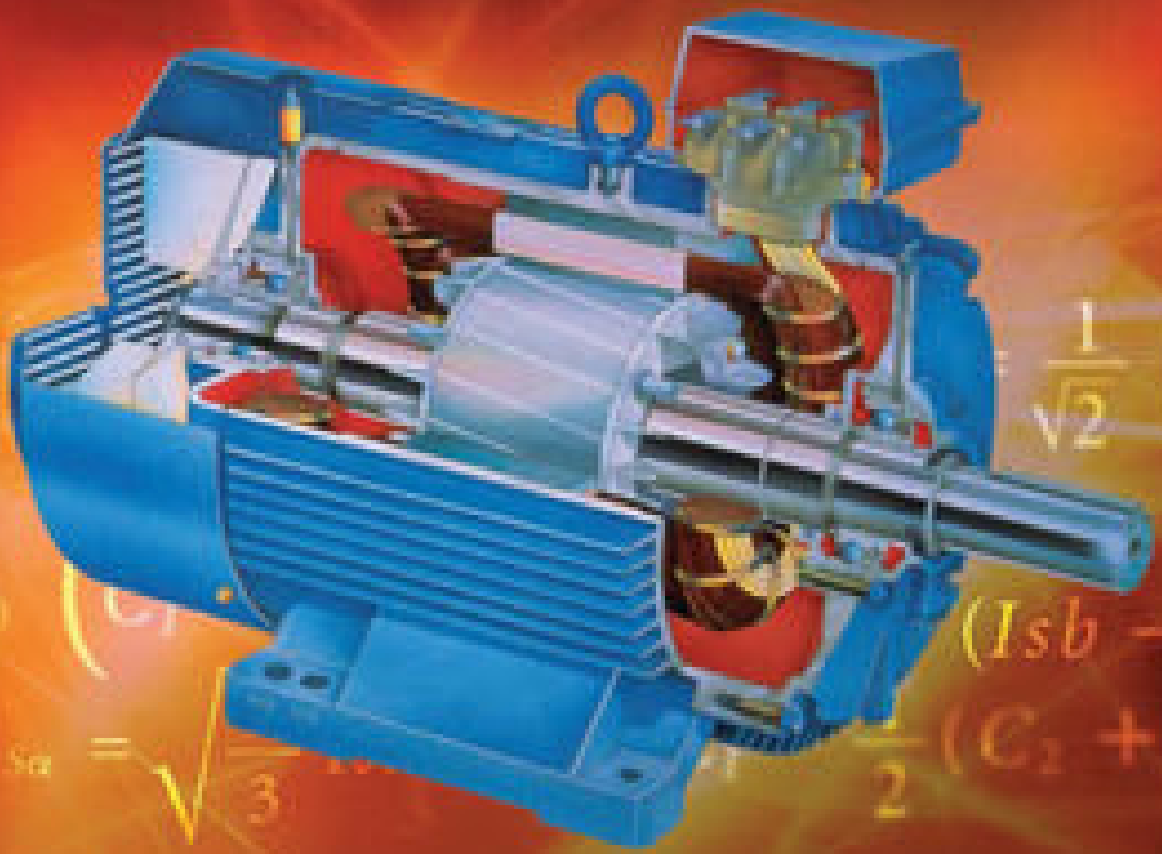
TZE-FUN CHAN | KELI SHI

APPLIED INTELLIGENT CONTROL OF INDUCTION MOTOR DRIVES



Companion Website

APPLIED INTELLIGENT CONTROL OF INDUCTION MOTOR DRIVES

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

APPLIED INTELLIGENT CONTROL OF INDUCTION MOTOR DRIVES

Tze-Fun Chan
The Hong Kong Polytechnic University, Hong Kong, China

Keli Shi
Netpower Technologies, Inc., Texas, USA

IEEE
IEEE PRESS

John Wiley & Sons (Asia) Pte Ltd

This edition first published 2011

Registered office
John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop, #02-01, Singapore 129809

Library of Congress Cataloging-in-Publication Data

Chan, Tze Fun.
Applied intelligent control of induction motor drives / Tze-Fun Chan, Keli Shi.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-470-82556-3 (cloth)
1. Intelligent control systems. 2. Electric motors, Induction. I. Shi, Keli. II. Title.
TJ217.5.C43 2011
621.46–dc22

2010035690

Print ISBN: 978-0-470-82556-3
ePDF ISBN: 978-0-470-82557-0
oBook ISBN: 978-0-470-82558-7
ePub ISBN: 978-0-470-82828-1

Typeset in 10/12pt Times by Thomson Digital, Noida, India.

Contents

Preface

Induction motors are the most important workhorses in industry and they are manufactured in large numbers. About half of the electrical energy generated in a developed country is ultimately consumed by electric motors, of which over 90 % are induction motors. For a relatively long period, induction motors have mainly been deployed in constant-speed motor drives for general purpose applications. The rapid development of power electronic devices and converter technologies in the past few decades, however, has made possible efficient speed control by varying the supply frequency, giving rise to various forms of adjustable-speed induction motor drives. In about the same period, there were also advances in control methods and artificial intelligence (AI) techniques, including expert system, fuzzy logic, neural networks and genetic algorithm. Researchers soon realized that the performance of induction motor drives can be enhanced by adopting artificial-intelligence-based methods. Since the 1990s, AI-based induction motor drives have received greater attention and numerous technical papers have been published. Speed-sensorless induction drives have also emerged as an important branch of induction motor research. A few good reference books on intelligent control and power electronic drives were written. Some electric drive manufacturers began to incorporate AI-control in their commercial products.

This book aims to explore possible areas of induction motor control that require further investigation and development and focuses on the application of intelligent control principles and algorithms in order to make the controller independent of, or less sensitive to, motor parameter changes. Intelligent control is becoming an important and necessary method to solve difficult problems in control of induction motor drives. Based on classical electrical machine and control theory, the authors have investigated the applications of expert-system control, fuzzy-logic control, neural-network control, and genetic algorithm to various forms of induction motor drive.

This book is the result of over fifteen years of research on intelligent control of induction motors undertaken by the authors at the Department of Electrical Engineering, the Hong Kong Polytechnic University and the United States. The methods are original and most of the work has been published in IEEE Transactions and international conferences. In the past few years, our publications have been increasingly cited by Science Citation Index journal papers, showing that our work is being rigorously followed up by the induction motor drives research community.

We believe that the publication of a book or monograph summarizing our latest research findings on intelligent control will benefit the research community. This book will complement

some of the fine references written by eminent electric drives and power electronic experts (such as Peter Vas, Bimal Bose, and Dote and Hoft, to name just a few), and at the same time the presentation will enable researchers to explore new research directions. Numerous examples, block diagrams, and simulation programs are provided for interested readers to conduct related investigations.

This book adopts a practical simulation approach that enables interested readers to embark on research in intelligent control of electric drives with the minimum effort and time. Intelligent control techniques have to be used in practical applications where controller designs involve noise distribution (Kalman filter), pseudo-random data (random PWM), inference similar to human, system identification, and lookup table identification. Artificial intelligence techniques are presented in the context of the drive applications being considered and a strong link between AI and the induction motor drive is established throughout the chapters. The numerous simulation examples and results presented will shed new light on possible future induction motor drives research.

There are twelve chapters in this book. Chapter 1 gives an overview of induction motor drives and reviews previous work in this important technical area. Chapter 2 presents the philosophy of induction motor control. From the classical induction motor model, the differential equations are formulated that fit in a generic control framework. Various control schemes are then discussed, followed by the development of general control algorithms. Modeling and simulation of induction motors are discussed in Chapter 3 with the aid of detailed MATLAB®/Simulink block diagrams.

Chapter 4 is a primer for simulation of intelligent control systems using MATLAB®/Simulink. Programming examples of fuzzy-logic, neural network, Kalman filter, and genetic algorithm are provided to familiarize readers with simulation programming involving intelligent techniques. The exercises will fast guide them into the intelligent control area. These models and simulation techniques form the basis of the intelligent control applications discussed in Chapters 5–10 which cover, in this order, expert-system-based acceleration control, hybrid fuzzy/PI two-stage control, neural-network-based direct self control, parameter estimation using neural networks, GA-optimized extended Kalman filter for speed estimation, and optimized random PWM strategy based on genetic algorithms.

In Chapter 5, an expert-system-based acceleration controller is developed to overcome the three drawbacks (sensitivity to parameter variations, error accumulation, and the needs for continuous control with initial state) of the vector controller. In every time interval of the control process, the acceleration increments produced by two different voltage vectors are compared, yielding one optimum stator voltage vector which is selected and retained. The on-line inference control is built using an expert system with heuristic knowledge about the relationship between the motor voltage and acceleration. Because integral calculation and motor parameters are not involved, the new controller has no accumulation error of integral as in the conventional vector control schemes and the same controller can be used for different induction motors without modification. Simulation results obtained on the expert-system-based controller show that the performance is comparable with that of a conventional direct self controller, hence proving the feasibility of expert-system-based control.

In Chapter 6, a hybrid fuzzy/PI two-stage control method is developed to optimize the dynamic performance of a current and slip frequency controller. Based on two features (current magnitude feature and slip frequency feature) of the field orientation principle, the authors apply different strategies to control the rotor speed during the acceleration stage and the steady-

state stage. The performance of the two-stage controller approximates that of a field-oriented controller. Besides, the new controller has the advantages of simplicity and insensitivity to motor parameter changes. Very encouraging results are obtained from a computer simulation using MATLAB®/Simulink software and a DSP-based experiment.

In Chapter 7, implementation of direct self control for an induction motor drive using artificial neural network (ANN) is discussed. ANN has the advantages of parallel computation and simple hardware, hence it is superior to a DSP-based controller in execution time and structure. In order to improve the performance of a direct self controller, an ANN-based DSC with seven layers of neurons is proposed at algorithm level. The execution time is decreased from 250 μs (for a DSP-based controller) to 21 μs (for the ANN-based controller), hence the torque and flux errors caused by long execution times are almost eliminated. A detailed simulation study is performed using MATLAB®/Simulink and Neural-network Toolbox.

Machine parameter estimation is important for field-oriented control (FOC) and sensorless control. Most parameter estimation methods are based on differential equations of the induction motor. Differential operators, however, will cause noise and greatly reduce the estimation precision. Nondifferentiable points will also exist in the motor currents due to rapid turn-on or turn-off of the ideal power electronic switches. Chapter 8 addresses the issue of parameter uncertainties of induction motors and presents a neural-network-based parameter estimation method using an integral model. By using the proposed ANN-based integral models, almost all the machine parameters can be derived directly from the measured data, namely the stator currents, stator voltages and rotor speed. With the estimated parameters, load, stator flux, and rotor speed may be estimated.

Addressing the current research trend, a speed-sensorless controller using an extended Kalman filter (EKF) is investigated in Chapter 9. To improve the performance of the speed-sensorless controller, noise covariance and weight matrices of the EKF are optimized by using a real-coded genetic algorithm (GA). MATLAB®/Simulink based simulation and DSP-based experimental results are presented to confirm the efficacy of the GA-optimized EKF for speed estimation in an induction motor drive.

Chapter 10 is devoted to optimized random pulse-width modulation (PWM) strategies. The optimized PWM inverter can spread harmonic energy and reduce total harmonic distortion, weighted total harmonic distortion, or distortion factor. Without incurring extra hardware cost and programming complexity, the optimized PWM is implemented by writing an optimized carrier sequence into the PWM controller in place of the conventional carrier generator. Comparison between simulation and experimental results verifies that output voltage of the optimized PWM technique is superior to that based on the standard triangular PWM and random PWM methods. A real-valued genetic algorithm is employed for implementing the optimization strategy.

Chapter 11 describes the details of the experimental system and presents the experiments and experimental results. At the hardware level, an experimental system for the intelligent control of induction motor drive is proposed. The system is configured by a DSP (ADMC331), a power module (IRPT1058A), a three-phase Hall-effect current sensor, an encoder (Model GBZ02), a data acquisition card (PCL818HG), a PC host and a data-acquisition PC, as well as a 147 W three-phase induction motor. With the experimental hardware, the MATLAB®/Simulink models, hybrid fuzzy/PI two-stage control algorithm, and GA-EKF method proposed in this book have been verified. It is proposed to use DSP TMS320F28335 for intelligent control with a real time data exchange (RTDX) technique. Many intelligent algorithms are complex and with

larger data block (such as GA and Neural Network) which cannot be written into a DSP chip. With the RTDX technique, hardware-in-the-loop training and simulation may be implemented in the laboratory environment. The RTDX examples of DSP target C programming and PC host MATLAB® programming are provided.

Chapter 12 gives some conclusions and explores possible new developments of AI applications to induction motor drives.

