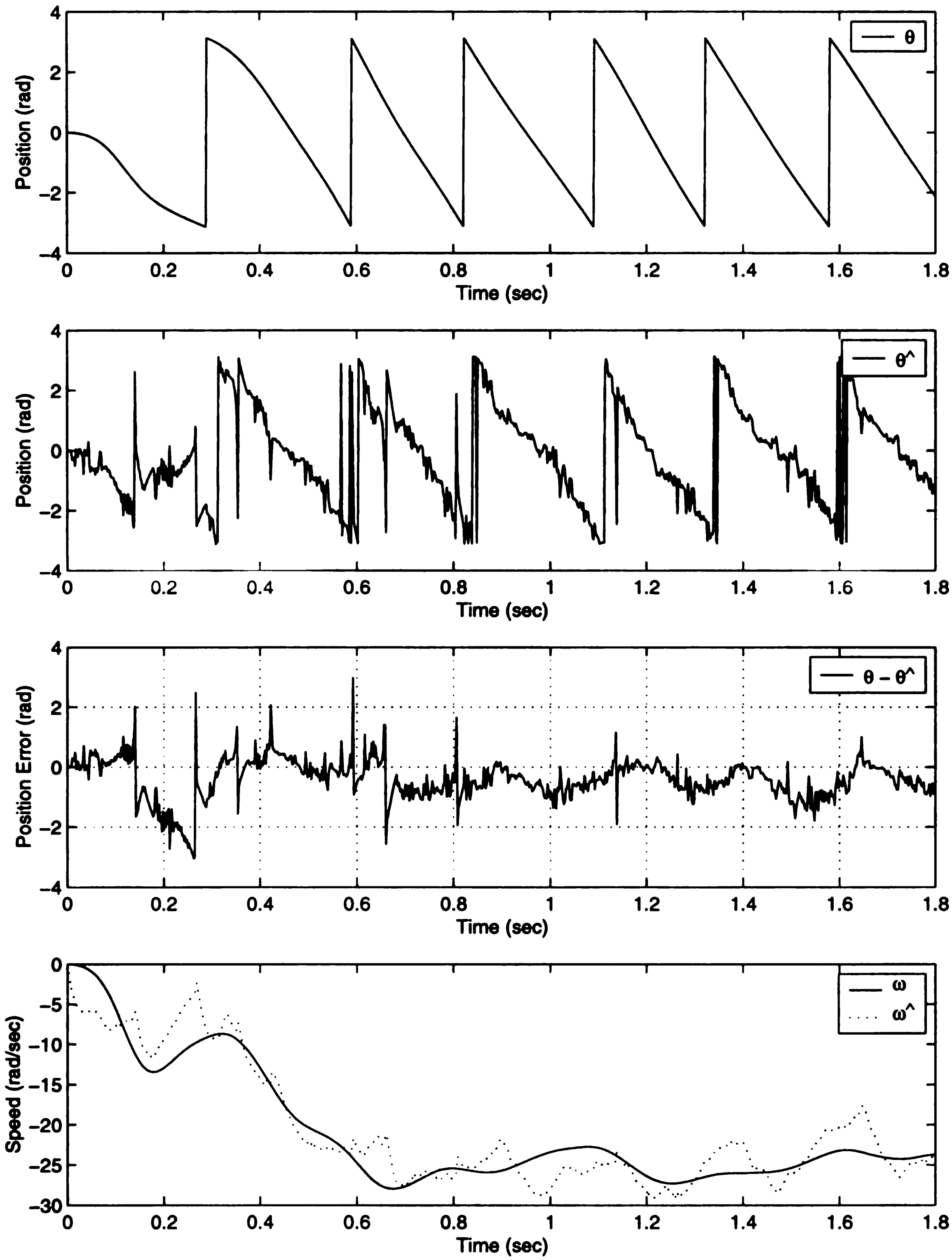

Figure 6.17. Experiment: ref speed: -75 RPM.

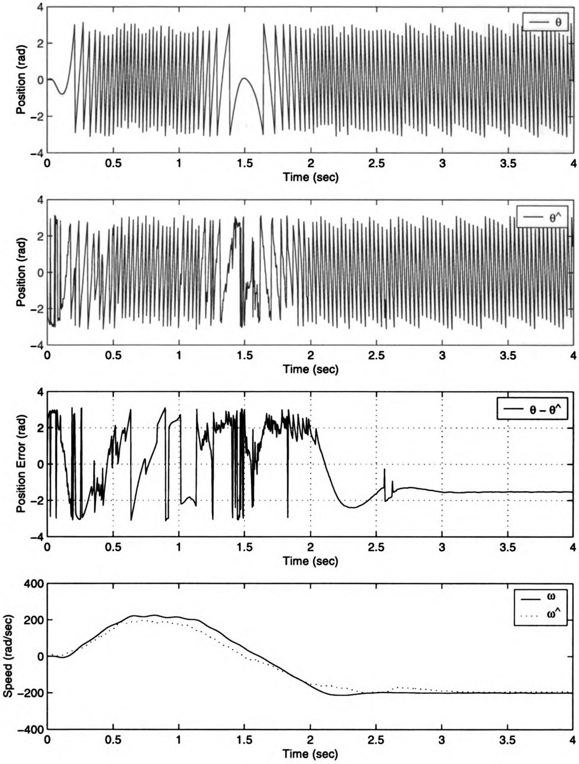

Figure 6.18. Experiment: ref speed: 600 and -600 RPM.

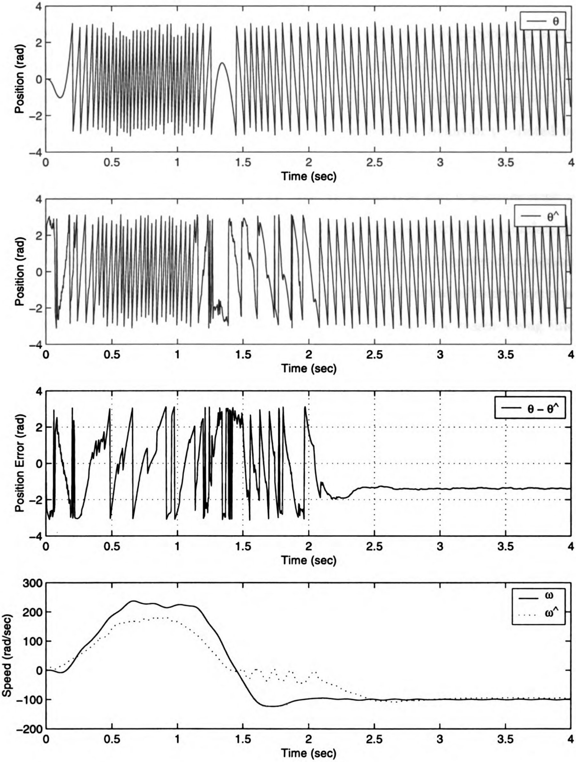

Figure 6.19. Experiment: ref speed: 600 and -300 RPM.

6.5 Discussion

As we have seen, the experimental results show the viability of the proposed scheme. When we compare the experimental results with the ones obtained from simulations, we notice some disparity. This disparity can be explained by taking a closer look at the various factors that contribute to this departure from the behavior predicted by simulations.

The main factor that causes disparity between simulation and implementation is the current hysteresis controller. Incorporating the hysteresis controller in simulations has proved impracticable. A simulation that runs within a few minutes of computer time with the assumption that the actual currents are equal to the reference currents, would take a few days if the hysteresis effect is incorporated. The reason being that at every single step of the simulation, the state of each of the three hysteresis current controllers changes thus significantly slowing down the simulation process. As seen in Figure 6.5, the behavior of the hysteresis current controller is far from ideal.

Another factor is the nonlinearity inherent in current and voltage sensors. While simulations assume that the values used in estimation are the actual ones, in practice, this cannot be realized perfectly. The sensors have been very carefully calibrated. A set of known voltages and currents was applied and the resulting data were plotted against the actual values. *Matlab* was used to find the coefficients of a linear polynomial that fit the data in a least-square sense. While this is an effective calibration technique, it cannot compensate for the nonlinearity inherent in the transducers.

Parameter detuning is another contributory factor. The parameter values change with temperature, for example that of resistance. While, values of inductance change as function of saturation.

This concludes our discussion of the experimental results obtained.

CHAPTER 7

Conclusions

7.1 Summary

In this dissertation, a promising scheme has been presented to estimate the position and speed of surface-mounted sinusoidal electromotive force (EMF) permanent magnet synchronous motors (PMSMs).