This book will be useful to academics and students (senior undergraduate, postgraduate, and PhD) who specialize in electric motor drives in general and induction motor drives in particular. The readers are assumed to have a good foundation on electrical machines (including reference frame theory and transformation techniques), control theory, and basics of artificial intelligence (such as expert systems, fuzzy logic theory, neural networks, and genetic algorithms). The book is at an advanced level, but senior undergraduate students specializing on electric motor drives projects should also find it a good reference. It also provides a practical guide to research students to get started with hardware implementation of intelligent control of induction motor drives.

Tze-Fun Chan and Keli Shi
March 2010

Acknowledgments

The authors wish to thank John Wiley & Sons (Asia) Pte Ltd in supporting this project.

The authors also wish to thank the Department of Electrical Engineering, the Hong Kong Polytechnic University, Hong Kong, China for the research facilities and support provided. In particular, they would like to offer their appreciation towards Dr Y.K. Wong and Prof. S.L. Ho of the same department for their stimulating ideas on intelligent control and induction motor drives.

In the course of research on intelligent control of induction motor drives, the authors published a number of papers in different journals. These works report the authors' original research results at various stages of development. The authors would like to express their gratitude to these journals for permitting the authors to reuse some of these published materials. In the writing of the book, the original materials are expanded and new results are included.

Thanks are due to IEEE for permission to reproduce materials from the following published papers in IEEE Transactions and IEEE sponsored conferences:

Transactions papers

- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'Speed estimation of an induction motor drive using an optimized extended Kalman filter,' *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. (Reproduced Figures 9.1–9.2, 9.4, 9.10 and Tables 9.1–9.2; Figures 10.16, 10.30, 10.31(c), 10.32(c) and 10.33(c); Figures 11.1, 11.24, 11.26, 11.28–11.29 and Table 11.12.)
- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'A rule-based acceleration control scheme for an induction motor,' *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. (Reproduced Figures 5.1–5.2, 5.4–5.5, 5.8–5.12 and Tables 5.1–5.2.)
- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'Direct self control of induction motor based on neural network,' *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. (Reproduced Figures 7.1–7.22 and Table 7.1.)
- K.L. Shi and Hui Li, 'Optimized PWM strategy based on genetic algorithms,' *IEEE Transaction on Industrial Electronics*, **52**(5), 2005: 1458–1461.

IEEE conference papers

- K.L. Shi, T.F. Chan and Y.K. Wong, 'A novel two-stage speed controller for an induction motor,' *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper MD2-4, May 18–21, 1997, Milwaukee, Wisconsin, USA.

- K.L. Shi, T.F. Chan and Y.K. Wong, 'Modeling of the three-phase induction motor using SIMULINK,' *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, May 18–21, 1997, Milwaukee, Wisconsin USA. (Reproduced Figures 3.2–3.4 and 3.7–3.12.)
- K.L. Shi, T.F. Chan and Y.K. Wong, 'Hybrid fuzzy two-stage controller for an induction motor,' *1998 IEEE International Conference on Systems, Man, and Cybernetics*, pp.1898–1903, October 11–14, 1998, San Diego, USA. (Reproduced Figures 6.1–6.5, 6.7–6.11 and Tables 6.1–6.4.)
- K.L. Shi, T.F. Chan and Y.K. Wong, 'Direct self control of induction motor using artificial neural network,' *1998 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1696–1701, October 11–14, 1998, San Diego, USA October.
- K.L. Shi, T.F. Chan and Y.K. Wong and S.L. Ho, 'An improved two-stage control scheme for an induction motor.' *Proceedings of the IEEE 1999 International Conference on Power Electronics and Drive Systems*, pp. 405–410, July 27–29, 1999, Hong Kong.
- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'A rule-based acceleration control scheme for an induction motor,' *Proceedings of IEEE International Electric Machines and Drives Conference (IEMDC '99)*, Seattle, Washington, USA, pp. 613–615.
- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'Speed estimation of induction motor using extended Kalman filter,' *IEEE 2000 Winter Meeting*, vol. 1, pp. 243–248, January 23–27, 2000, Singapore.
- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'Direct self control of induction motor based on neural network,' *IEEE Industry Applications Society (IEEE-IAS) 2000 Meeting*, October 8–12, 2000, Vol. 3, pp. 1380–1387, Rome, Italy.
- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'A novel hybrid fuzzy/PI two-stage controller for an induction motor drive,' *IEEE International Electric Machines and Drives Conference (IEMDC '2001)*, pp.415–421, June 17–20, 2001, Cambridge, MA, USA. (Reproduced Figures 6.31 and 11.16.)
- K.L. Shi and Hui Li, 'An optimized PWM method using genetic algorithms,' in *Proc. IEEE IECON 2003*, Nov 2–6, 2003, Roanoke, VA, pp. 7–11.

Thanks are due to Taylor & Francis Ltd for permission to reuse the contents of the following article in Chapter 5:

- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'A new acceleration control scheme for an inverter-fed induction motor,' *Electric Power Components and Systems*, **27**(5), 1999: 527–554.

Thanks are due to ACTA Press for permission to reuse the contents of the following article in Chapters 3 and 6:

- K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, 'Modeling and simulation of a novel two-stage controller for an induction motor,' *International Association of Science and Technology for Development (IASTED) Journal on Power and Energy Systems*, **19**(3), 1999: 257–264.

Thanks are also due to International Journal on Electrical Engineering Education for permission to reuse the contents of the following article in Chapter 3:

- K.L. Shi, T.F. Chan and Y.K. Wong; 'Modeling and simulation of the three-phase induction motor,' *International Journal on Electrical Engineering Education*, **36**(2), 1999: 163–172.

Last but not least, the authors thank the production staff of John Wiley & Sons (Asia) Pte Ltd for their strong support and smooth cooperation.

About the Authors

Tze-Fun Chan received his B.Sc. (Eng.) and M.Phil. degrees in electrical engineering from the University of Hong Kong, Hong Kong, China, in 1974 and 1980, respectively. He received his PhD degree in electrical engineering from City University London, UK, in 2005. Since 1978, he has been with the Department of Electrical Engineering, the Hong Kong Polytechnic University, Hong Kong, China, where he is now Associate Professor and Associate Head of Department. Dr Chan's research interests are self-excited induction generators, brushless a.c. generators, permanent-magnet machines, finite element analysis of electric machines, and electric motor drives control. In 2006, he was awarded a Prize Paper by IEEE Power Engineering Society Power Generation and Energy Development Committee. In 2007, he co-authored (with Prof. Loi Lei Lai) a book entitled *Distributed Generation – Induction and Permanent Magnet Generators* published by Wiley (ISBN: 978-0470-06208-1). In 2009, he was awarded another Prize Paper by IEEE Power Engineering Society Power Generation and Power Committee. Dr Chan is a Chartered Engineer, a member of Institution of Engineering and Technology, UK, a member of Hong Kong Institution of Engineers, Hong Kong, and a member of the Institute of Electrical and Electronic Engineers, USA.

Keli Shi received his BS degree in electronics and electrical engineering from Chengdu University of Science and Technology and MS degree in electrical engineering from Harbin Institute of Technology in 1983 and 1989, respectively. He received his PhD in electrical engineering from the Hong Kong Polytechnic University in 2001. From 2001 to 2002, he was a Postdoctoral Scholar in the Electrical and Computer Engineering Department of Ryerson University, Canada. From 2003 to 2004, he was a Postdoctoral Scholar of Florida State University, Florida, USA. Currently, Dr Shi is a Director of Test Engineering in Netpower Technologies Inc., Texas, USA, where he has been since 2004. His research interests are DSP applications and intelligent control of induction and permanent-magnet motors.

List of Symbols

A, B, C	input and output matrices of a continuous system
A_n, B_n, C_n	input and output matrices of a discrete system
b	bias vector of neural network
c_f	friction coefficient
$G(t)$	weighting matrix of noise
H	matrix of output prediction in Kalman filter algorithm
i_r^e	vector of rotor current in the excitation reference frame, A
i_{dr}^e, i_{qr}^e	components of the vector of rotor current in the excitation reference frame, A
i_s^e	vector of stator current in the excitation reference frame, A
i_{ds}^e, i_{qs}^e	components the stator current vector in the excitation reference frame, A
i_s^s	stator current vector in the stator reference frame, A
i_{dr}^s, i_{qr}^s	components of the rotor current vector in the stator reference frame, A
i_{ds}^s, i_{qs}^s	components of the stator current vector in the stator reference frame, A
J_M	moment of inertia of the rotor, kg m^2
J_L	moment of inertia of the load, kg m^2
k	coefficient to calculate slip increment
k_T	torque constant, N.m/Wb/A
K_n	Kalman filter gain
L_s	stator inductance in the 'T' equivalent circuit, H/ph
L_S	stator inductance in the 'Γ' equivalent circuit, H/ph
L_M	mutual inductance in the 'T' equivalent circuit, H/ph
L_μ	stator inductance in the 'Γ' equivalent circuit, H/ph
L_r	rotor inductance in the 'T' equivalent circuit, H/ph
L_{lr}	rotor leakage inductance in the 'T' equivalent circuit, H/ph
L_R	rotor inductance in the 'Γ' equivalent circuit, H/ph
M	sampling period, s
p	differentiation operator (d/dt), s^{-1}
P	number of poles
P_n	error covariance matrix of Kalman filter algorithm
$q(x)$	feedback signal
Q	covariance matrix of system noise
R	covariance matrix of measurement noise
R_r	rotor resistance in the 'T' equivalent circuit, Ω/ph

R_R	rotor resistance in the 'Γ' equivalent circuit, Ω/ph
R_s	stator resistance in the 'T' equivalent circuit, Ω/ph
R_S	stator resistance in the 'Γ' equivalent circuit, Ω/ph
T	developed torque, N.m
T_{steady}	steady-state torque, N.m
T_L	load torque, N.m
u	control function, vector
V_r^e	vector of the rotor voltage in the excitation reference frame, V
V_{dr}^e, V_{qr}^e	components of the vector of rotor voltage in the excitation reference frame, V
V_s^e	vector of the stator voltage in the excitation reference frame, V
V_{ds}^e, V_{qs}^e	components of the vector of stator voltage in the excitation reference frame, V
V_s^s	vector of stator voltage in the stator reference frame, V
V_{ds}^s, V_{qs}^s	components of the vector of stator voltage in the stator reference frame, V
$v(t)$	noise matrix of output model (measurement noise)
$w(t)$	noise matrix of state model (system noise)
w	weight vector of neural network
x	system state
y	system output
ω	supply angular frequency, synchronous speed of a 2-pole motor, rad/s
ω_r	slip speed of a 2-pole motor, rad/s
ω_o	rotor speed, rad/s
$\Delta\omega_o$	speed error, rad/s
ω_o*	speed command, rad/s
ω_r*	instantaneous slip speed command, rad/s
λ_μ	vector of stator flux of the 'Γ' equivalent circuit in the stator reference-frame, Wb
$\lambda_\mu{}^*$	command of stator flux vector of the 'Γ' equivalent circuit in the stator reference-frame, Wb
$\lambda_{d\mu}, \lambda_{q\mu}$	components of stator flux vector of the 'Γ' equivalent circuit in the stator reference-frame, Wb
λ_r^e	vector of the rotor flux of the 'T' equivalent circuit in the excitation reference frame, Wb
λ_r^e*	command of the rotor flux vector of the 'T' equivalent circuit in the excitation reference frame, Wb
$\lambda_{dr}^e, \lambda_{qr}^e$	components of the rotor flux vector of the 'T' equivalent circuit in the excitation reference frame, Wb
λ_r^s	vector of the rotor flux of the 'T' equivalent circuit in the stator reference frame, Wb
$\lambda_{dr}^s, \lambda_{qr}^s$	components of the rotor flux vector of the 'T' equivalent circuit in the stator reference frame, Wb
λ_M^s	vector of the airgap flux of the 'Γ' equivalent circuit in the stator reference frame, Wb
θ_r	angular position (phase) of the rotor flux vector in the stator reference frame, rad

θ_μ	angular position (phase) of the stator flux vector in the stator reference frame, rad
$\theta(i_s)$	angular position (phase) of the stator current vector in the stator reference frame, rad
γ	normalized mechanical time constant kg.m
Φ	matrix of state prediction in Kalman filter algorithm

1

Introduction

1.1 Induction Motor

Conversion from electrical energy to mechanical energy is an important process in modern industrial civilization. About half of the electricity generated in a developed country is eventually converted to mechanical energy, usually by means of electrical machines (Leonhard, 1996; Sen, 1997).

Typical applications of electrical machine drives are:

1. Appliances (washing machines, blowers, compressors, pumps);
2. Heating/ventilation/air conditioning (HVAC);
3. Industrial servo drives (motion control, robotics);
4. Automotive control (electric vehicles).

Since its invention in 1888, the induction motor has become the most widely used motor in industry. Compared with d.c. motors, the cage induction motor has distinct advantages (Novotny and Lipo, 1996) as listed below:

1. No commutator and brushes,
2. Ruggedness,
3. Lower rotor inertia,
4. Maintenance free, simpler protection,
5. Smaller size and weight,
6. Lower price.

Consequently, most industrial drive applications employ induction motors. Unfortunately, the speed of an induction motor cannot be continuously varied without additional expensive equipment. High-performance control of an induction motor is more difficult than d.c. motors, because the induction motor is inherently a dynamic, recurrent, and nonlinear system.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

1.2 Induction Motor Control

Induction motor control problems have attracted the attention of researchers for many years. Most of the earlier researches are based on classical control theory and electric machine theory, using precise mathematical models of the induction motor. As shown in Figure 1.1, an induction motor control system consists of the controller, sensors, inverter, and the induction motor. It can be seen that a study of induction motor control involves three main electrical engineering areas: control, power electronics, and electrical machines (Bose, 1981).

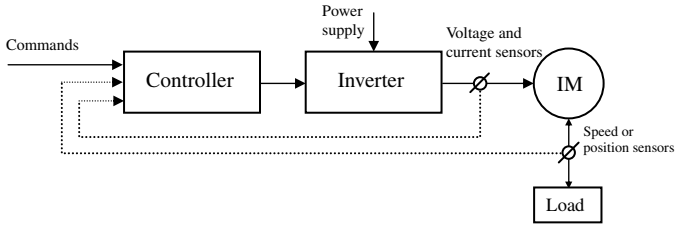


Figure 1.1 An induction motor control system.

The induction motor can be described by a fifth order nonlinear differential equation with two inputs and only three state variables are available for measurement (Marino and Tomei, 1995). The control task is further complicated by the fact that the induction motor is subject to unpredictable disturbances (such as noise and load changes) and there are uncertainties in machine parameters. Induction motor control has constituted a theoretically interesting and practically important class of nonlinear systems, and is evolving into a benchmark example for nonlinear control (Ortega and Asher, 1998).

Intelligent control, which includes expert-system control, fuzzy-logic control, neural-network control, and genetic algorithm, is not only based on artificial intelligence (AI) theory, but also based on conventional control theory. Consequently, new control methods can be developed by the application of artificial intelligence (Bose, 1993).

1.3 Review of Previous Work

Scientists and experts have devoted a lot of efforts to induction motor control in the past decades. Developing new control principle, algorithm, and hardware for induction motor control has become a challenge that industry must face today. The development of induction motor control may be summarized as follows.

In 1946, Weygandt and Charp investigated the transient performance of induction motor by using an analog computer (Weygandt and Charp, 1946).

In 1956, Bell Laboratories invented the thyristor (or silicon-controlled rectifier) (Bose, 1989).

In 1959, Kovacs and Racz applied rotating reference frames and space vectors to the study of induction motor transients (Kovacs and Racz, 1959).

Since 1960, various scalar control strategies of constant voltage/frequency (V/Hz) control of induction motor had been proposed (Bose, 1981).

In 1961, McMurray and Shattuck proposed the inverter circuit with pulse width modulation (PWM) (McMurray and Slattuck, 1961).

In 1968 and in 1970, field orientation principle was first formulated by Hasse and Blaschke (Hasse, 1969; Blashke, 1972).

In 1985, direct self control was proposed by M. Depenbrock, I. Takahashi, and T. Noguchi (Depenbrock, 1985; Takahashi and Noguchi, 1986).

In the 1990s, intelligent control of induction motor received wide attention (Bose, 1992).

Recently, revolutionary advances in computer technology, power electronics, modern control, and artificial intelligence have led to a new generation of induction motor control that may provide significant economic benefits.

The voltage or current supplied to an induction motor can be expressed as a sinusoidal function of magnitude and frequency or magnitude and phase. Accordingly, induction motor control methods are classified into two categories: scalar control in which the voltage magnitude and frequency are adjusted, and vector control in which the voltage magnitude and phase are adjusted.

1.3.1 Scalar Control

The scalar controllers are usually used in low-cost and low-performance drives. They control the magnitude/frequency of voltage or current. Typical studies of scalar control include open-loop voltage/frequency (V/Hz) control, closed-loop V/Hz control, and stator current and slip-frequency control (Bose, 1981).

When the load torque is constant and there are no stringent requirements on speed regulation, it suffices to use a variable-frequency induction motor drive with open-loop V/Hz control. Applications which require only a gradual change in speed are being replaced by open-loop controllers, often referred to as general purpose AC drives (Rajashekara, Kawamura, and Matsuse, 1996).

When the drive requirements include faster dynamic response and more accurate speed or torque control, it is necessary to operate the motor in the closed-loop mode. Closed-loop scalar control includes closed-loop V/Hz control and stator current and slip frequency control.

1.3.2 Vector Control (Rajashekara, Kawamura, and Matsuse, 1996)

The vector controllers are expensive and high-performance drives, which aim to control the magnitude and phase of voltage or current vectors. Vector control methods include field-oriented control (FOC) and direct self control (DSC). Both methods attempt to reduce the complex nonlinear control structure into a linear one, a process that involves the evaluation of definite integrals. FOC uses the definite integral to obtain the rotor flux angle, whereas DSC uses the definite integral to obtain the stator flux space vector. Although the implementation of both methods has largely been successful, they suffer from the following drawbacks:

1. Sensitivity to parameter variations;
2. Error accumulation when evaluating the definite integrals; if the control time is long, degradation in the steady-state and transient responses will result due to drift in parameter values and excessive error accumulation;
3. In both methods, the control must be continuous and the calculation must begin from an initial state.

1.3.3 Speed Sensorless Control

Speed sensorless control of induction motors is a new and promising research trend. To eliminate the speed and position sensors, many speed and position estimation algorithms have been proposed recently. These algorithms are generally based on complex calculations which involve the machine parameters and the measurement of terminal voltages and currents of the induction motor. Speed sensorless control can be regarded as open-loop control because the measurement is included in the controller (Rajashekara, Kawamura, and Matsuse, 1996).

1.3.4 Intelligent Control of Induction Motor

Despite the great efforts devoted to induction motor control, many of the theoretical results cannot be directly applied to practical systems. The difficulties that arise in induction motor control are complex computations, model nonlinearity, and uncertainties in machine parameters. Recently, intelligent techniques are introduced in order to overcome these difficulties. Intelligent control methodology uses human motivated techniques and procedures (for example, forms of knowledge representation or decision making) for system control (Bose, 1997; Narendra and Mukhopadhyay, 1996).

1.3.5 Application Status and Research Trends of Induction Motor Control

Among the above control techniques, market evidence shows that up to the present only two have found general acceptance. They are the open-loop constant V/Hz control for low-performance applications and the indirect vector control for high-performance applications (Bose, 1993). Vector control principle, intelligent-based algorithm, and DSP-based hardware represent recent research trends of induction motor control.

1.4 Present Study

The present research status of induction motor control suggests the areas that require further investigation and development. The objective of this book is to investigate intelligent control principles and algorithms in order to make the performance of the controller independent of, or less sensitive to, motor parameter changes. Based on theories of the induction motor and control principles, expert-system control, fuzzy-logic control, neural-network control, and genetic algorithm for induction motor drive will be investigated and developed. The scope of the present book is summarized as follows:

1. **Computer modeling of induction motor**
   The induction motor model typically consists of an electrical model and a mechanical model, which is a fifth-order nonlinear system. Using MATLAB®/Simulink software, three induction motor models (current-input model, voltage-input model, discrete-state model) are constructed for the simulation studies of the induction motor drive. The three models can be used to simulate the actual induction motor effectively. In addition, a PWM model, an encoder model, and a decoder model are also proposed.

2. **Expert-system based acceleration control**
   An expert-system based acceleration controller is developed to overcome the drawbacks (sensitivity to parameter variations, error accumulation, and the needs for continuous control with initial state) of the vector controller. In every time interval of the control process, the acceleration increments produced by two different voltage vectors are compared, yielding one optimum stator voltage vector which is selected and retained. The on-line inference control is built using an expert system with heuristic knowledge about the relationship between the motor voltage and acceleration. Because integral calculation and motor parameters are not involved, the new controller has no accumulation error of integral as in the conventional vector control schemes and the same controller can be used for different induction motors without modification. Simulation results obtained on the expert-system based controller show that the performance is comparable with that of a conventional direct self controller, hence proving the feasibility of expert-system based control.
3. **Hybrid fuzzy/PI two-stage control**
   A hybrid fuzzy/PI two-stage control method is developed to optimize the dynamic performance of a current and slip frequency controller. Based on two features (current magnitude feature and slip frequency feature) of the field orientation principle, different strategies are proposed to control the rotor speed during the acceleration stage and the steady-state stage. The performance of the two-stage controller approximates that of a field-oriented controller. Besides, the new controller has the advantages of simplicity and insensitivity to motor parameter changes. Very encouraging results are obtained from a computer simulation using MATLAB®/Simulink software and experimental verification using a DSP-based drive.
4. **Neural-network-based direct self control (DSC)**
   Artificial neural network (ANN) has the advantages of parallel computation and simple hardware, hence it is superior to a DSP-based controller in execution time and structure. In order to improve the performance of a direct self controller, an ANN-based DSC with seven layers of neurons is proposed at algorithm level. The execution time is decreased from 250 μs (for a DSP-based controller) to 21 μs (for the ANN-based controller), hence the torque and flux errors caused by long execution times are almost eliminated. A detailed simulation study is performed using MATLAB®/Simulink and Neural-network Toolbox.
5. **Genetic algorithm based extended Kalman filter for rotor speed estimation of induction motor**
   Addressing the current research trend, speed-sensorless controller with the extended Kalman filter is investigated. To improve the performance of the speed-sensorless controller, noise covariance and weight matrices of the extended Kalman filter are optimized by using a real-coded genetic algorithm (GA). MATLAB®/Simulink-based simulation and DSP-based experimental results are presented to confirm the efficacy of the GA-optimized EKF for speed estimation in induction motor drives.
6. **Parameter estimation using neural networks**
   Integral models of an induction motor are described and implemented by using an artificial neural network (ANN) approach. By using the proposed ANN-based integral models, almost all the machine parameters can be derived directly from the measured data, namely the stator currents, stator voltages and rotor speed. With the estimated parameters, load, stator flux, and rotor speed may be estimated for induction motor control.

7. **Optimized random PWM strategies based on genetic algorithm**
   Random carrier-frequency PWM, random pulse-position PWM, random pulse-width PWM, and hybrid random pulse-position and pulse-width PWM are optimized by genetic algorithm (GA). A single-phase inverter is employed for the optimization study, and the resulting waveforms are evaluated based on Fourier analysis. The validity of the GA-optimized random carrier-frequency PWM is verified by experimental studies on a DSP-based voltage controlled inverter. The GA-optimized PWM proposed may be applied to single-phase ac induction motor drives for low performance applications, such as pumps, fans and mixers, as well as uninterruptible power supply (UPS).
8. **Hardware experiments**
   At the hardware level, an experimental system for intelligent control of an induction motor is proposed and implemented. The system is configured by a DSP (ADMC331), a power module (IRPT1058A), a three-phase Hall-effect current sensor, an encoder (Model GBZ02), a data acquisition card (PCL818HG), a PC host and a data-acquisition PC, as well as a 147-W 3-phase induction motor. With the experimental hardware, the MATLAB®/Simulink models, hybrid fuzzy/PI two-stage control algorithm, and GA-EKF method described in this book are verified. Using a TMS320F2812 DSP board and an IRAMX16UP60A inverter module, a GA-optimized single-phase random-carrier-frequency PWM inverter is implemented. Besides, programming examples are presented to demonstrate RTDX (Real Time Data exchange) technique to exchange real-time data between a TMS320F28335 DSP and MATLAB® software. With the RTDX technique, real-time DSP applications can be supported by a complex MATLAB® AI program running simultaneously on a PC.
9. **Programming examples**
   Using MATLAB®/Simulink software and CCStudio_v3.3 software, a large number of programming examples are described in the book and the source codes can be found on the book companion website as supplementary materials. The programming examples may be classified into the following categories.

a. Modeling and simulation of induction motor (Chapter 3)
b. Fundamentals of intelligent control simulation (Chapter 4)
c. Induction motor control
   - Expert-system based acceleration control (Chapter 5)
   - Hybrid fuzzy/PI two-stage control (Chapter 6)
   - Direct self control of induction motor (Chapter 7)
   - Neural-network based direct self control (Chapter 7)
   - Field-oriented control of induction motor (Chapter 8)
   - Voltage-frequency controlled induction motor drive (Chapter 9).

d. Estimations for induction motor drives
   - Parameter estimation using neural networks (Chapter 8)
   - Load estimation based on integral model of induction motor (Chapter 8)
   - Flux estimation based on integral model of induction motor (Chapter 8)
   - Rotor speed estimation based on integral model of induction motor (Chapter 8)
   - GA-optimized extended Kalman filter for speed estimation (Chapter 9).

e. Sensorless control of induction motor
   - Integral-model-based sensorless control of induction motor (Chapter 8)

EKF-based sensorless V/Hz control of induction motor (Chapter 9)
EKF-based sensorless field-oriented control (FOC) of induction motor (Chapter 9).

f. PWM strategies
Space vector PWM Simulink model (in the folder 'Chapter 8.4' of the book companion website)
Optimized random PWM strategy based on genetic algorithms (Chapter 10).

g. DSP TMS320F28335 programming examples
3-phase PWM programming example (Chapter 11)
RTDX programming example (Chapter 11)
ADC programming example (Chapter 11)
CAP programming example (Chapter 11).