After introducing the PMSM vis-à-vis other contenders for servo-control, the first chapter discussed the need and the disadvantages of using sensors in the PMSM control. It indicated that vector control of PMSMs involves orienting the current space vector orthogonally to the rotor flux vector. This explains the need for sensing the rotor position. But position sensors increase the cost and lower the reliability of the drive: hence the research on sensorless operation.

The second chapter presented a review of the research done in the field over the last couple of decades. It developed the mathematical model of the PMSM. It presented the pros and cons of the major approaches proposed: back EMF, excitation monitoring, magnetic saliency, motor modification, and observers. It was noticed that observers provided the most promising and most generally applicable of all the approaches.

The third chapter presented the problem statement and the proposed methodology. The difference of current derivatives was presented as the means of estimating position and speed. Error in position estimate was shown to manifest itself in the difference of derivatives of the flux-producing components of stator current. The error in speed estimate was shown to show up as the difference of derivatives of the torque-producing components of stator current. This led to the expressions for position and speed estimates which were used in the rest of the work. All the intermediate derivations leading to the algorithm were presented in the concluding section of the chapter.

The fourth chapter presented results of numerical simulations of the proposed scheme: the estimates were shown to converge to the actual variables. Another observation was the large initial error allowable in position estimate, which would still result in stable estimation of position and speed. The concluding simulations explored the impact of parameter mismatch on convergence of the estimates. It was observed that the position estimate still converged, but speed estimate suffered significantly in the face of parameter detuning.

The convergence of the estimation scheme observed in numerical simulations was analytically proved in chapter five. It showed that the scheme consists of two uncoupled first-order high-gain observers, one each for position and speed. Each observer was driven by an independent measurement. This fact allowed us to study the stability of each observer independent of the other.

The sixth chapter presented results from a set of experiments carried out to investigate the viability of the proposed scheme. It detailed the experimental setup and also included a flowchart of the Assembly language code developed for the purpose.

7.2 Contribution

In this work, we have advanced the state of the art of position and speed estimation of permanent magnet synchronous motors. We have presented a novel scheme using current derivatives to do this estimation. The current derivatives are first calculated using high-gain observers and then using the motor model. The difference in the two results is used to drive the position and speed observers.

A systematic and analytical approach for developing the scheme has been presented. After developing the idea, we have validated it three different ways: first through numerical simulations, then with mathematical analysis and finally with experimentation. The numerical simulations have been carried out using *Matlab* and *Simulink*. The mathematical analysis part has proved stability of the estimator dynamics using the notion of high-gain observers. The experimental part consists of implementing the scheme with an off-the-shelf motor, using a digital signal processor. The observer shows promising results not only in the low speed region but also with speed reversal.

The scheme does not rely on the motor having salient poles. It does not involve physical modification such as placement of search coils. The scheme does not rely on integration of the speed estimate to get the position estimate. It works in a closed-loop mode i.e., it includes inherent correction mechanism. Also, it is computationally less intensive than many other schemes proposed. It does not require the knowledge of the mechanical load or the moment of inertia. All it needs is the knowledge of three motor parameters to do the estimation: its resistance, inductance and rotor flux linkage. These parameters are always included in the manufacturer's data sheets.

7.3 Future Work

A number of avenues can be explored to further analyze, simulate and implement the proposed technique. Some ideas for possible future work are discussed in this section.

7.3.1 Numerical Simulations

One important contribution would be to incorporate more of the phenomena observed at the implementation stage into numerical simulations. For instance, hysteresis current controller model; inverter switching delays; saturation effects; analog-to-digital converter quantization effects; parameter detuning effects. One such incorporation was done with the hysteresis current controller model. The resulting simulation was found to be too slow to be of use. With faster processors and larger memories becoming economical, such increasingly realistic simulations should become more feasible in future. A related idea would be to use integration algorithms, which are real-time implementable.

7.3.2 Mathematical Analysis

The analysis carried out here was based on linearization of the estimator dynamics. One possible research direction would be to carry out this analysis based on the nonlinear system theory. Also, while studying the effect of parameter mismatch, the analysis focused on the stator resistance value. One possible direction would be to explore the impact of variation in the other two parameters: stator inductance and rotor flux linkage. Also, it was shown that the position estimate was insensitive to resistance mismatch. The speed estimate, on the other hand, was seen to be directly affected by such a mismatch. One contribution would be to make the speed estimate also insensitive to such a mismatch. Another possibility would be to investigate the extension of these results to other AC motors.

7.3.3 Experimental Implementation

Further experiments can be conducted to bring the real-time implementation results closer to those predicted by numerical simulations. Given that the prototype has been built and tested, further work could be carried out without major effort with the setup.

Real-time implementation of a simulated algorithm often entails tuning of various filters, high-gain observers and controllers used. This is needed to account for the phenomena which were not modeled in simulations. In the experimental part of this work, we focused our tuning effort on ensuring that the position estimate converged to the right value. The speed estimation, being more important for the controller, was considered to have secondary significance. One possible improvement would be to ensure that the whole setup is tuned so that both the position and the speed estimates converged to the right values.

A related idea is to implement platform migration from the present DSP, to Real Time Linux with the objective of cutting down on development time and speeding up testing various settings. This approach should make it easier to focus on the main objectives rather than being caught up in Assembly language coding details. Another advantage of this idea would be the much larger memory available for storing various variables of interest and be able to run the experiments longer. The present setup is able to store 32,000 floating point variables hence limiting the length of experimental runs: while storing 32 such variables, at 10 kHz interrupt frequency, the experiment can be run for only 0.1s. The various plots included in the implementation results were obtained by staggering data storage across different interrupts.

Another possibility is to incorporate integration algorithms other than the Euler's used in the present setup. Also, the use of other advanced PWM techniques can be investigated: the present setup uses a hysteresis current controlled PWM inverter. Other approaches such as space vector based PWM could be investigated as alternatives.

One improvement would be to implement the scheme without using voltage sensors. In principle, the scheme can be implemented with just two current sensors. The knowledge of the pulse pattern generated by the hysteresis current controller and the DC bus voltage can be translated to the voltage applied to each phase.