References

Blashke, F. (1972) The principle of field-orientation as applied to the new 'Transvektor' closed-loop control system for rotating-field machines. *Simians Review*, **34**(5), 21–220.

Bose, B.K. (1981) *Adjustable Speed AC Drive Systems*, IEEE Press, New York.

Bose, B.K. (1989) Power electronics – an emerging technology. *IEEE Transactions on Industrial Electronics*, **36**, 403–411.

Bose, B.K. (1992) Recent advances in power electronics. *IEEE Transactions on Power Electronics*, **7**(1), 2–16.

Bose, B.K. (1993) Power electronics and motion control-technology status and recent trends. *IEEE Transactions on Industry Applications*, **29**, 902–909.

Bose, B.K. (1997) Expert system, fuzzy logic, and neural networks in power electronics and drives, in *Power Electronics and Variable Frequency Drives: Technology and Applications* (ed. B.K. Bose), IEEE Press, New Jersey.

Depenbrock, M. (Inventor) (18, Oct. 1985) 'Direct Self-control of the Flux and Rotary Moment of a Rotary-field Machine,' United States Patent 4,678,248.

Hasse, K. (1969) 'About the Dynamics of Adjustable-speed Drives with Converter-fed Squirrel-cage Induction Motors' (in German), Dissertation, Darmstadt Technische Hochschule.

Kovacs, K.P. and Racz, J. (1959) *Transiente Vorgane in Wechse Istrommaschinen*, Verlag der Ungarischen Akademie der Wissenschaften, Budapest.

Leonhard, W. (1996) *Control of Electrical Drives*, Springer-Verlag Berlin, Heidelberg.

Marino, R. and Tomei, P. (1995) *Nonlinear Control Design*, Prentice Hall Europe, Hemel Hempstead.

McMurray, W. and Slattuck, D.D. (1961) A silicon-controlled rectifier inverter with improved commutation. *AIEE Transactions on Communications and Electronics*, **80**, 531–542.

Narendra, K.S. and Mukhopadhyay, S. (1996) Intelligent Control Using Neural Networks, in *Intelligent Control Systems: Theory and Applications* (eds M.M. Gupta and N.K. Sinha), IEEE Press, New York.

Novotny, D.W. and Lipo, T.A. (1996) *Vector Control and Dynamics of AC Drives*, Oxford University Press, Oxford.

Ortega, R. and Asher, G. (1998) Joint special issue on nonlinear control of induction motor. *IEEE Transactions on Industrial Electronics*, **45**(2), 367.

Rajashekara, K., Kawamura, A., and Matsuse, K. (1996) Speed sensorless control of induction motor, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura,and K. Matsuse), IEEE Press, New Jersey.

Sen, P.C. (1997) *Principles of Electric Machines and Power Electronics*, John Wiley & Sons, Inc., New York.

Takahashi, I. and Noguchi, T. (1986) A new quick-response and high-efficiency control strategy of an induction motor. *IEEE Transactions on Industry Applications*, **22**(5), 820–827.

Weygandt, C.N. and Charp, S. (1946) Electromechanical transient performance of induction motors. *AIEE Transactions*, **64**(Pt. III), 1000.

2

Philosophy of Induction Motor Control

2.1 Introduction

Strong demands for high-performance motor drives in industry have stimulated the research on induction motor control. Despite the great effort devoted to this research area for many years, the desired performance in induction motor drives has not been achieved satisfactorily. In the 1970s, nonlinear controllability and observability began to be studied using differential geometric tools (Marino and Tomei, 1995), while the field-oriented controller was introduced in 1968 and 1970 on the basis of the mathematical model of the induction motor (Hasse, 1969; Blashke, 1972). Field-oriented control has the innovative feature that, based on the reference frame theory (proposed by R.H. Park in the late 1920s) (Krause, Wasynczuk, and Sudhoff, 1995), it makes use of nonlinear transformation of stator coordinates and of nonlinear state feedback (which aims at nonlinearity cancellation) to make the closed-loop system linear in the new coordinates. A theory of nonlinear feedback control design was developed during the 1980s (Marino and Tomei, 1995) and in 1985 direct self control was proposed (Depenbrock, 1985). In the direct self control scheme, the errors in the torque and flux are directly used to choose the inverter switching state with the hysteresis (or bang-bang) control strategy. Recently, induction motor control has involved more and more subject areas, such as modern nonlinear control, electrical machine, artificial intelligence, power electronics, signal processing, and computer science (Bose, 1993). Many research papers in this area have been published, and it is clear that the development of a new controller cannot be implemented briefly and in a single step. Consequently, the understanding and design of the controller should be made at theory, algorithm, and hardware levels to facilitate analysis and system realization.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

2.2 Induction Motor Control Theory

2.2.1 Nonlinear Feedback Control

Modern control design is fundamentally a time-domain technique. A state-space model of the system to be controlled, or plant, is required (Marino and Tomei, 1995). A commonly used model for a nonlinear system is (Vidyasagar, 1993):

$$\dot{x}(t) = f(t, x(t), u(t)), \qquad x(0) = x_0 \tag{2.1}$$

$$y(t) = h(t, x(t)) \tag{2.2}$$

where t denotes time, x_0 is the initial condition, $x(t)$ denotes the value of the function $x(.)$ at time t and is an n-dimensional vector, $u(t)$ is similarly defined and is an m-dimensional vector, and the function f associates, with each value of t, $x(t)$, and $u(t)$, a corresponding n-dimensional vector. Following common convention, this is denoted as: $t \in R^+$, $x(t) \in R^n$, $u(t) \in R^m$, and f: $R^+ \times R^n \times R^m \rightarrow sR^n$. The quantity $x(t)$, which is a vector of internal variables, is referred to as the *state* of the system at time t, while $u(t)$ is called the *control function*. $y(t) \in R^s$ is a vector of measured *outputs*. When $m = s = 1$, we speak of single-input single-output (SISO) systems; we speak of multiple-input multiple-output (MIMO) systems when either $m > 1$ and $s > 1$.

If we define

$$v(t) = q(x(t)) + s(x(t))u(t) \tag{2.3}$$

then the resulting variables $y(t)$ and $v(t)$ satisfy a linear differential equation of the form:

$$\dot{y}(t) = E_1 y(t) + E_2 v(t) \tag{2.4}$$

where the pair (E_1, E_2) is controllable. If this is the case, then the system represented by Equation (2.1) is said to be feedback linearizable. Note that since $s(x(t)) \neq 0$ in some neighborhood of zero, Equation (2.3) can be rewritten as:

$$u(t) = -\frac{q(x(t))}{s(x(t))} + \frac{1}{s(x(t))} v(t) \tag{2.5}$$

where $-q(x(t))/s(x(t))$ and $1/s(x(t))$ are smooth functions.

Hence, if we think of $v(t)$ as the external reference input applied to the system, then Equation (2.3) or Equation (2.5) represents nonlinear state feedback, and a nonlinear state-dependent pre-filter, applied to the system Equation (2.1). Similarly, Equation (2.2) represents a nonlinear state-variable transformation. Feedback signals are a function of system state variables. Hence, a state feedback control system with the overall effects Equations (2.2) and (2.3) can be represented by Figure 2.1.

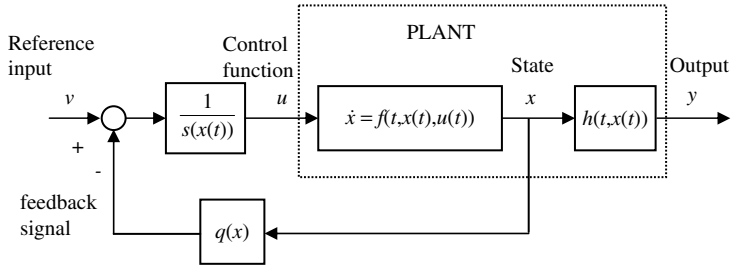


Figure 2.1 A state feedback control system.

In induction motor drives, the relationship between the rotor speed and the power input is not linear. In a vector control scheme, only three state variables (speed and currents in two phases) and two inputs (voltages) are available for measurement, hence the other two state variables (fluxes) have to be estimated. Using the state variables measured or estimated, feedback signals can be calculated by the vector controller. Consequently, the complex nonlinear control structure can be transformed into a linear one.

2.2.2 *Induction Motor Models*

An induction motor model based on a 'Γ' equivalent circuit (Appendix A) in the stator reference frame may be expressed as a fifth-order nonlinear equation (Trzynadlowski, 1994):

$$\begin{cases} \dfrac{d\omega_o}{dt} = \dfrac{2}{3}\dfrac{P}{2J}\left(\lambda^s_{d\mu} i^s_{qs} - \lambda^s_{q\mu} i^s_{ds}\right) - \dfrac{T_L}{J} \\ \dfrac{d\lambda^s_{d\mu}}{dt} = -R_s i^s_{ds} + V^s_{ds} \\ \dfrac{d\lambda^s_{q\mu}}{dt} = -R_s i^s_{qs} + V^s_{qs} \\ \dfrac{di^s_{ds}}{dt} = -k_\Gamma(L_\mu R_R + L_R R_s) i^s_{ds} + \dfrac{P}{2}\omega_o i^s_{qs} + k_\Gamma R_R \lambda^s_{d\mu} + \dfrac{P}{2} k_\Gamma \omega_o \lambda^s_{q\mu} + k_\Gamma L_R V^s_{ds} \\ \dfrac{di^s_{qs}}{dt} = -k_\Gamma(L_\mu R_R + L_R R_s) i^s_{qs} - \dfrac{P}{2}\omega_o i^s_{ds} + k_\Gamma R_R \lambda^s_{q\mu} - \dfrac{P}{2} k_\Gamma \omega_o \lambda^s_{d\mu} + k_\Gamma L_R V^s_{qs} \end{cases} \tag{2.6}$$

where $k_\Gamma = \frac{1}{L^2_\mu - L_\mu L_R}$, and electromagnetic torque $T = \frac{2}{3}\frac{P}{2}\left(\lambda^s_{d\mu} i^s_{qs} - \lambda^s_{q\mu} i^s_{ds}\right)$.

In Equation (2.6), rotor speed ω_o, stator fluxes $(\lambda^s_{d\mu}, \lambda^s_{q\mu})$, and stator currents (i^s_{ds}, i^s_{qs}) are the states, while rotor inertia J, stator and rotor inductances (L_μ, L_R), stator and rotor resistances (R_s, R_R), and the number of poles P are the parameters. The voltages (V^s_{ds}, V^s_{qs}) are inputs, and T_L is the load torque.

Let $\lambda_{\mu}^{s} = \lambda_{d\mu}^{s} + j\lambda_{q\mu}^{s}$, $\lambda_{R}^{s} = \lambda_{dR}^{s} + j\lambda_{qR}^{s}$, $i_{s}^{s} = i_{ds}^{s} + ji_{qs}^{s}$, and $i_{R}^{s} = i_{dR}^{s} + ji_{qR}^{s}$. The flux equations can be written as:

$$\lambda_{\mu}^{s} = L_{\mu} i_{s}^{s} + L_{\mu} i_{R}^{s} \tag{2.7}$$

$$\lambda_{R}^{s} = \lambda_{\mu}^{s} + L_{R} i_{R}^{s}. \tag{2.8}$$

From Equations (2.7) and (2.8), the following equations can be derived.

$$i_{s}^{s} = \frac{\lambda_{\mu}^{s}}{L_{\mu}} - i_{R}^{s} \tag{2.9}$$

$$i_{s}^{s} = \frac{L_{R} + L_{\mu}}{L_{\mu} L_{R}} \lambda_{\mu}^{s} - \frac{1}{L_{R}} \lambda_{R}^{s} \tag{2.10}$$

$$\lambda_{\mu}^{s} = \frac{L_{\mu}}{L_{R} + L_{\mu}} \lambda_{R}^{s} + \frac{L_{\mu} L_{R}}{L_{R} + L_{\mu}} i_{s}^{s} \tag{2.11}$$

$$\lambda_{\mu}^{s} = \lambda_{R}^{s} - L_{R} i_{R}^{s} \tag{2.12}$$

$$i_{s}^{s} = \frac{\lambda_{R}^{s}}{L_{\mu}} - \left(1 + \frac{L_{R}}{L_{\mu}}\right) i_{R}^{s} \tag{2.13}$$

For convenience, Equation (2.6) is referred to as the 1st fifth-order equation.