APPENDIX

APPENDIX A

DSP Code

```
/*-----------------------------------------------------------------*
 *                                                                 *
 *     File:            estimator.s                                *
 *     Description:     Implements the proposed scheme             *
 *     Author:          Ali Khurram                                *
 *-----------------------------------------------------------------*/


#define     ivtp        r22e
#define     out         r21
#define     PortA       0x400000
#define     PortB       0x400008
#define     PortC       0x400010
#define     PortD       0x400018
#define     PControl    0x40001C
#define     SWReset     0x40001C
#define     Controla    0x400020
#define     Statusa     0x400020
#define     Counter     0x400024
#define     VADC0       0x400064
#define     VADC1       0x4000e4

/*-------------------------------------------------------------------*/

.rsect ".start"

      goto main
      nop

/*-------------------------------------------------------------------*/
```

```
.rsect ".table"              /* 800,000  l= 30  */

 itable:  2*nop
          goto 0x8004d4
          nop
          6*nop
          goto isr
          nop

/*----------------------------------------------------------*/

.rsect ".prog"              /*  10  l = 7f0 */
.align 4


/*-----------------------------------------------------------*
 *                                                           *
 *                    system parameters                      *
 *                                                           *
 *   (ad= 6/2^16 )*(lem = 1000)/(Rm =50.4) / (turns = 3)     *
 *             La =7.2e-3   abc:  L=1.5*La dq                *
 *-----------------------------------------------------------*/


      count_lim:     int      0x1000          /* start up */
      count_dummy:   int      0
      curr_conv:     float    6.0550750248e-4
      Ke:            float    0.0572
      L:             float    12e-3
      LoKe:          float    0.20979020979021
      oneoL:         float    83.3333333333333
      R:             float    6.0
      Rhat:          float    6.0

      c_speed:       float    200.0       /* the const speed   */
      v_speed:       float    400.0       /* the varbl speed   */
      t_inc:         float    200e-6      /* f_int: 5 kHz      */

      eid:           float    50.0        /*  hgo gains, d/dt   */
      eiq:           float    50.0
      eomega:        float    50.0
      eomegahat:     float    5.0         /*  for feed-back  */
      eth:           float    10.0        /*  msd th -> speed */
      ethhat:        float    10.0        /*  for feed-back  *
      pzone:         float     1.0        /* time to switch  */
```

```
/*----------------------------------------------------------*
 *                                                          *
 *            tuning parameters                             *
 *                                                          *
 *      [A,B,C,D]=butter(2,2*pi*fc,'s')                     *
 *    a21 = -a12; a22=0; b1= -a12; b2=0; c1=0; c2=1;  d=0;  *
 *----------------------------------------------------------*/

     i11:          float   -1332.86488144751  /* 150Hz lpfltr */
     i12:          float   -942.477796076938
     v11:          float   -1332.86488144751
     v12:          float   -942.477796076938

     fact:         float   0.01840776945463   /* 6*pi/1024 */
     hyst:         float   0.05               /* bandwidth */
     iqrm:         float   -1.5
     Ki:           float   4.0e-5
     Kp:           float   0.1
     offsetv0:     float   0.074364
     offsetv1:     float   -0.710401
     slopev0:      float   0.0094
     slopev1:      float   0.0095
     offsetia:     float   0.0077345
     offsetib:     float   -0.010243277
     slopeia:      float   -6.025112245e-4
     slopeib:      float   -6.069107457e-4
     t_start:      float   0.0        /* time to start      */
     t_stop:       float   2.0        /* time to stop       */
     store_start:  float   0.0        /* start data storage */
     store_stop:   float   2.0        /* stop   "   "  "  " */

/*______________________constants_________________________*/

     n_16:              float   16.0
     one:               float   1.0
     one_neg:           float   -1.0
     one_over_2_pi:     float   0.159154943092      /* 1/(2*pi) _bigsin */
     one_over_pi_by_2:  float   0.636619772366      /* 2/pi     _sin    */
     oneosqrt3:         float   0.57735026918963    /* 1/sqrt3  _dq     */
     one_third:         float   0.33333333333334    /* 1/3      _dq     */
     pi:                float   3.14159265359       /*  _sin        */
     pi_by_2:           float   1.5707963268        /*  _dq             */
     point_five:        float   0.5                 /*  wr              */
     point_five_neg:    float   -0.5                /*      iqr         */
     point_two:         float   0.2                 /*     _estimator   */
     two:               float   2.0                 /*      _dq         */
     two_pi_by_3:       float   2.094395102393      /*      _ref        */
```

```
      two_pi:          float   6.28318530718      /*      _sin        */
      zero:            float   0.000000

/*_________________________variables_________________________*/

      costhhat:     float   0.0
      delomega:     float   0.0
      i_c:          float   0.0   /* ic is reserved */
      ia:           float   0.0
      iaf:          float   0.0   /* x2 of the lpf */
      iar:          float   0.0
      iax1:         float   0.0
      ib:           float   0.0
      ib_f:         float   0.0   /* ibx2, ibf is reserved */
      ibr:          float   0.0
      ibx1:         float   0.0
      icr:          float   0.0
      id:           float   0.0
      idp:          float   0.0
      idpm:         float   0.0
      idf:          float   0.0
      iQ:           float   0.0
      iq:           float   0.0
      iqf:          float   0.0
      iqp:          float   0.0
      iqpm:         float   0.0
      iqr1:         float   0.0
      iqr2:         float   0.0
      iqr2p:        float   0.0
      iqr:          float   0.0
      _localV:      2*float 0.0
      loops:        float   0.0
      loops_temp:   float   0.0   /* intra interrupt wait loops */
      omega:        float   0.0
      omegaf:       float   0.0   /* filter (hgo) _filter */
      omegahat:     float   0.0
      omegahatf:    float   0.0   /* filtered omegahat */
      omegahatp:    float   0.0
      omegahatfsw:  float   0.0
      omegahatfsw_inv:float 0.0
      omegap:       float   0.0
      portcv:       float   0.0   /* the variable storing float(PortC) */
      portdv:       float   0.0
      portcv_old:   float   0.0   /* old value  - forward   */
      portdv_old:   float   0.0   /* old value  - reverse   */
      posnet:       float   0.0
      posn:         float   0.0
```

```
        posp:           float   0.0
        sign_w:         float   0.0
        sign_what:      float   0.0     /* to remedy wr+ w- scenario */
        sinthhat:       float   0.0
        t:              float   0.0
        th:             float   0.0
        thf:            float   0.0     /* th -> hgo -> omega  */
        thhat:          float   0.0
        thhatf:         float   0.0
        thhatp:         float   0.0
        thr:            float   0.0
        vba:            float   0.0     /*     ch0: green     */
        vbaf:           float   0.0
        vbax1:          float   0.0     /* x2 is the filtered */
        vca:            float   0.0     /*     ch1: white     */
        vcaf:           float   0.0
        vcax1:          float   0.0     /* x2 is the filtered */
        vd:             float   0.0
        vD:             float   0.0
        vQ:             float   0.0
        vq:             float   0.0
        wr:             float   0.0

/*_________________________ integers _________________________*/

        count:          int     0        /*    starting only    */
        flagstart:      int     0        /* start s/r open loop */
        one_int:        int     1
        twl:            int     0
/*__________________________________________________________*/

main:       call _initialize(r19)           /* the prog starts  */
            nop

end:        5*nop                           /* loops_temp++     */
            r1e = one
            r2e = loops_temp
            *r2 = a0 = *r2 + *r1            /* no ai used as ip */
            goto end
            nop

/*___________________________________________________________*
 *                                                           *
 *                      initialization                       *
 *___________________________________________________________*/
```

```
_initialize:

     ivtp = itable

     r7e  = count           /* int */
     *r7  = 0

     r7   = 0x4432
     pcw  = r7

     r8e  = 0x200008        /* interrupt frequency (Hz)     */
     r9e  = 0xf830          /* 20  k: fe0c; 10k : fc18      */
     *r8  = r9              /* 6.25k: f9c0;  5k : f830      */
                            /* 6.67k: fa24;  2k:  ec78      */
     r2e  = Controla        /* 1.0k:  d8f0;  dec2hex(55536) */
     r1   = 0x0000
     *r2  = r1
                            /*   0x400024 analog board   */
     r2e  = Counter         /* 20k: fe71, 10k: fce1, 5k: f9c1 */
     r1   = 0xf9c1          /*   interrupt freq */
     *r2  = r1              /* 1k: e0c1 */
     r13e = 0x80000         /* points to beginning of data buffer */

     r2e  = SWReset         /* read the SW reset register     */
     r1   = *r2
     r2e  = PControl        /* write into the PC Control reg  */