Substituting Equation (2.7) into Equation (2.6), the 2nd fifth-order equation is obtained.

Substituting Equation (2.9) into Equation (2.6), the 3rd fifth-order equation is obtained.

Substituting Equation (2.10) into Equation (2.6), the 4th fifth-order equation is obtained.

Substituting Equation (2.11) into Equation (2.6), the 5th fifth-order equation is obtained.

Substituting Equations (2.12) and (2.13) into Equation (2.6), the 6th fifth-order equation is obtained.

The six fifth-order equations are listed as follows:

1st equation:	with state variables $\{\lambda_{d\mu}^{s}, \lambda_{q\mu}^{s}, i_{ds}^{s}, i_{qs}^{s}, \omega_{o}\}$.
2nd equation:	with state variables $\{i_{ds}^{s}, i_{qs}^{s}, i_{dR}^{s}, i_{qR}^{s}, \omega_{o}\}$.
3rd equation:	with state variables $\{\lambda_{d\mu}^{s}, \lambda_{q\mu}^{s}, i_{dR}^{s}, i_{qR}^{s}, \omega_{o}\}$.
4th equation:	with state variables $\{\lambda_{dR}^{s}, \lambda_{qR}^{s}, \lambda_{d\mu}^{s}, \lambda_{q\mu}^{s}, \omega_{o}\}$.
5th equation:	with state variables $\{\lambda_{dR}^{s}, \lambda_{qR}^{s}, i_{ds}^{s}, i_{qs}^{s}, \omega_{o}\}$.
6th equation:	with state variables $\{\lambda_{dR}^{s}, \lambda_{qR}^{s}, i_{dR}^{s}, i_{qR}^{s}, \omega_{o}\}$.

An induction motor 'T' equivalent circuit model (Appendix A) can be expressed as another fifth-order nonlinear equation in the stator reference frame (Trzynadlowski, 1994):

$$\begin{cases} \dfrac{d\omega_o}{dt} = \dfrac{2}{3}\dfrac{PL_M}{2JL_r}\left(\lambda^s_{dr} i^s_{qs} - \lambda^s_{qr} i^s_{ds}\right) - \dfrac{T_L}{J} \\ \dfrac{d\lambda^s_{dr}}{dt} = -\dfrac{R_r}{L_r}\lambda^s_{dr} - \dfrac{P}{2}\omega_o \lambda^s_{qr} + \dfrac{R_r L_M}{L_r} i^s_{ds} \\ \dfrac{d\lambda^s_{qr}}{dt} = -\dfrac{R_r}{L_r}\lambda^s_{qr} + \dfrac{P}{2}\omega_o \lambda^s_{dr} + \dfrac{R_r L_M}{L_r} i^s_{qs} \\ \dfrac{di^s_{ds}}{dt} = \dfrac{L_M R_r}{(L_r L_s - L_M^2)L_r}\lambda^s_{dr} + \dfrac{PL_M}{2(L_r L_s - L_M^2)}\omega_o \lambda^s_{qr} - \dfrac{L_M^2 R_r + L_r^2 R_s}{(L_r L_s - L_M^2)L_r} i^s_{ds} + \dfrac{L_r}{(L_r L_s - L_M^2)} V^s_{ds} \\ \dfrac{di^s_{qs}}{dt} = \dfrac{L_M R_r}{(L_r L_s - L_M^2)L_r}\lambda^s_{qr} - \dfrac{PL_M}{2(L_r L_s - L_M^2)}\omega_o \lambda^s_{dr} - \dfrac{{L_M}^2 R_r + L_r^2 R_s}{(L_r L_s - L_M^2)L_r} i^s_{qs} + \dfrac{L_r}{(L_r L_s - L_M^2)} V^s_{qs} \end{cases} \tag{2.14}$$

In Equation (2.14), rotor speed ω_o, rotor fluxes $(\lambda^s_{dr}, \lambda^s_{qr})$, and stator currents (i^s_{ds}, i^s_{qs}) are the states, while rotor inertia J, stator and rotor inductances (L_s, L_r), mutual inductance L_M, stator and rotor resistances (R_s, R_r), and the number of poles P are the parameters. The voltages (V^s_{ds}, V^s_{qs}) are inputs, and T_L is the load torque.

From the 'T' equivalent circuit, another six fifth-order equations with different state variables can be obtained:

7th equation: with state variables $\{\lambda^s_{dM}, \lambda^s_{qM}, i^s_{ds}, i^s_{qs}, \omega_o\}$.
8th equation: with state variables $\{i^s_{ds}, i^s_{qs}, i^s_{dr}, i^s_{qr}, \omega_o\}$.
9th equation: with state variables $\{\lambda^s_{dM}, \lambda^s_{qM}, i^s_{dr}, i^s_{qr}, \omega_o\}$.
10th equation: with state variables $\{\lambda^s_{dr}, \lambda^s_{qr}, \lambda^s_{dM}, \lambda^s_{qM}, \omega_o\}$.
11th equation: with state variables $\{\lambda^s_{dr}, \lambda^s_{qr}, i^s_{ds}, i^s_{qs}, \omega_o\}$.
12th equation: with state variables $\{\lambda^s_{dr}, \lambda^s_{qr}, i^s_{dr}, i^s_{qr}, \omega_o\}$.

The twelve fifth-order equations illustrate the major difficulty encountered in the control of induction motors. For example, in Equations (2.6) and (2.14), the multiplier operators of states render the induction motor becoming a nonlinear system (Trzynadlowski, 1994), while the differential operations give rise to a dynamic system (Delgado, Kambhampati, and Warwick, 1995). The system output speed ω_o is used to calculate the system variables, which makes the motor a recurrent system (Kung, 1993). In addition, local recurrent calculations are involved in Equation (2.6). For example, the term i^e_{ds} appears on both sides of the current equation. Consequently, the induction motor can be referred to as a dynamic, recurrent, and nonlinear system.

2.2.3 *Field-Oriented Control*

When the stator currents are controlled, the last two equations in Equation (2.14) (the 11th fifth-order equation) are neglected. Using the substitution $\omega_r = \omega - (P/2)\omega_o$, the reduced-order

equation in the excitation reference-frame can be expressed as (Trzynadlowski, 1994):

$$\begin{cases} \dfrac{d\omega_o}{dt} = \dfrac{2}{3}\dfrac{PL_M}{2JL_r}\left(\lambda^e_{dr} i^e_{qs} - \lambda^e_{qr} i^e_{ds}\right) - \dfrac{T_L}{J} \\ \dfrac{d\lambda^e_{dr}}{dt} = -\dfrac{R_r}{L_r}\lambda^e_{dr} + \omega_r \lambda^e_{qr} + \dfrac{R_r}{L_r} L_M i^e_{ds} \\ \dfrac{d\lambda^e_{qr}}{dt} = -\dfrac{R_r}{L_r}\lambda^e_{qr} - \omega_r \lambda^e_{dr} + \dfrac{R_r}{L_r} L_M i^e_{qs} \end{cases} \tag{2.15}$$

Based on Equation (2.15), the classical field-oriented control and the recent nonlinear input-output decoupled control have been proposed (Chan, Leung, and Ng, 1990).

The field-orientation conditions can be expressed as (Trzynadlowski, 1994):

$$\lambda^e_{qr} = 0$$
$$\lambda^e_{dr} = \text{constant}$$

Replacing T and λ^e_{dr} with T^* and $\lambda^{e\,*}_{dr}$ respectively and substituting the field orientation conditions into Equation (2.15) with $T = \frac{PL_M}{3L_r}\lambda^e_{dr} i^e_{qs}$, an indirect FOC scheme for an induction motor can be derived as follows:

$$i^e_{ds} = \frac{d\lambda^{e\,*}_{dr}}{dt}\frac{L_r}{L_M R_r} + \frac{1}{L_M}\lambda^{e\,*}_{dr} \tag{2.16}$$

$$i^e_{qs} = \frac{3L_r}{PL_M \lambda^{e\,*}_{dr}} T^* \tag{2.17}$$

$$\omega_r = \frac{3R_r T^*}{P\lambda^{e\,*2}_{dr}} \tag{2.18}$$

$$i^s_s = \left(i^e_{ds} + j i^e_{qs}\right) e^{j(\omega_r + \omega_o)t}. \tag{2.19}$$

T^* and $\lambda^{e\,*}_{dr}$ are the control commands, while ω_o is the feedback signal. The current vectors i^s_{ds} and i^s_{qs} and slip ω_r are obtained from Equations (2.16)–(2.18), and then substituted into Equation (2.19) to give the vector i^s_s for induction motor control.

2.2.4 *Direct Self Control*

Equation (2.6) (the 1st fifth-order equation) is used to realize direct self control. Using the first three equations in Equation (2.6), we can obtain the stator flux λ^s_μ and torque T from the measured stator currents and voltages. Figure 2.2 shows that the direct self control system has the same architecture as the state feedback control system in modern control theory as shown in Figure 2.1.

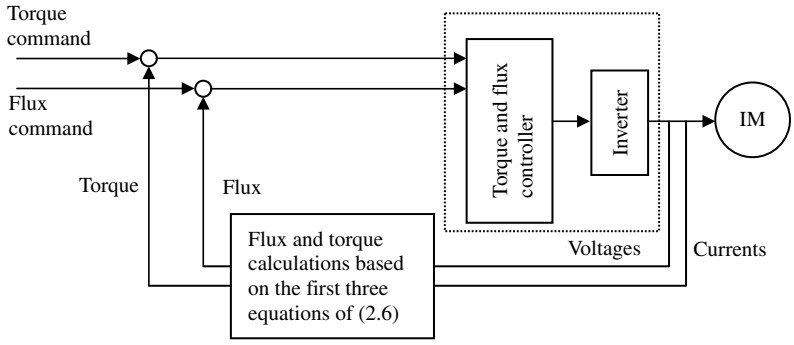


Figure 2.2 Direct self control system.

Using one of the last two equations in Equation (2.6), we can estimate the rotor speed by measurements of voltage and current together with the flux. A control scheme of DSC with speed estimation is shown in Figure 2.3.

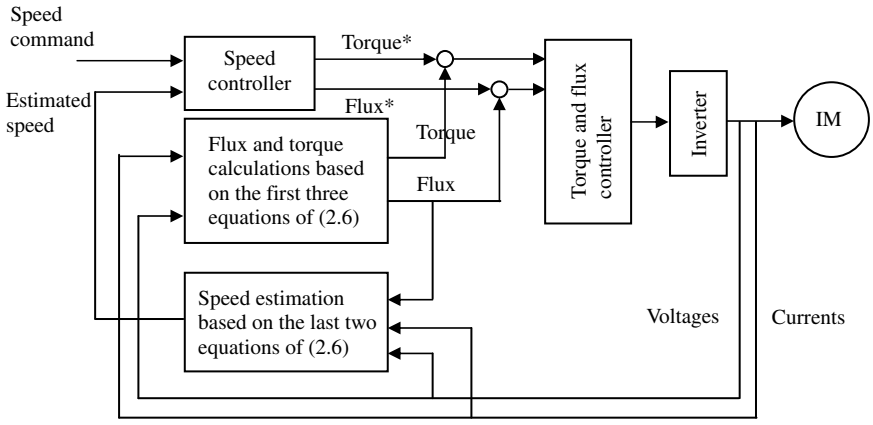


Figure 2.3 Direct self control system with speed estimation.

2.2.5 *Acceleration Control Proposed*

From the first three equations of Equation (2.6), torque T may be expressed as a function of stator current i_s^s and stator flux λ_μ^s while the λ_μ^s is a function of i_s^s and stator voltage V_s^s. The two functions may be expressed as:

$$T = f_1\left(i_s^s, \lambda_\mu^s\right) \tag{2.20}$$

$$\lambda_\mu^s = f_2\left(i_s^s, V_s^s\right). \tag{2.21}$$

Equations (2.20) and (2.21) can be rewritten as:

$$\lambda_{\mu}^{s} = f_3\left(i_s^s, T\right) \tag{2.22}$$

$$V_s^s = f_4\left(\lambda_{\mu}^{s}, i_s^s\right). \tag{2.23}$$

Substituting Equation (2.22) and $T = J\dfrac{d\omega_o}{dt} + T_L$ into Equation (2.23),

$$V_s^s = f_5\left(\frac{d\omega_o}{dt}, i_s^s\right). \tag{2.24}$$

In acceleration control, Equation (2.24) is used as a control function while the acceleration $d\omega_o/dt$ and i_s^s are obtained or derived from the measured signals. Although the concept of acceleration control can be explained using the above theory, the details of the control scheme need to be developed at the algorithm level (Shi, Chan, and Wong, 1997).