     r1   = 0x11            /* Ports single buffered */
     *r2  = r1
                            /* PortB: 0x400008 cntr set/reset */
     r2e  = PortB           /* sending H to reset the counter */
     r3   = 0xFF
     *r2  = r3              /* PortC,D: 0x400010,8 pulse count */
                            /* L to put it in the counter mode */
     r3   = 0x00
     *r2  = r3              /*        PortA: 0x400000          */
                            /* inverter switching pulse        */
     r3e  = PortA           /* open all the inverter switches  */
     out  = 0x00            /*        ...  01  01  01          */
     *r3  = out             /*        ...0 aa' bb' cc'    */
                            /*      to avoid 111: 110011       */
                            /*     top:     1 on; 0 off        */
                            /*    bottom:   0 on; 1 off        */
  return (r19)
  nop
```

```
/*------------------------------------------------------------*
 *                                                            *
 *                the interrupt service routine               *
 *------------------------------------------------------------*/

 isr:

     call _start(r19)                /*  starting s/r */
     nop

     call _loops(r19)               /* ensure enough time */
     nop

     call _store(r19)               /* store data: 0x80000 onwards */
     nop

     call _measure_v(r20)           /* voltage measurement */
     nop

     call _time(r19)                /* timing of the program */
     nop

     call _pos_speed_meas(r19)      /* position  */
     nop

     call _measure_i(r20)           /* called by _start as well */
     nop

     call _lowpass(r19)             /* filtering the signals sampled */
     nop

     call _dq(r19)
     nop

     call _idp(r19)
     nop

     call _estimator(r19)
     nop

     call _ref(r19)                 /* generate reference currents */
     nop

     call _hysteresis(r19)          /* hysteresis controller */
     nop

ireturn
```

```
/*------------------------------------------------------------------*
 *                                                                  *
 *                     start-up: open loop                          *
 *                                                                  *
 *          flagstart initially set to zero (not yet started)       *
 *           when set to one => started, so bypass this s/r;        *
 *          if ia within hysteresis band, keep applying 0x30        *
 *          else, apply 0x00 (open all) onto PortA                  *
 *           orient rotor along the as-axis, 0-deg position         *
 *                                                                  *
 *------------------------------------------------------------------*/

_start:

     r1e = flagstart            /* int */
     r2  = *r1

     r3e = one_int              /* r2 = r2-1 */
     r4  = *r3
     nop
     r4 - r2                    /* CA condition: no latency */

     if(ne) pcgoto not_started_yet
     nop

     return(r19)                /* bypass this s/r, motor started */
     nop                        /* execute the other subroutines */

/*----------------------------------------------------------------*/

not_started_yet:

      r6e= count                /* count++ */
      r3 = *r6
      nop
      r3 = r3 + 1
      *r6= r3

      r5e= count_lim
      r4 = *r5
      nop                   /* if count != count_lim */
      r3 - r4
      if(ne) pcgoto to_be_cont_s
      nop

      r3e =  PortA
```

```
    out =  0x00
    *r3 =  out

    r1e = flagstart   /*           flag_start <- 1          */
    r2e = one_int     /* count_lim reached: set  start flag  */
    r3  = *r2         /* to 1 & never come back to this  s/r */

    nop
    *r1 = r3

    return(r19)          /* bypass this s/r, execute others */
    nop

to_be_cont_s:

    call _measure_i(r20)  /*    ia, ib, i_c updated    */
    nop

                            /* hysteresis controller    */

    r3e = ia                  /* |ib| = |ic| = 0.5*|ia| */
    r6e = one                 /* so ia check enough     */
    r7e = one_neg             /* ia > 0, ib = ic < 0 KCL*/

    a0  = *r3                 /*      a0 has ia         */
    a2  = *r6                 /*   a2 has ceiling =  1.0 */
    a3  = *r7                 /* a3 has floor    = -1.0 */

                              /*   ceiling check     */
    a1 = a2 - a0              /* is ceiling - ia >= 0? */
    3*nop                     /* if  ia <= ceiling, safe */
    if(age) pcgoto cont2_s    /* no alarm, do floor check */
    nop

    pcgoto cont9_s         /* else, alarm, apply 00hex (cont9_s) */
    nop

cont2_s:                   /*       floor check                */

    a1 = a3 - a0           /* a1 = floor - ia; alt=less than 0  */
    3*nop                  /* if ia > floor, all safe           */
    if(alt) pcgoto cont3_s /* no alarm, keep applying 0x30      */
    nop

    pcgoto cont9_s         /* else, alarm, apply 00hex (cont9_s) */
    nop
```

```
                                /* write out the corresponding word */

cont3_s:                        /* no alarm: currents safe */

        out = 0x30              /* 110000, a:top on, b,c: top off */
        r3e = PortA             /* Vdc applied to Za + (Zb || Zc) */
        *r3 = out

        ireturn

cont9_s:                        /* alarm: current too big */

        out = 0x00              /* all top three off, bottom on */
        r3e = PortA             /* => effectively all off       */
        *r3 = out

        ireturn                 /* no more s/r be executed */

/*-----------------------------------------------------------------------*
 *                                                                       *
 *     store number of wait loops counted waiting: loopstemp -> loops   *
 *                                                                       *
 *    (development only, so don't incorporate into other s/rs)          *
 *-----------------------------------------------------------------------*/

_loops:  r1e = loops
         r2e = loops_temp
         r3e = zero
         *r1 = a0 = *r2            /* loops <- loopstemp */
         *r2 = a0 = *r3            /* loopstemp <- 0.0   */

         return(r19)
         nop

/*-------------------------------------------------------------------*
 *                                                                   *
 *        32,000 * 4 spaces/float = 128,000                          *
 *   _store: store data: 0x80000 onwards: 32000 floats  = 128 K      *
 *    data stored on every twl_th interrupt, and that too, only      *
 *      if      store_start < time < store_stop   (AND condition)    *
 *     total 1000 data-storing-interrupts permissible                *
 *     2-sec run: 20kHz -> 40,000 interrupts, store every 40th       *
 *     10kHz -> 20,000 ints, store every 20th, 5kH, every 10th       *
 *-------------------------------------------------------------------*/

_store:  r1e = t            /*    store if w_sta < t < store_stop     */
         r2e = store_start
```

```
    r3e = store_stop

    a0 = *r2 - *r1    /*      store_start - t        */
    3*nop             /*  to have optimal optimization of space  */
                      /* age: no storage at t=0, 32000 i/o 32032 */
    if(age) pcgoto sto_exit /* if t < store_start, don't store */
    nop               /* sto_A:= bypass this s/r, run others    */

    a1 = *r3 - *r1        /*      store_stop - t         */
    3*nop

    if(alt) pcgoto sto_exit   /* if t > store_stop, don't store */
    nop

    r1e = twl
    r2  = *r1
    nop                /*     needed else, 'using reg loaded */

    r2  = r2 - 1       /* in previous  */
    *r1 = r2          /* instruction' warning message shows up */

   if(pl) pcgoto sto_exit
    nop

r1e = t                     /*      -1-     */
*r13++ = a0 = *r1
r1e = iaf                   /*      -2-     */
*r13++ = a0 = *r1
r1e = iar                   /*      -3-     */
*r13++ = a0 = *r1
r1e = ib_f                  /*      -4-     */
*r13++ = a0 = *r1
r1e = ibr                   /*      -5-     */
*r13++ = a0 = *r1
r1e = id                    /*      -6-     */
*r13++ = a0 = *r1           /* iD is iaf    */
r1e = idp                   /*      -7-     */
*r13++ = a0 = *r1
r1e = idpm                  /*      -8-     */
*r13++ = a0 = *r1
r1e = idf                   /*      -9-     */
*r13++ = a0 = *r1
r1e = iq                    /*      -10-    */
*r13++ = a0 = *r1
r1e = iqf                   /*      -11-    */
*r13++ = a0 = *r1
r1e = iqp                   /*      -12-    */
```

```
*r13++ = a0 = *r1
r1e = iqpm                /*      -13-    */
*r13++ = a0 = *r1
r1e = iqr                 /*      -14-    */
*r13++ = a0 = *r1
r1e = omega               /*      -15-    */
*r13++ = a0 = *r1
r1e = omegaf              /*      -16-    */
*r13++ = a0 = *r1
r1e = omegahat            /*      -17-    */
*r13++ = a0 = *r1
r1e = omegahatf           /*      -18-    */
*r13++ = a0 = *r1
r1e = wr                  /*      -19-    */
*r13++ = a0 = *r1
r1e = th                  /*      -20-    */
*r13++ = a0 = *r1
r1e =  thf                /*      -21-    */
*r13++ = a0 = *r1
r1e = thhat               /*      -22-    */
*r13++ = a0 = *r1
r1e = thhatf              /*      -23-    */
*r13++ = a0 = *r1
r1e = thhatp              /*      -24-    */
*r13++ = a0 = *r1
r1e = thr                 /*      -25-    */
*r13++ = a0 = *r1
r1e = vbaf                /*      -26-    */
*r13++ = a0 = *r1
r1e = vcaf                /*      -27-    */
*r13++ = a0 = *r1
r1e = vd                  /*      -28-    */
*r13++ = a0 = *r1
r1e = vq                  /*      -29-    */
*r13++ = a0 = *r1
r1e = sign_w              /*      -30-    */