2.2.6 *Need for Intelligent Control*

The difficulties that arise in induction motor control can be classified under three categories: (a) complex computation, (b) nonlinearity, (c) uncertainty (Narendra and Mukhopadhyay, 1996).

a. **Complex computation –** In induction motor control, application of conventional control theory and control algorithm often results in complex computations. The vector controls (especially direct self control) are seldom used in practical drive systems due to these complex computations and the associated control time delays. In state estimation control schemes (for example, speed sensorless control) and parameter identification, the computations will be even more complex.
b. **Nonlinearity –** The presence of nonlinearites in an induction motor drive makes the control problem complicated. Current research efforts in nonlinear control theory focus on differential geometric methods and attempt to extend well-known results in linear control theory to the nonlinear domain. Despite the great interest in this area, many fundamental theoretical issues related to nonlinear control are currently still not well-understood. Consequently, many of the well-established theoretical results cannot be directly used for practical control.
c. **Uncertainty –** Certain essential information required in the mathematical model of the induction motor drive system, such as load, exact values of machine parameters, and noise, is unknown. Although some parameter identification and state estimation algorithms have been proposed to resolve the problem at the expense of more complex computations, the uncertainty problem has not been completely solved in practical applications.

Based on the nonlinear control theory as well as the human ability to comprehend, reason, and learn, intelligent techniques may be used to overcome the above difficulties.

2.2.7 *Intelligent Induction Motor Control Schemes*

Intelligence is defined as the ability to comprehend, reason, and learn. The definition of intelligent control from Åström and McAvoy has been used widely: 'An intelligent control system has the ability to comprehend, reason, and learn about processes, disturbances and operating conditions in order to optimize the performance of the process under consideration' (Åström and Björn, 1995). Intelligent control techniques are generally classified as expert-system control, fuzzy-logic control, neural-network control, and genetic algorithm (Bose, 1993). Intelligent induction motor control thus refers to the control of an induction motor drive using the above artificial intelligence techniques. The applications of expert-system, fuzzy-logic, neural-network, and genetic algorithm in induction motor drive system have been proposed in the literature (Bose, 1997b).

2.2.7.1 Expert-System Control Scheme (Bose, 1997a; Åström and Årzén, 1993)

Expert system is the forerunner among all the AI techniques, and from the beginning (1960s) to 1980s, both terms (expert system and artificial intelligence) have been used synonymously in the literature. Expert systems have been considered as a powerful method to solve control problems without having strict knowledge of mathematical description, particularly to deal with qualitative knowledge and reasoning with symbolic operation in a complex system. Expert systems have been used for choice of a.c. drive products, monitoring and diagnostics, design and simulation for a drive system. However, their applications in induction motor control are relatively few. In this book, an expert-system based acceleration control scheme is proposed. The acceleration control knowledge and human comparison strategies are employed so that the cumulative error due to evaluation of integrals and machine parameters effects of the classical vector controller can be eliminated.

2.2.7.2 Fuzzy-Logic Control Scheme (Bose, 1997b)

Fuzzy logic is another form of artificial intelligence, but its history and applications are more recent than expert systems. It is argued that human thinking does not always follow crisp 'yes-no' logic, but is often vague, uncertain, indecisive, or fuzzy. Based on this, Lofty Zadeh, a computer scientist, introduced the 'fuzzy logic' or fuzzy set theory in 1965 (Zadeh, 1965) that gradually emerged as a discipline in AI. The main characteristic of the fuzzy logic technique is to use the fuzzy rule sets and the linguistic representation of a human's knowledge to describe the controlled plant or to construct the fuzzy controller. A fuzzy-logic controller consists of fuzzification, fuzzy inference with rulebase and database, and defuzzification. Some fuzzy-logic controllers have already been designed for induction motor control, such as FOC with fuzzy efficiency optimizer and fuzzy-logic based DSC (Sousa, Bose, and Cleland, 1995; Mir, Zinger, and Elbuluk, 1994). In this book, a hybrid fuzzy/PI two-stage control scheme is proposed. In the scheme, the fuzzy-logic frequency controller and the PI current controller produce almost the same frequency and current magnitude control characteristics as a field-oriented controller. Effects of parameter variation, effects of noise in measured speed and input current, and effects of magnetic saturation are investigated.

2.2.7.3 Neural-Network Control Scheme (Bose, 1997b; Simoes and Bose, 1995)

Neural network is the most generic form of AI for emulation of human thinking compared to expert systems and fuzzy logic. In 1943, McCulloch and Pitts first proposed a network composed of binary-valued artificial neurons that were capable of performing simple threshold logic computations. The modern era of neural network with rejuvenated research practically started in 1982 when Hopfield presented his invention. Since then, many network models and learning rules have been introduced. The neural network is famous for its learning ability and arbitrary approximation to any continuous function. Research of the neural-network nonlinear dynamical control has been in progress since 1988 (Narendra and Mukhopadhyay, 1996). Recently, neural networks have been used for the parameter identification and state estimation of induction motor drive systems. Hybrid fuzzy and neural controller (also called a neuro-fuzzy controller) has been designed to control a 100 kW induction motor (Bose, Patel, and Rajashekara, 1997). Neural network with the advantage of parallel computation can be used to decrease the controller time-delay caused by complex computation. In this book, a neural-network-based DSC scheme is proposed to decrease the controller time delay so that the torque and flux errors of a DSC can almost be eliminated. The proposed neural-network controller employs the individual training strategy with the fixed-weight and the supervised methods (Kung, 1993).

2.2.7.4 Genetic Algorithm (Fogel, 1994)

Over the past 30 years, genetic algorithms were mainly developed in the USA by J. H. Holland, while evolutionary strategies were developed in Germany by I. Rechenberg and H.-P. Schwefel. Each of these constitutes a different approach, but they are both inspired by the principles of natural evolution. The GA is a stochastic global search method that mimics the metaphor of natural biological evolution. GA operates on a population of potential solutions applying the principle of survival of the fittest to produce (hopefully) better approximations to a solution. The most commonly used representation in GA is the binary alphabet {0, 1}, while there is an increasing interest in alternative encoding strategies, such as integer and real-coded representations (Wright, 1991). The GA differs substantially from the more traditional search and optimization methods, and the followings are the most significant:

1. GA searches a population of points in parallel instead of a single point.
2. GA does not require derivative information or other auxiliary knowledge; only the objective function and the corresponding fitness levels will influence the directions of search.
3. GA uses probabilistic transition rules instead of deterministic ones.

In this book, a real-coded genetic algorithm is proposed to optimize the extended Kalman filter (EKF) for estimating the rotor speed of an induction motor. It is shown that the real-coded GA is effective for optimizing the EKF performance of three different controllers, namely the closed-loop V/Hz controller, DSC, and FOC. The GA-EKF computer simulations for speed estimation show good noise rejection and they are less sensitive to machine parameter variations. Experiments are performed on a DSP-based FOC induction motor drive in order to verify the feasibility of the GA-EKF approach.

2.3 Induction Motor Control Algorithms

Different control algorithms are designed based on the control theory, principle, and hardware. Control algorithms of induction motor can be classified into three main categories (Rajashekara, Kawamura, and Matsuse, 1996; Bose, 1997b).

(I) Scalar control:	1. Open-loop voltage/frequency (V/Hz) control
	2. Slip frequency and voltage control
	3. Slip frequency and current control
(II) Vector control:	4. Direct FOC
	5. Indirect FOC
	6. Direct self control (DSC)
	7. Acceleration control
(III) Intelligent control:	8. Fuzzy control
	9. Neural-network control
	10. Expert-system control
	11. Genetic algorithm

A fictitious control system is shown in Figure 2.4 for explaining different induction motor control algorithms.

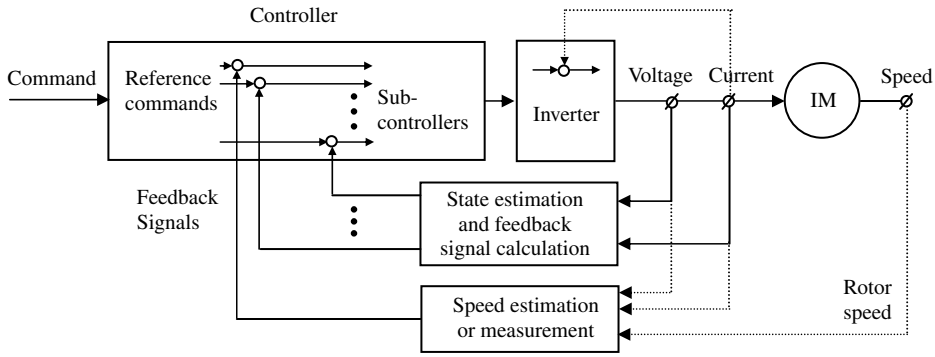


Figure 2.4 A fictitious control system at the algorithm level.

Figure 2.4 shows that a control algorithm can be characterized by the choice of feedback signals with the appropriate control function and control strategy. Various control algorithms are represented as follows:

1. Open-loop V/F control algorithm (Bose, 1981):
   a. Feedback signal $F = [0]$
   The open loop V/Hz control scheme has no feedback signal.
   b. Control function $u = \begin{bmatrix} |V_s| \\ \omega \end{bmatrix} = \begin{bmatrix} \text{const.} \times \omega^* \\ \omega^* \end{bmatrix}$
   where $|V_s|$ is stator voltage magnitude, ω is supply frequency, and ω^* is supply frequency command. The ratio $|V_s|/\omega^*$ is maintained constant.
   c. Control strategy: constant V/Hz control to maintain constant stator flux.

2. Slip frequency and voltage control algorithm (Trzynadlowski, 1994):
   a. Feedback signal $F = [\omega_o]$
   b. Control function $u = \begin{bmatrix} |V_s| \\ \omega \end{bmatrix} = \begin{bmatrix} \text{const.} \times \omega \\ \omega_o + \omega_r \end{bmatrix}$ and $\omega_r = f(\omega_o{}^* - \omega_o)$
   where $\omega_o{}^*$ is rotor speed command and ω_r is slip frequency.
   c. Control strategy: slip frequency control and constant V/Hz control. Setting the limit on the slip speed in the vicinity of the peak torque point provides fast response of the drive system according to the speed command.
3. Slip frequency and current control algorithm (Rajashekara, Kawamura, and Matsuse, 1996):
   a. Feedback signal $F = \begin{bmatrix} \omega_o \\ |i_s| \end{bmatrix}$
   b. Control function $u = \begin{bmatrix} |i_s| \\ \omega \end{bmatrix} = \begin{bmatrix} f_1(\omega_o{}^* - \omega_o) \\ \omega_o + \omega_r \end{bmatrix}$ and $\omega_r = f_2(\omega_o{}^* - \omega_o)$
   c. Control strategies: slip frequency control and current control. Setting the limit on the slip speed in vicinity of the peak torque point provides fast speed response of the drive system according to the reference speed.
4. Direct FOC algorithm (Trzynadlowski, 1994):
   The field orientation control algorithms can be divided into two types: direct FOC and indirect FOC. In the direct FOC scheme, the rotor flux λ_r^e, torque T, and rotor flux angle θ_r are feedback signals which are obtained from the air-gap flux (or stator voltage) and stator current signals. In the indirect FOC scheme, the rotor flux λ_r^e and torque T are control commands, while the rotor flux angle θ_r is the feedback signal which is obtained by summation of the rotor speed and reference slip frequency.
   In direct field-oriented control,
   a. Feedback signal $F = \begin{bmatrix} \theta_r \\ i_{ds}^s \\ i_{qs}^s \\ \lambda_{dr}^s \\ T_L \end{bmatrix} = q(x)$, where $x = \begin{bmatrix} i_{ds}^s \\ i_{qs}^s \\ \lambda_{dr}^s \\ \lambda_{qr}^s \end{bmatrix}$ or $x = \begin{bmatrix} i_{ds}^s \\ i_{qs}^s \\ V_{ds}^s \\ V_{qs}^s \end{bmatrix}$
   b. Control function $u = \begin{bmatrix} i_{ds}(T^* - T, \lambda_{dr}^s{}^* - \lambda_{dr}^s, \theta_r) \\ i_{qs}(T^* - T, \lambda_{dr}^s{}^* - \lambda_{dr}^s, \theta_r) \end{bmatrix}$
   with the field-orientation conditions:

$$\lambda_{qr}^e = 0 \quad \text{and} \quad \lambda_{dr}^e = \text{const.}$$

   c. Control strategies: coordinate transformations, current controlled and field-orientation conditions.
5. Indirect FOC algorithm (Trzynadlowski, 1994):
   a. Feedback signal $F = [\theta_r]$
   b. Control function $u = \begin{bmatrix} i_{ds}^s(T^* - T, \lambda_{dr}^s{}^* - \lambda_{dr}^s, \theta_r) \\ i_{qs}^s(T^* - T, \lambda_{dr}^s{}^* - \lambda_{dr}^s, \theta_r) \end{bmatrix}$

with the field-orientation conditions:

$$\lambda_{qr}^{e} = 0 \quad \text{and} \quad \lambda_{dr}^{e} = \text{const.}$$

Control strategies: coordinate transformations, current controlled and field orientation conditions.