*r13++ = a0 = *r1
r1e = vca                 /*      -31-    */
*r13++ = a0 = *r1
r1e = vba                 /*      -32-    */
*r13++ = a0 = *r1


r1e = 9           /*   1=> skip 1 interrupt:      */
r2e = twl         /* store every 2nd interrupt:   */
*r2 = r1          /* t_stop*2 - 1 should be put?  */
                  /* store every 40th interrupt   */
```

```
                    /* 19 for 10kHz, 39 for 20kHz   */
                    /* N -> store every N+1th intpt */
sto_exit:  return(r19)  /* 9 for 5kHz, 1 for 1 kHz      */
           nop          /*   restore t_w_l              */

/*---------------------------------------------------------------*
 *                                                               *
 *              program timing: apply inputs only if             *
 *                     t_start < t < t_stop                      *
 *---------------------------------------------------------------*/

_time:
      r1e = t
      r2e = t_stop

      a1  = *r2 - *r1          /* t_stop - t ?>= 0 */

      r3e = t_inc              /* t_inc = 1/f_int  */

      *r1 = a0 = *r1 + *r3     /* t = t + t_inc    */

      nop                      /* if(age)'s L=3 being fulfilled    */
      if(age) pcgoto go_on     /* if t_stop >= t, run the other s/r */
      nop                      /* else 'game over': infinity loop   */

      r3e = PortA              /* the final infinity loop */
      out = 0x00               /*  game over      */
      *r3 = out

finish:   pcgoto finish
          nop

go_on:    r2e = t
          r4e = wr
          r5e = c_speed
          r7e = point_five
          r8e = v_speed
                                         /*     inc_speed:      */
          a0 = *r2 - *r7                 /*   a0 = t - 0.5      */
          3*nop

          if(age) pcgoto const_speed    /* if t - 0.5 >= 0      */
          nop                           /* if t >= 0.5, c_spd   */

          *r4 = a2 = *r8 * *r2          /* else, wr = v_speed*t  */
          3*nop
```

```
return(r19)
nop

const_speed:

        *r4 = a2 = *r5                  /* wr = c_speed   */
        3*nop                           /* need it !!!    */

return(r19)
nop

    /* ------------------------------------------------------- *
     *                                                         *
     *           rotor position measurement                    *
     *                                                         *
     * ------------------------------------------------------- */

_pos_speed_meas:

        r1e = portcv_old      /*      CCW-Motion  */
        r2e = portdv_old      /*      CW-Motion   */
        r3e = PortC           /*      a port name     */
        r4e = PortD
        r5e = n_16            /*  rollover correction */
        r7e = portcv          /*  a variable name     */
        r8e = portdv

        *r7 = a2 = float(*r3) /* a2, portcv 've PortC's float version */
        *r8 = a1 = float(*r4) /* a1, portdv 've PortD's float version */

                              /*  rollover processing           */
        a2 = a2 - *r1         /*   a2 = portcv - portcv_old     */
        3*nop                 /*          if(age) L=3           */

        if(age) pcgoto pos_1  /*   if new >= old, no rollover   */
        nop

        a2 = a2 + *r5         /* else, yes rollover, add 16.0 to diff */

pos_1:                        /*          a1 is ccw            */
      a1 = a1 - *r2          /*    a1 = portdv - portdv_old    */
      3*nop

      if(age) pcgoto pos_2    /* new >= old: no rollover, keep going */
      nop

      a1 = a1 + *r5        /*  yes rollover,add 16.0 to the diff  */
```

```
pos_2:                      /*      a2 is cw ?          */
      3*nop                 /*  if (a1 - a2)>0 then speed is cw ?  */
      a0 = a1 - a2          /* a0 = net position = portcv - portdv */
                            /*    a2 latency: L = 2, 'mult' input  */

      r5e = posnet
      r7e = posp
      r8e = posn

      *r5 = a0 = a0 + *r5       /*          posnet += posnet_new        */
      *r7 = a2 = a2 + *r7       /* posp   += rollover_processed_portcv */
      *r8 = a1 = a1 + *r8       /* posn   += rollover_processed_portdv */

      r5e = portcv
      r6e = portdv
                            /* book-keeping for the next interrupt */
      *r1 = a0 = *r5        /*         portcv_old <- portcv        */
      *r2 = a1 = *r6        /*         portdv_old <- portdv        */

      r2e = th              /*  posnet: float -> rad   */
      r3e = fact            /*      6*pi/1024          */
      r5e = posnet

      *r2 = a0 = *r5 * *r3      /*   a0 has the th value   */

      r1e = th                 /*  th -> hgo -> omega    */
      r2e = thf
      r3e = omega
      r4e = eth               /*    eth = 10.0   */

      call _hgo(r20)
      nop                     /* r20e = .+4     */

      a0 = *r3               /* to affect flags */

      r6e = sign_w
      r7e = one
      r8e = one_neg

      if(alt) pcgoto _minus
      nop

      *r6 = a2 = *r7

       pcgoto _nextw
       nop
```

```
_minus:   *r6 = a2 = *r8

_nextw:   r1e = omega           /* omega cleanup  */
      r2e = omegaf
      r3e = omegap
      r4e = eomega

      call _hgo(r20)
      nop                      /* r20e=.+4  */

return(r19)
nop


.rsect ".text"                 /* prog continues at 8006ac, l=7954 */

/*_________________________________________________________________*
 *                                                                 *
 *   lowpass : Low Pass Filter: 2nd order, Butterworth             *
 *_________________________________________________________________*/

_lowpass:
      r7e  = i11
      r8e  = i12

      r1e  = ia
      r3e  = iax1
      r4e  = iaf               /* same as iax2 */

      call _lpf(r20)
      r20e =. +4

      r1e  = ib
      r3e  = ibx1
      r4e  = ib_f

      call _lpf(r20)
      r20e =. +4

      r7e  = v11
      r8e  = v12

      r1e  = vba
      r3e  = vbax1
      r4e  = vbaf

      call _lpf(r20)
```

```
     r20e = .+4

     r1e  = vca
     r3e  = vcax1
     r4e  = vcaf

     call _lpf(r20)        /* uses same cut off freq as above */
     r20e = .+4

return(r19)
nop


_lpf:                      /*      x1dot formation      */
     a0 = *r4 - *r1        /*        x2 - u             */
     2*nop                 /*  a0 latency L=2, * input  */
     a1 = *r8 * a0         /*      a12 * ( x2 - u )     */
     a2 = a1 + *r7 * *r3   /*  x1dot = a12(x2-u) + a11x1*/
     a3 = -*r8 * *r3       /*  x2dot = -a12.x1          */

     r2e = t_inc           /*   a2, a3 used as mult input  */

     *r3 = a2 = *r3 + a2 * *r2 /*  x1=x1 + x1d . h      */
     *r4 = a3 = *r4 + a3 * *r2 /*  x2=x2 + x2d . h      */

return(r20)
nop

/*____________________________________________________________*
 *                                                            *
 *                      measure currents                      *
 *____________________________________________________________*/

_measure_i:

     r1e = 0x200000        /* channel a's A/D */
     a0  = float(*r1)

     r3e = offsetia
     r6e = ia
     r4e = slopeia

     *r6 = a1 = *r3 + a0 * *r4

     r1e = 0x200004        /* channel b's A/D */
     r2e = ib
```

```
        a3  = float(*r1)

        r3e = offsetib
        r4e = slopeib

        *r2 = a2 = *r3 + a3 * *r4

        r4e = i_c                  /*    a1 latency L=2, '*' input */
        *r4 = a3  = -a2 - a1       /*        storing i_c           */

return (r20)
nop

/*------------------------------------------------------------------*
 *                                                                  *
 *                  abc   ->  dq frame of reference                 *
 *                                                                  *
 *         iD = iaf                                                 *
 *         iQ = 1/sqrt(3) *( iaf  + 2*ib_f  )                       *
 *         vD =    1/3    *( -vbaf -   vcaf )                       *
 *         vQ = 1/sqrt(3) *(  vbaf -   vcaf )                       *
 *------------------------------------------------------------------*/


_dq:    r1e = iaf              /* iD = iaf */
        r4e = ib_f             /* iQ = 1/sqrt(3) *(iaf+2*ib_f) */
        r5e = two

        a1  = *r4 * *r5        /*       2ib_f            */
        a2  =  a1 + *r1        /*    iaf+2ib_f           */

        r6e = oneosqrt3
        r7e = iQ               /* a2 mult input, L=2       */

        *r7 = a3 = *r6 * a2    /* iQ = 1/sqrt(3)(iaf+2ib_f)    */
                               /*------------------------------*/

        r1e = vbaf             /* vD = 1/3*( -vbaf - vcaf) */
        r2e = vcaf

        a0  = -*r1 - *r2       /* -vbaf - vcaf */
        a2  =  *r1 - *r2       /*  vbaf - vcaf */

        r3e = one_third
        r4e = vD

        *r4 = a1 = a0 * *r3        /* a0 mult input, L=2   */
```

```
        r5e = vQ          /* vQ = 1/sqrt(3)*( vbaf -vcaf) */
        *r5 = a3 = *r6 * a2       /* a2 mult input, L=2   */


        r1e = thhatf            /* sin(thhat) */
        r2e = in

        *r2 = a0 = *r1

        call _bigsin(r18)
        r18e = .+4

        r1e = outs
        r2e = sinthhat

        *r2 = a0 = *r1

        r1e = thhatf          /* cos(thhat) = sin(thhat+pi/2) */
        r2e = in
        r3e = pi_by_2

        *r2 = a0 = *r1 + *r3      /* thhat + pi/2 */

        call _bigsin(r18)
        r18e = .+4

        r1e = outs
        r6e = costhhat
        *r6 = a0 = *r1
/*----------------------------------------------------------------*
 *                                                                *
 *             DQ to dq                                           *
 *                                                                *