6. DSC control algorithm (Baader, 1992):

a. Feedback signal $F = \begin{bmatrix} \left|\lambda_{\mu}^{s}\right| \\ \theta_{\mu} \\ T \end{bmatrix} = q(x)$, where $x = \begin{bmatrix} i_{ds}^{s} \\ i_{qs}^{s} \\ V_{ds}^{s} \\ V_{qs}^{s} \end{bmatrix}$

b. Control function $u = \begin{bmatrix} S_a\left(V_{ds}^{s}, V_{qs}^{s}, i_{ds}^{s}, i_{qs}^{s}, T^{*}, \left|\lambda_{\mu}\right|^{*}\right) \\ S_b\left(V_{ds}^{s}, V_{qs}^{s}, i_{ds}^{s}, i_{qs}^{s}, T^{*}, \left|\lambda_{\mu}\right|^{*}\right) \\ S_c\left(V_{ds}^{s}, V_{qs}^{s}, i_{ds}^{s}, i_{qs}^{s}, T^{*}, \left|\lambda_{\mu}\right|^{*}\right) \end{bmatrix}$

S_a, S_b, and S_c are the status of inverter switches.

c. Control strategy: hysteresis (bang-bang) control and optimum switching table.

The principle of direct self control for the torque and the flux is based on hysteresis control with an optimum switching table. In this system, the instantaneous values of the flux and torque are calculated from the stator voltage and current. The flux and torque can then be controlled directly and independently by selecting the optimum inverter switching modes. The selection is made so as to restrict the errors of the flux and torque within the hysteresis bands and to obtain fast torque response with a low inverter switching frequency and low harmonic losses.

7. Acceleration control algorithm (Shi, 1997):

a. Feedback signal $F = \begin{bmatrix} a \\ \theta(i_s) \end{bmatrix} = q(x)$ where $x = \begin{bmatrix} \omega_o \\ i_{ds} \\ i_{qs} \end{bmatrix}$

b. Control function $u = \begin{bmatrix} S_a(\theta(i_s), a^{*}, a) \\ S_b(\theta(i_s), a^{*}, a) \\ S_c(\theta(i_s), a^{*}, a) \end{bmatrix}$

S_a, S_b, and S_c are the status of inverter switches, a is rotor acceleration and a^{*} is acceleration command.

c. Control strategy: flux angle reasoning, acceleration comparison and expert system principle.

In the acceleration control scheme, the approximate flux angles are obtained by an inference system from the stator current angles and acceleration states. The inverter switching modes are determined by comparing two accelerations produced by different voltage vectors with respect to the stator current angle. The acceleration and stator current angle are used as feedback signals.

8. Fuzzy control algorithm (Sousa and Bose, 1994; Tang and Xu, 1994):
   a. Fuzzy controller input $F_{input} = \begin{bmatrix} R \\ E \\ \Delta E \end{bmatrix}$
   where R is the feedback signal, E is the error signal, and ΔE is the change in error signal.
   b. Fuzzification $\begin{bmatrix} F_{linguistic} \\ \mu_{input} \end{bmatrix} = f(F_{input})$
   where $F_{linguistic}$ denotes the input fuzzy linguistic value and μ_{input} denotes the degree of membership of the input.
   c. Fuzzy inference $\begin{bmatrix} u \\ \mu_{output} \end{bmatrix} = K\left(\begin{bmatrix} F_{linguistic} \\ \mu_{input} \end{bmatrix}\right)$
   where u denotes the crisp value of output fuzzy linguistic value and μ_{output} denotes the fuzzy degree of membership of the output.
   d. Defuzzification $y = g\left(\begin{bmatrix} u \\ \mu_{output} \end{bmatrix}\right)$
   where y denotes the output crisp value required by the plant.

   The control algorithm is based on the fuzzy set theory. A fuzzy control algorithm consists of fuzzification with a database, a fuzzy logic inference based on a rulebase, and defuzzification. The fuzzification operation implements the process of converting the crisp input values to fuzzy sets. The fuzzy set consists of elements each having a degree of membership and associated with linguistic values. The defuzzification operation is the process of determining the best numerical value to represent a given fuzzy set. The database stores memberships of fuzzy variables. The rulebase provides the necessary linguistic control rules for the fuzzy inference.
9. Neural-network control algorithm (Bose, 1997a):
   a. Input-output pair $\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} g(t) \\ IM(x) \end{bmatrix}$
   where x is the input samples, z is the outputs,
   $g(t)$ denotes the generation function of input samples, and IM denotes the plant outputs.
   b. Training $\begin{bmatrix} w \\ b \end{bmatrix} = R_e\left\{f\left(\begin{bmatrix} x \\ z \end{bmatrix}\right)\right\}$
   where w is the weight of network, b is the bias of network, f denotes the activation function, and R_e denotes the training algorithm.
   c. Implementation $z' = f(wx' + b)$
   where z' denotes the output of neural-network and x' denotes the practical inputs.

   A neural model is mathematically represented by a basis function [w, b] and an activation function f(.). The selection of these functions often depends on the applications of neural network. Neural network is used to approximate the control function of induction motor through training procedures.
10. Expert system control algorithm (Åström and Årzén, 1993; Lu, 1996; Bose, 1997a):
    a. Knowledge acquisition $K_{base} = f(E_{xpert})$
    where K_{base} denotes the knowledge base, E_{xpert} denotes the expert knowledge, and f is the knowledge acquisition procedure.

b. Input and output interfaces $x' = g(x)$, and $x = h(z')$
where x is the electrical signal. x' and z' denote the numerical and linguistic codes. $G(.)$ implements the electrical signal encoding and $h(.)$ implements the numerical and linguistic decoding.

c. Inference engine $z' = I_{nference}\left(\begin{bmatrix} K_{base} \\ x' \end{bmatrix}\right)$
where $I_{nference}$ denotes the inference procedure.

d. User interface $L = e(E_{xecution})$

$$K'_{base} = f'(U_{ser}, K_{base})$$

where $E_{xecution}$ is the execution of the rules, L is natural language, U_{ser} denotes a user, $e(.)$ is an explanation function, and f' denotes the modification procedure of the knowledge base.

Based on the knowledge acquired from control experts in encoded form, the control function is derived by the on-line inference engine. The humanlike knowledge-inference system may be applied to cope with complex induction motor drive systems, or those with parameter uncertainties.

11. Genetic algorithm in control:

a. Input-output pair $\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} g(t) \\ IM(x) \end{bmatrix}$
where x denotes the input samples, z denotes the outputs, $g(t)$ denotes the generation function of input samples, and IM denotes the plant outputs.

b. Coding parameters $S = M(P)$
where P is controller parameters, M is coding method, and S is a genetic string.

c. Reproduction with fitness evaluation $S_{n+1} = R(S_n)$
where S_{n+1} is the new reproduction and R denotes the reproduction algorithm with fitness evaluation.

d. Crossover and Mutation $S_{n+2} = CM(S_{n+1})$
where CM is the crossover and mutation method.

e. Decoding parameters $P_L = N(S_L)$
where N is decoding method, S_L is an optimum string, and P_L is optimum controller parameters.

Genetic algorithm is a search algorithm based on the mechanics of natural selection and natural genetics. GA has the properties that make it a powerful technique for optimizing controller parameters for an induction motor drive.

2.4 Speed Estimation Algorithms

The speed sensorless strategies have aroused great interest among induction motor control researchers. In these strategies, the motor speed is estimated and used as a feedback signal for closed-loop speed control. These algorithms may be classified into the following categories (Rajashekara, Kawamura, and Matsuse, 1996; Holtz, 1996; Ilas *et al.*, 1996):

1. **Open loop with slip compensation algorithm** (Rajashekara, Kawamura, and Matsuse, 1996). The rotor speed is obtained from summation of synchronous speed and slip speed which is estimated from the load. This algorithm can only be used in the steady-state case.

Model:	Steady-state torque-speed relationship.
Speed estimation function:	$\omega_o = \omega + \omega_r(T_L)$

2. **Slip frequency algorithm** (Abbondanti and Brennen, 1975). The slip speed is calculated based on a steady-state model of the motor.

Model:	Steady-state 'T' equivalent circuit
Speed estimation function:	$\omega_o = (\omega, i_s^s, V_s^s)$

3. **Speed estimation algorithm using state equations** (Joetten and Maeder, 1983). The rotor speed can be calculated from the state equation Equation (2.14), which can be expressed as

Model:	the 11^{th} fifth-order model with 'T' equivalent circuit
Speed estimation function:	$\omega_o = f(V_{ds}^s, V_{qs}^s, i_{ds}^s, i_{qs}^s, \lambda_{dr}^s, \lambda_{qr}^s)$

4. **Flux estimation and flux vector control algorithm** (Xu and Doncker, 1988). The rotor speed is obtained by estimating the synchronous speed and slip speed. The synchronous speed is replaced by stator flux speed and the slip speed is estimated based on machine parameters of the motor.

Model:	the first fifth-order equation with 'Γ' equivalent circuit
Speed estimation function:	$\omega_o = f(V_{ds}^s, V_{qs}^s, i_{ds}^s, i_{qs}^s, \lambda_{d\mu}^s, \lambda_{q\mu}^s)$

5. **Observer algorithm** (Cuzner, 1990; Jansen, 1996). The fluxes are calculated by a voltage model and a current model, separately. The speed is estimated based on the difference between two fluxes.

Model:	Voltage model and current model
Speed estimation function:	$\omega_o = f(V_{ds}, V_{qs}, i_{ds}, i_{qs}, \lambda_{dr}, \lambda_{qr}, k_p, k_I)$

where K_p and K_I are the adaptation mechanism gains.

6. **Model reference adaptive algorithm** (Trzynadlowski, 1994). A comparison is made between the outputs of two estimators. The first estimator, which does not involve the estimated rotor speed, is considered as a reference model of the induction motor. The second estimator, which involves the estimated rotor speed, is regarded as an adjustable model. The error between the outputs of the two estimators is used to derive a suitable adaptation mechanism that generates the estimated rotor speed to modify the adjustable model.

Model:	Reference model of induction motor and adjustable model
Speed estimation function:	$\omega_o = f(\varepsilon_\alpha, \varepsilon_\beta, \tilde{\varepsilon}_\alpha, \tilde{\varepsilon}_\beta, k_p, k_I)$

where K_p and K_I are the adaptation mechanism gains, ε_α and ε_β are outputs of the reference model of induction motor, and $\tilde{\varepsilon}_\alpha$ and $\tilde{\varepsilon}_\beta$ are outputs of the adjustable model. The outputs may be flux, counter emf, reactive power, and so on.

7. **Extended Kalman filter estimation algorithm** (Salvatore, Stasi, and Tarchioni, 1993; Kim, Sul, and Park, 1996). Extended Kalman filter is employed to estimate the rotor speed, based on the measured stator currents and voltages. Kalman filter algorithm is based on complete electrical model of induction motor for determination of the system state. The rotor speed can be determined based on the measured voltages and currents. Using the state equations and Kalman filter, the rotor speed (an extended state) is estimated.

Model:	Complete electrical model of induction motor with an extended state, rotor speed.
Speed estimation function:	$\omega_o = f(V_{ds}, V_{qs}, i_{ds}, i_{qs}, \omega_o)$

8. **Neural network algorithm** (Simoes and Bose, 1995). Neural network algorithm is based on a learning process. Neural networks have the advantages of extremely fast parallel computation and fault tolerance characteristics due to distributed network intelligence, which would be ideal for speed estimation of an induction motor.

Model:	Neural network
Speed estimation function:	$\omega_o = f(i_{ds}, i_{qs}, \lambda_{ds}, \lambda_{qs})$

All speed sensorless algorithms depend on the mathematical model of the induction motor.