 *     fd =  fD*cos(thhat) + fQ*sin(thhat)                        *
 *     fq = -fD*sin(thhat) + fQ*cos(thhat)                        *
 *                                                                *
 *----------------------------------------------------------------*/

        r1e = vQ
        r2e = vD
        r3e = vq
        r4e = vd

        r5e = sinthhat               /* r6e = costhhat    */
                                     /*   fd formation    */
```

```
        a0 = *r1 * *r5                /*   fQ*sin(thhat)   */
        *r4 = a1 = a0 + *r2 * *r6     /* " + fD*cos(thhat) */
                                      /*   fq formation    */
        a2 = *r1 * *r6                /*   fQ*cos(thhat)   */
        *r3 = a3 = a2 - *r2 * *r5     /* " - fD*sin(thhat) */

        r1e = iQ
        r2e = iaf                     /*     iD = iaf      */
        r3e = iq
        r4e = id
                                      /*   fd formation    */
        a0 = *r1 * *r5                /*   fQ*sin(thhat)   */
        *r4 = a1 = a0 + *r2 * *r6     /* " + fD*cos(thhat) */
                                      /*   fq formation    */
        a2 = *r1 * *r6                /*   fQ*cos(thhat)   */
        *r3 = a3 = a2 - *r2 * *r5     /* " - fD*sin(thhat) */

return (r19)
nop

/*-----------------------------------------------------------------*/

_idp:
        r1e = id           /*    input          */
        r2e = idf          /*     state 1       */
        r3e = idp          /* derivative, x2    */
        r4e = eid

        call _hgo(r20)
        r20e = .+4

_iqp:
        r1e = iq
        r2e = iqf
        r3e = iqp
        r4e = eiq

        call _hgo(r20)
        r20e = .+4

return (r19)
nop
      /* ------------------------------------------------------- *
       *                                                         *
       *                 high gain observer                      *
       *                                                         *
       *        x1dot   =  ud   +   [e*(u - uh)]                 *
```

```
     *        x2dot   =         e*[e*(u - uh)]            *
     *                                                    *
     *      usage: r1e = u   (input)                      *
     *             r2e = uh  (x1, the filtered output)    *
     *             r3e = ud  (x2, the reqd derivative)    *
     *             r4e = e   (gain: 1/eps)                *
     *                                                    *
     *          call _hgo(r20)                            *
     *          r20e = .+4                                *
     *___________________________________________________*/

_hgo: a0 = *r1 - *r2            /*         u - x1             */
      2*nop                     /*  a0 mult input, L=2        */

      a1 = a0 * *r4             /*       e(u - x1)            */
      a2 = a1 + *r3             /*  x2 + e(u - x1) = x1dot    */

      r5 = t_inc                /* a1, a2  latency, L=2, *input  */
      a1 = a1 * *r4             /*     e*e(u - x1) = x2dot    */
      nop

      *r2 = a2 = *r2 + a2 * *r5     /* uh = uh + x1dot * tinc    */
      *r3 = a3 = *r3 + a1 * *r5     /* ud = ud + x2dot * tinc    */

return(r20)
nop

/*___________________________________________________________*
 *                                                           *
 *                 ref:  calculate iar, ibr, icr             *
 *___________________________________________________________*/

_ref: r6e = delomega
      r7e = wr
      r3e = omegahatf

      *r6 = a1 = *r7 - *r3      /* delomega = wr - omega    */
                        /*    iqr= PI of delomega    */
      r8e = Kp          /*     iqr1 = Kp*delomega    */
      r9e = iqr1

      *r9 = a2 = a1 * *r8       /*      a1 mult input L=2   */

      r10e = Ki                 /*  iqr2p = Ki * delomega   */
      r11e = iqr2p

      *r11 = a3 = *r10 * *r6    /*   *r6 L=3, memory write-read */
```

```
     r12e = iqr2                /*  iqr2 = iqr2 + iqr2p * t_inc */
     r3e  = t_inc

     *r12 = a0 = *r12 + a3 * *r3   /*         a3 mult ip, L=2      */

     r4e = iqr                  /*        iqr = iqr1 + iqr2     */
     *r4 = a1 = + a0 + *r9

     nop                /* mult input lat = 2 */
     r1e = sign_w

     *r4 = a1 = a1 * *r1

     2*nop
     r5e = point_five

     if (alt) pcgoto _iqrneg
     nop

/*______iqr > 0: ensuring iqr never dips below +0.5______*/

     a0 = a1 - *r5
     3*nop

     if(age) pcgoto _next    /* do nothing if >= 0.5 */
     nop              /* else it could be 0.4 then put 0.5 */

     *r4 = a0 = *r5

 pcgoto _next
 nop

_iqrneg:  /* if it is -0.4, then put -0.5, not now */

     a0 = a1 + *r5
     2*nop
     r7e = point_five_neg

     if(ale) pcgoto _next
     nop

     *r4 = a0 =*r7

_next:

     r5e = one              /*       saturating the iqr     */
```

```
      r6e = one_neg

      a0  = *r4 - *r5       /*      is  iqr - 1.0 >= 0      */
      3*nop                 /*      is  iqr >= 1.0          */

      if(age) pcgoto hi_q   /*   if so, saturate it at 1.0  */
      nop                   /*  else, check for lower limit */

      a1 = *r6 - *r4        /*   is -1.0 - iqr >= 0 ?       */
      3*nop                 /*   -1.0 ?>= iqr,  iqr ?<= -1. */

      if(age) pcgoto lo_q   /*  if so, saturate it at -1.0    */
      nop                   /*  else, all clear, do next task */

      pcgoto clear_q
      nop

hi_q: *r4 = a0 = *r5        /* iqr =1, overwritten, if too big */

      pcgoto clear_q        /*     all clear, do next task     */
      nop

lo_q: *r4 = a0 = *r6        /* iqr=-1, overwritten, if smaller */


clear_q:  r11e = wr         /* thr = thr + wr * t_inc */
          r12e = thr

          a3 = *r11         /*    a3 has wr   */
          2*nop             /* a3 mult ip L=2 */

          *r12 = a0 = *r12 + a3 * *r3
          3*nop             /* r12 lat: 3 */

        r1e =t           /* after pzone sec, thr <- thhatf */
        r2e =pzone       /*   try reducing this time */

        a0  = *r1 - *r2
        3*nop
        if (alt) pcgoto _sless
        nop

        r12e = thr
        r1e  = thhatf
        *r12 = a0 = *r1

        r12e = iqrm
```

```
        r1e  = iqr
        *r12 = a0 = *r1


_sless:
      r2e = thr              /* ia* = -iq * sin(thhat) */
      r1e = in               /* inport of sin(x) */
      *r1 = a0 = *r2         /*  sin(thhat) */

      call _bigsin(r18)
      r18e = .+4             /* out has result */

      r1e = iar              /* *iq to get ia~ */
      r4e = outs
      r5e = iqrm             /* iqr or iqrm=-1  */

      *r1 = a1 = -*r5 * *r4 /*    iar = -iqrm * sin(thr) */



      r6e = thr              /* ibr = -iqr * sin(th-2*pi/3) */
      r7e = two_pi_by_3
      r1e = in               /*     argument of _bigsin()   */

      *r1 = a0 = *r6 - *r7  /*        in = thr - 2pi/3     */

      call _bigsin(r18)
      r18e = .+4

      r1e = outs                /* got the result */
      r9e = ibr
      r10e= iqrm
                              /*   *r1 L=3, memory write/read  */
      *r9 = a2 = -*r10 * *r1  /* ibr = a2 = -iqrm * sin(thr-2pi/3) */

      r1e = iar               /* icr = -iar - ibr  */
      r3e = icr

      *r3 = a2 = -a2 - *r1

return (r19)
nop

/*____________________________________________________________*
 *                                                            *
 *           hysteresis current controller                    *
 *____________________________________________________________*/
```

```
_hysteresis:
                                          /* phase a  */
      r5e = hyst
      r6e = iar

      a1  = *r5                         /* a1 = hyst  */
      a0  = *r6                        /* a0 = ia_tilde  */

      r3e = ia

      a2 = a0 + a1             /* a2 has ceiling = ia_tilde + hyst */
      a3 = a0 - a1             /* a3 has floor   = ia_tilde - hyst */

      a0 = *r3                          /* a0 has ia */


cont1:
      a1 = -a0 + a2                /*       is -ia + ceiling  >= 0 ?    */
      3*nop                        /*  ie   is       ceiling >= ia ?   */

      if(age) pcgoto cont2         /*  if  yes, do nothing              */
      nop                      /*  ..01 00 11 11 ; & 1 => b c unchanged*/
                               /* else, ia too big:open top switch: 0  */
      out = out & 0x4f         /* & close bottom sw.: 0: force 0: & 0  */



cont2:
      a1 = a3 - a0           /* a1 = floor - ia <0; alt=less than 0  */
      3*nop                  /* is floor < ia, if so, ia > floor     */

      if(alt) pcgoto cont3       /*          do nothing         */
      nop                    /* else, ia too low: close top switch: 1*/
                             /* & open bottom switch: send 1:force 1:|*/
      out = out | 0x30       /* ..00 11 00 00 ; | 0 => b c  unchanged*/

cont3:                         /*             phase b            */
                               /* r5 still points to location hyst */
      a1  = *r5                /*          a1 = hyst             */

      r4e = ibr                /*     r4 points to the reqd ib~   */
      a0  = *r4                /*         a0 = ib_tilde           */
                               /*      a1 'mult' ip, needs L=2        */
      a2  = a0 + a1            /*     a2 has ceiling = ib_tilde +hyst */
```

```
        a3  = a0 - a1          /*     a3 has floor   = ib_tilde -hyst */

        r3e = ib
        a0  = *r3              /*        a0 has ib       */

cont4:
        a1 = -a0 + a2          /* a2 L = 2 */
        3*nop

        if(age) pcgoto cont5      /* if ib <= ceiling, do nothing */
        nop

        out = out & 0x73       /* 01 11 00 11; & 1 => a c unchanged */


cont5:
        a1 = a3 - a0           /* a1 = floor - ib; alt=less than 0 */
        3*nop

        if(alt) pcgoto cont6      /* if ib > floor, do nothing */
        nop

        out = out | 0x0c       /* 00 00 11 00; | 0 => a c unchanged*/

                              /* r5 still points to location hyst */
cont6:
        a1  =  *r5             /* a1 = hyst  */
        r3e =  icr             /* phase c */
        a0  =  *r3             /* a0 = icr ictilde */
                            /* because a1 needs L=2, 'mult' ip */
        a2  =  a0 + a1         /* a2 has ceiling = icr + hyst  */
        a3  =  a0 - a1         /* a3 has floor   = icr - hyst  */

        r4e = i_c           /* ic: reserved name, so ic     */
        a0  = *r4           /* a0 has i_c */


cont7:
        a1 = -a0 + a2
        3*nop

        if(age) pcgoto cont8  /* if ic <= ceiling, do nothing */
        nop

        out = out & 0x7c       /*  01 11 11 00; & 1 => a b unchanged */
```

```
cont8:
      a1 = a3 - a0          /* a1 = floor - i_c; alt=less than 0 */
      3*nop                 /* a0 safest X field*/

      if(alt) pcgoto cont9  /* if i_c > floor, do nothing */
      nop

      out = out | 0x03      /* ... 00 00 11: | 0 => a b unchanged*/

                            /* write out the corresponding word */
cont9:
      r3e = PortA           /*writing the 6 bits out*/
      *r3 = out

return (r19)
nop

/*________________________bigsin(x)________r18 ______________________*
 *                                                                   *
 *        the subroutine  to compute the sin of any number           *
 *                                                                   *
 *        usage: the variable in has the argument                    *
 *               the variable outs has the result                    *
 *-------------------------------------------------------------------*/

 _sinA:   float 0.2601903036e-5, -0.198074182e-3
          float 0.8333025139e-2, -0.166665668

      in:   float   0.0
      outs: float   0.0

      neg:  int 0
      n:    int     0
      nn:   int 0
      abc:  int 4

/*____________________________________________________*
 *                                                    *
 *      absolute value function simulation            *
 *                                                    *
 *     find if x < 0, if so set a variable            *
 *     sign to be negative; take absolute value       *
 *     of x, proceed with the algorithm, and at       *
 * the end, negate the result because sin(-x)=-sin(x) *
 *----------------------------------------------------*/
```

```
_bigsin:
      r1e = in
      a0  = *r1             /* flags affected */

      r1e = neg         /* new portion: x o/s range */

      2*nop             /* new, L=3     */

      if(age) pcgoto pos
      nop

      a0  = -a0         /*if x<0, negate it*/
      r2  = 1
      *r1 = r2

      pcgoto hoo
      nop

pos:  r2  = 0
      *r1 = r2

hoo:  dauc = 0x10                   /* truncation i/o rounding */
      r1e  = n
      r2e  = one_over_2_pi

      a1   = a0 * *r2               /* n=x/360:get no. of cycles  */
      *r1  = a2 = int(a1)           /* ceiling or floor: rounding */

      r3e  = two_pi
      2*nop                        /* latency=3 for memory write */

      a2 = float(*r1)
      2*nop                         /* latency=2 for acc mult ip */

      a0 = a0 - a2 * *r3            /* x = x - n*360 */

      r3e = nn
      r4e = one_over_pi_by_2       /* a0 L=2 mult input */

      a1  = a0 * *r4               /* quadrant no: x/90 */
      *r3 = a2 = int(a1)           /*     rounding     */
      3*nop

      r4 = *r3                     /* *r3 L=3 mem write/read */

quad_1:
      r5 = 0
```

```
        r5 - r4
        if(ne) pcgoto quad_4
        nop

        call _sin(r20)
        r20e = .+4

        pcgoto minus
        nop

quad_4:
        r5 = 3                    /* because of truncation: floor() */
        r5 - r4
        if(ne) pcgoto quad_23
        nop

        r3e = two_pi
        a0  = a0 - *r3            /*x=x-360 use the alg directly*/

        call _sin(r20)
        r20e =.+4

        pcgoto minus
        nop

quad_23:
        r3e = pi
        a0  = a0 - *r3

        call _sin(r20)
        r20e = .+4

        a0 = -a0                 /*negate the result */

minus:
        r1e = neg                /*if x<0, negate the result */
        r2  = 1
        r3  = *r1
        2*nop

        r2 - r3

        if(ne) pcgoto quad_5
        nop

        a0 = -a0
```

```
quad_5:
     r1e = outs
     *r1 = a0 = a0

return (r18)
nop

/*-----------------------------------------------------------------*/

_sin:
          r2e = _sinA
          a2  = a0 * a0

          2*nop

          a3 = a2 * a0
          a2 = *r2++ + a2* *r2++
          a1 = *r2++ + a2* *r2++
          a3 = a3 * a0
          a2 = a2 * a3
          a1 = a0 + a1 * a3
          2*nop

          a0 = a1 + a2 * a3
          nop

return (r20)
nop

/*-----------------------------------------------------------------*/

_measure_v:

      r1e = VADC0
      r2e = vba

      a0  = float(*r1)

      r3e = offsetv0
      r4e = slopev0

      *r2 = a1 = *r3 + a0 * *r4

      r1e = VADC1
      r2e = vca

      a1  = float(*r1)
```

```
      r3e = offsetv1
      r4e = slopev1

      *r2 = a1 = *r3 + a1 * *r4

      return(r20)
      nop

/*________________ inverse of a no.:   a0=1/a0 __________________*/

_inv:
      a2   = seed(a0)
      r14e = inv_A
      nop

      a0  = *r14++ - a0 * a2
      a1  = a2 * *r14++
      a2  = *r14++ - a0 * a2
      a0  = *r14++ + a0 * a0
      a1  = a0 * a1
      a2  = a2 * a1
      nop
      a0  = a2 + a0 * a1

return(r18)
nop

inv_A:  float  1.4074074347,    0.81
        float  2.27424702,     -0.263374728

/*_______________________________________________________________*
*                                                                *
*               estimator: speed/position                        *
*                                                                *
* iqpm=(vq-10.8e-3*omegahatf*id-Rc*iq-0.0572*omegahatf)/10.8e-3 *
*   idpm=(vd+10.8e-3*omegahatf*iq-Rc*id)/10.8e-3                *
*   deliqp=iqp-iqpm                                              *
*   delidp=idp-idpm                                              *
*   omegahat=omegahatf-0.1888*delpiq                             *
*                                                                *
*   if omegahat>=0.2, omegahatfsw=omegahatf                      *
*   else omegahatfsw=0.5                                         *
*   end                                                          *
*                                                                *
*   thhat   =thhatf+0.1888*delpid/omegahatfsw                    *
*_______________________________________________________________*/
```

```
_estimator:

    r1e = omegahat           /* estimate from last iteration    */
    r2e = omegahatf          /* not overwritten since then      */
    r3e = omegahatp
    r4e = eomegahat

    call _hgo(r20)
    r20e = .+4

    r1e = thhat
    r2e = thhatf
    r3e = thhatp
    r4e = ethhat

    call _hgo(r20)
    r20e = .+4

/*-----------------------------------------------------------------*/

                        /* iqpm=([vq-Rhat*iq]-[Ke+L*id]*omegahatf)/L */
    r1e = Ke
    r2e = L
    r3e = id

    a0 = *r1                 /* a0 = Ke        */
    a0 = a0 + *r2 * *r3      /* a0 = Ke + L id */

    r4e = vq
    r5e = Rhat
    r6e = iq

    a1 = *r4                     /* a1 = vq */
    a1 = a1 - *r5 * *r6          /* a1 = vq - Rhat * iq */

    r7e = omegahatf          /* to ensure iqp = iqpm */
    a0  = a1 - a0 * *r7      /* a0 = [vq-Rhat*iq]-[Ke+Lid]*omegahatf */

    r8e = oneoL
    r9e = iqpm                     /* a0 L = 2 */

    *r9 = a0 = a0 * *r8 /* iqpm=([vq-Rhat*iq]-[Ke+Lid]*omegahatf)/L */
```

```
                        /* deliqp = iqp - iqpm  */
    r4e = iqp          /* no need of saving deliqp */

    a0 = -a0 + *r4      /* a0 = deliqp = 0, if iqp = iqpm */

                    /* omegahat = omegahatf -  deliqp * (L/Ke) */
    r7e = omegahatf

    r1e = omegahat      /* if dont put r7e=, subtract from omegaf! */
    r4e = LoKe          /* L over Ke */

    *r1 = a0 = *r7 - a0 * *r4   /* a0 L = 2 */

    r1e = point_two       /* omegahatfsw */

    a1 = a0 - *r1         /* is omegahat - 0.2 >= 0 ? */
                          /* is omeghahat >= 0.2 ?    */
    r1e = omegahatfsw
    r4e = point_five
    nop

    if(age) pcgoto actual     /* if so, use it else, use 0.5 */
    nop

    *r1 = a0 = *r4            /* omegahatfsw = 0.5 */

    pcgoto go_on2
    nop

actual: *r1 = a0 = a0         /* ensure enough nops before _inv */


go_on2:
    call _inv(r18)        /* a0 = 1/a0 */
    r18e=.+4              /* argument, result = a0 */

    r1e = omegahatfsw_inv
    *r1 = a0 = a0         /* a0 to be preserved for use l.o. */

                /*_____id, idp, idpm_____________________*/

                /* idpm=(vd+L*omegahatf*iq-Rhat*id)/L */

    a1 = *r2 * *r7        /* a1 = L*omegahatf */

    r1e = vd
    nop
```

```
    a1 =  a1 * *r6            /* a1 = L*omegahtf*iq */

    a1 =  a1 - *r5 * *r3      /* a1 = L*omegahatf*iq - Rhat*id */

    a1 =  a1 + *r1            /* a2 = vd+L*omegahatf*iq-Rhat*id */

    r1e = idpm
    r5e = idp

    *r1 = a1 = a1 * *r8       /* a1 = idpm = " * 1/L */

    a1 = -a1 + *r5            /* a1 = del_idp = idp - idpm */

    r1e = thhat
    r2e = thhatf              /* r4 points to LoKe */

    a1 = a1 * a0              /* a1 = del_idp / omegahatfsw */

    nop
    r4e = LoKe
                         /* thhat=thhatf+(del_idp/omegahatfsw)*L/Ke */
    *r1 = a1 = *r2 + a1 * *r4

return (r19)
nop
```

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] P. P. Acarnley, R. J. Hill, and C. W. Hooper. Detection of rotor position in stepping and switched reluctance motors by monitoring of current waveform. *IEEE Transactions on Industrial Electronics*, 32:215–222, 1985.

[2] M. S. Arefeen, M. Ehsani, and T. A. Lipo. Sensorless position measurement in synchronous reluctance motor. *IEEE Transactions on Power Electronics*, 9:624–630, 1994.

[3] J. P. M. Bahlmann. A full-wave motor drive IC based on the back EMF sensing principle. *IEEE Transactions on Consumer Electronics*, 35:415–420, 1989.

[4] K. J. Binns, K. M. Al-Aubidy, and D. W. Simmin. Implicit rotor position sensing using search coils for self-commutating permanent magnet motors. *Proc. Inst. Elect. Eng., pt. B.*, 137:253–258, 1990.

[5] A. Consoli, S. Musumeci, A Raciti, and A. Testa. Sensorless vector control of brushless motor drives. *IEEE Transactions on Industrial Electronics*, 41:91–95, 1994.

[6] A. Dabroom. Output feedback sampled-data control of nonlinear systems using high-gain observers. *Ph.D. Dissertation, Michigan State University*, 20–22, 2000.

[7] R. Dhaouadi, N. Mohan, and L. Norumu. Design and implementation of an extended Kalman filter for the state estimation of a permanent magnet synchronous motor. *IEEE Transactions on Power Electronics*, 6:491–497, 1991.

[8] M. Ehsani and I. Husain. Rotor sensing in switched reluctance motor drives by measuring mutually induced voltages. *IEEE Industry Applications Society Annual Meeting Conf. Rec., Houston, TX*, pages 422–429, 1992.

[9] F. Esfandiari and H. K. Khalil. Output feedback stabilization of fully linearizable systems. *International Journal of Control*, 56:1007–1037, 1992.

[10] C. French and P. Acarnley. Control of permanent magnet motor drives using a new position estimation technique. *IEEE Transactions on Industry Applications*, 32:1089–1097, 1996.

[11] T. Furuhashi, S. Sangwongwanich, and S. Okuma. A position-and-velocity sensorless control for brushless DC motors using an adaptive sliding mode observer. *IEEE Transactions on Industry Applications*, 28:89–95, 1992.

[12] D. C. Hanselman. Resolver signal requirements for high accuracy resolver-to-digital conversion. *IEEE Transactions on Industrial Electronics*, 37:556–561, 1990.

[13] W. D. Harris and J. H. Lang. A simple motion estimator for variable reluctance motors. *IEEE Transactions on Industry Applications*, 26:237–243, 1990.

[14] J. Holtz. Pulsewidth Modulation for Electronic Power Conversion. *Proceedings of the IEEE*, 82:1194–1214, 1994.

[15] Jun Hu and Bin Wu. New integration algorithms for estimating motor flux over a wide speed range. *IEEE Transactions on Power Electronics*, 13:969–977, 1998.

[16] K. Iizuka, H. Uzuhashi, M. Kano, T. Endo, and K. Mohri. Microcomputer control for sensorless brushless motor. *IEEE Transactions on Industry Applications*, 21:595–601, 1985.

[17] A. Isidori. *Nonlinear Control Systems.* Springer-Verlag, 1995.

[18] H. K. Khalil, E. G. Strangas and J.M. Miller. A torque controller for induction motors without rotor position sensors. *ICEM 96, Vigo, Spain.*, September 1996.

[19] H. K. Khalil. *Nonlinear Systems.* Prentice Hall, 1996.

[20] H. K. Khalil. High-gain observers in nonlinear feedback control. *New Directions in Nonlinear Observer Design: Lecture Notes in Control and Information Sciences,* H. Nijmeijer and T.I. Fossen, editors, 244:249–268, 1999.

[21] J. S. Kim and S. Sul. High peformance PMSM drives without rotational position sensors using reduced order observer. *IEEE IAS Conf. Rec.*, pages 75–82, 1995.

[22] P. C. Krause, O Wasynczuk, S. D. Sudhoff. *Analysis of Electric Machinery.* IEEE Press, 1995.

[23] A. B. Kulkarni and M. Ehsani. A novel position sensor elimination technique for the interior permanent magnet synchronous motor drive. *IEEE Transactions on Industry Applications*, 28:144–150, 1992.

[24] W. Leonhard. *Control of Electrical Drives.* Springer, 1996.

[25] N. Matsui and M. Shigyo. Brushless DC motor control without position and speed sensors. *IEEE Transactions on Industry Applications*, 28:120–127, 1992.

[26] T. J. Miller *Brushless Permanent-Magnet and Reluctance Motor Drives.* Oxford University Press, 1989.

[27] M. Naidu and B. K. Bose. Rotor position estimation scheme of a permanent magnet synchronous machine for high performance variable speed drive. *IEEE Industry Applications Society Annual Meeting Conference Record, Houston, TX*, 28:48–53, 1992.

[28] S. Ogasawara and H. Akagi. An approach to position sensorless drive for brushless DC motors. *IEEE Transactions on Industry Applications*, 27:928–933, 1991.

[29] P. Pillay and Krihsnan R. Modeling, simulation, and analysis of permanent magnet motor drives, Part I: The permanent magnet synchronous motor drive. *IEEE Transactions on Industry Applications*, 25:265–273, 1989.

[30] Powerex. *IGBTMOD and Intellimod - Intellgent Power Modules: Applications and Technical Data Book.* Powerex, Inc., 1994.

[31] P. Vas. *Electrical machines and drives: a space-vector theory approach.* Oxford University Press, 1993.

[32] P. Vas. *Sensorless vector and direct torque control.* Oxford University Press, 1998.

[33] R. Wu and G. R. Slemon. A permanent magnet motor drive without a shaft sensor. *IEEE Transactions on Industry Applications*, 27:1005–1011, 1991.