2.5 Hardware

Hardware implementation of induction motor control involves three essential components, and hence three important technical areas:

1. Induction motor (involving electrical machine techniques)
2. Power inverter (involving power electronics techniques)
3. Controller (involving computer and control techniques).

A scalar controller is usually implemented with analog electronic components, and a vector controller is implemented with a digital device DSP or a microcontroller. The fuzzy control algorithm may be implemented with a special fuzzy microcontroller or a DSP, while the expert system algorithm may be implemented with DSP. The DSP based controller is the current trend of induction motor control. With the ability of implementing complex computations, a DSP such as ADMC331 is a suitable solution to implement high-performance algorithms and intelligent algorithms for induction motor drives.

Based on an understanding of the induction motor control system, a taxonomy of induction motor control is given in Table 2.1.

Table 2.1 Taxonomy of induction motor control.

Algorithm	Open-loop V/Hz control	Slip and voltage control	Slip and current control		Fuzzy slip and current control
Feedback signal	$F = \{\}$	$F = \{\text{speed}\}$	$F = \{\text{current, speed}\}$		
Control function	$u = \{\lvert V_s^s \rvert, \omega\}$		$u = \{\lvert i_s^s \rvert, \omega\}$		
Control strategies	Constant V/Hz control	Slip control based on static torque-speed characteristic	Slip control based on static torque-speed characteristic	Fuzzy-logic	
	Open-loop control	Constant V/Hz control	PI current magnitude control	Slip control based on features of FOC	
		Closed-loop control	Closed-loop control	PI current magnitude control Closed-loop control	
Characteristics	Slow response Robustness	Medium-slow response Robustness	Medium response Robustness	Medium-fast response Robustness	
Hardware	Electronic components PWM inverter		Electronic components PWM current-controlled inverter	Fuzzy device PWM current-controlled inverter	

Algorithm	Direct FOC	Indirect FOC	ANN-FOC Fuzzy-logic FOC	DSC	ANN-DSC Fuzz-logic DSC	Expert-system based Acceleration control
Feedback signal	$F = \{$flux, voltage, current, speed$\}$	$F = \{$speed$\}$		$F = \{$voltage, current, speed$\}$		$F = \{$current, speed$\}$
Control function	$u = \{i_{qs}, i_{ds}\}$			$u = \{S_a, S_b, S_c\}$		$u = \{S_a, S_b, S_c\}$
Control strategies	Coordinate transformation		Implementation of neural network	Bang-bang control	Implementation of neural network	Acceleration comparison
	PI control		Controller is partly implemented by fuzzy logic	Optimum switch table	Controller is partly implemented by fuzzy logic	Flux angle estimation
	Flux and torque estimation			Flux and torque estimation		Bang-bang control
	Speed estimation			Speed estimation		Production system
	Parameter identification			Parameter identification		

(continued)

Table 2.1 (*Continued*)

Algorithm	Direct FOC	Indirect FOC	ANN-FOC Fuzzy-logic FOC	DSC	ANN-DSC Fuzz-logic DSC	Expert-system based Acceleration control
Characteristics	Fast response Sensitive to motor parameter changes		Fast response	Fast response Integral drift	Fast response Integral drift	Fast response Sensitive to speed sensor noise Robustness against parameter changes and current noise Discontinuous control
Hardware	DSP PWM Current-controlled inverter		ANN and fuzzy devices PWM, Current-controlled inverter	DSP Inverter	ANN and fuzzy devices Inverter	DSP Inverter Speed sensor

References

Abbondanti, A. and Brennen, M.B. (1975) Variable speed induction motor drives use electronic slip calculator based on motor voltage and currents. *IEEE Transactions on Industry Applications*, **IA-11**(5), 483–488.

Åström, K.J. and Årzén, K.E. (1993) Expert control, in *An Introduction to Intelligent and Autonomous Control* (eds P.J. Antsaklis, K.M. Passino, and MA. Norwell) Kluwer Academic Publishers, pp. 163–168.

Åström, K.J. and Björn, W. (1995) *Adaptive Control*, Addison Wesley Publishing Company, Reading, MA.

Baader, U., Depenbrock, M. and Gierse, G. (1992) Direct Self Control (DSC) of inverter-fed induction machine: a basis for speed control without speed measurement. *IEEE Transactions on Industry Applications*, **28**(3), 581–589.

Blashke, F. (1972) The principle of field-orientation as applied to the new 'transvektor' closed-loop control system for rotating-field machines. *Simians Review*, **34**(5), 21–220.

Bose, B.K. (1981) *Adjustable Speed AC Drive Systems*, IEEE Press, New York.

Bose, B.K. (1993) Power electronics and motion control-technology status and recent trends. *IEEE Transactions on Industry Applications*, **29**, 902–909.

Bose, B.K. (1997a) Expert system, fuzzy logic, and neural networks in power electronics and drives, in *Power Electronics and Variable Frequency Drives: Technology and Applications* (ed. B.K. Bose), IEEE Press, New Jersey.

Bose, B.K. (1997b) Intelligent control and estimation in power electronics and drives. *The 1997 IEEE International Electric Machines and Drives Conference,* USA, May.

Bose, B.K., Patel, N.R., and Rajashekara, K. (1997) A neuro-fuzzy based on-line efficiency optimization control of a stator flux oriented direct vector controlled induction motor drive. *IEEE Transactions on Industrial Electronics*, **44**, 270–273.

Chan, C.C., Leung, W.S., and Ng, C.W. (1990) Adaptive decoupling control of induction motor drives. *IEEE Transactions on Industrial Electronics*, **37**(1), 41–47.

Cuzner, R.M., Lorenz, R.D., and Novotny, D.W. (1990) Application of nonlinear observers for rotor position detection on an induction motor using machine voltages and currents. *IEEE Industry Applications Society*, 416–421.

Delgado, A., Kambhampati, C., and Warwick, K. (1995) Dynamic recurrent neural network for system identification and control. *IEE Proceedings – Control Theory and Applications*, **142**(4), 307–314.

Depenbrock, M. (Inventor) (18, Oct. 1985) Direct Self-control of the Flux and Rotary Moment of a Rotary-field Machine, United States Patent 4,678,248.

Fogel, D.B. (1994) An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, **5**(1), 3–14.

Hasse, K. (1969) 'About the Dynamics of Adjustable-speed Drives with Converter-fed Squirrel-cage Induction Motors' (in German), Dissertation, Darmstadt Technische Hochschule.

Holtz, J. (1996) Methods for speed sensorless control of AC drives, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura, and K. Matsuse), IEEE Press, New Jersey.

Ilas, C., Bettini, A., Ferraris, L. *et al.* (1996) Comparison of different schemes without shaft sensors for field oriented control drives, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura, and K. Matsuse) IEEE Press, New Jersey.

Jansen, P.L., Lorenz, R.D., and Novotny, D.W. (1996) Observer-based direct field orientation: analysis and comparison of alternative methods, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura, and K. Matsuse) IEEE Press, New Jersey.

Joetten, R. and Maeder, G. (1983) Control methods for good dynamic performance induction motor drives based on current and voltage as measured quantities. *IEEE Transactions on Industry Applications*, **IA-19**(3), 356–363.

Kim, Y.R., Sul, S.K., and Park, M.H. (1996) Speed sensorless vector control of induction motor using extended kalman filter, in *Sensorless Control of AC Motor Drives*, IEEE Press, New Jersey, pp. 215–223.

Krause, P.C., Wasynczuk, O., and Sudhoff, S.D. (1995) *Analysis of Electric Machinery*, IEEE Press, New Jersey

Kung, S.Y. (1993) *Digital Neural Networks*, PTR Prentice-Hall, Inc., New Jersey.

Lu, Y.Z. (1996) *Industrial Intelligent Control: Fundamentals and Applications*, John Wiley & Sons Ltd, Chichester.

Marino, R. and Tomei, P. (1995) *Nonlinear Control Design*, Prentice Hall Europe, London.

Mir, S.A., Zinger, D.S., and Elbuluk, M.E. (1994) Fuzzy controller for inverter fed induction machines. *IEEE Transactions on Industry Applications*, **30**(1), 78–84.

Narendra, K.S. and Mukhopadhyay, S. (1996) Intelligent control using neural networks, in *Intelligent Control Systems: Theory and Applications* (eds M.M. Gupta and N.K. Sinha) IEEE Press, New Jersey.

Rajashekara, K., Kawamura, A., and Matsuse, K. (1996) Speed sensorless control of induction motor, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura, and K. Matsuse), IEEE Press, New Jersey.

Salvatore, L., Stasi, S., and Tarchioni, L. (1993) A new EKF-based algorithm for flux estimation in induction machines. *IEEE Transactions on Industrial Electronics*, **40**(5), 496–504.

Shi, K.L., Chan, T.F., and Wong, Y.K. (1997) A new direct self-control scheme for an inverter-fed induction motor. Second International Power Electronics and Motion Control Conference, Hangzhou, November.

Simoes, M.G. and Bose, B.K. (1995) Neural network based estimation of feedback signals for a vector controlled induction motor drive. *IEEE Transactions on Industry Applications*, **31**, 620–629.

Sousa, G.C.D. and Bose, B.K. (1994) A fuzzy set theory based control of a phase-controlled converter DC machine drive. *IEEE Transactions on Industry Applications*, **30**(1), 34–44.

Sousa, G.C.D., Bose, B.K., and Cleland, J.G. (1995) Fuzzy logic based efficiency optimization control of an indirect vector controlled induction motor drive. *IEEE Transactions on Industrial Electronics*, **42**(2), 192–198.

Tang, Y.F. and Xu, L.Y. (1994) Fuzzy logic application for intelligent control of a variable speed drive. *IEEE Transactions on Energy Conversion*, **9**(4), 679–685.

Trzynadlowski, A.M. (1994) *The Field Orientation Principle in Control of Induction Motors*, Kluwer Academic Publishers, Boston.

Vidyasagar, M. (1993) *Nonliear Systems Analysis*, Prentice-Hall Inc., New Jersey.

Wright, A.H. (1991) Genetic algorithms for real parameter optimization, in *Foundations of Genetic Algorithms* (ed. J.E. Rawlins), Morgan Kaufmann, San Francisco, CA, pp. 205–218.

Xu, X. and Doncker, R.D. (1988) A stator flux oriented induction machine drive. *IEEE Power Electronics Specialists Conference*, pp. 870–876.

Zadeh, L.A. (1965) Fuzzy sets. *Information and Control*, **8**, 338–353.

3

Modeling and Simulation of Induction Motor[1]

3.1 Introduction

Evaluation of an induction motor control system can be performed in two ways. The first approach (at hardware level) uses a real motor with an inverter and a controller. The controller is usually constructed using electronic devices, such as ICs (integrated circuits), or a DSP (digital signal processor) with the algorithm coded in assembler or 'C' language. The advantages of the experimental approach are that it includes the actual noise present, the practical inverter voltage and current waveforms, and sensor characteristics that may not be included in a computer simulation. The main disadvantage of the practical approach is that the experimental results are valid only for the system being investigated such as motor type, rating, and inverter supply. Besides, some of the electrical parameters may be difficult or impossible to measure. The second approach (at algorithm level) is a computer simulation of the system (Krause and Thomas, 1965; Chan, Jiang and Chen, 1993; Mohan *et al.*, 1997). In the computer simulation environment, all quantities can be readily observed and the parameters can be altered to investigate their effect on the system, thereby providing useful information for controller design. Process noise and sensor noise can be added to simulate the performance. Motor type, size, and power supply may be changed easily. Development of a new motor drive often involves expensive prototypes. Advanced computer simulation and mathematical modeling techniques with solution procedures can produce optimal designs with minimum time, cost, and effort.

[1] (a) Portions reprinted from K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling and simulation of the three-phase induction motor," *International Journal on Electrical Engineering Education*, **36**(2), 163–172, © 1999, with permission from Manchester University Press.
(b) Portions reprinted by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK," The 1997 IEEE Biennial International Electrical Machines and Drives Conference, Paper WB3-6, May 18–21, 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.
(c) Portions reprinted from K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Modeling and simulation of a novel two-stage controller for an induction motor," *International Association of Science and Technology for Development (IASTED) Journal on Power and Energy Systems*, **19**(3), 257–264, © 1999, with the permission from ACTA Press.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

Modeling and simulation of the three-phase induction machine is well documented in the literature and digital computer solution can be performed using several methods, such as numeric programming, symbolic programming and the electromagnetic transient program (EMTP) (Domijan and Yin, 1994). Throughout this book, MATLAB®/Simulink software will be used for the dynamic modeling of the induction motor and simulation studies. The main advantage of Simulink over other programming softwares is that, instead of compilation of program code, the simulation model is built up systematically by means of basic function blocks. A set of machine differential equations can thus be modeled by interconnecting appropriate function blocks, each of which performing a specific mathematical operation. Programming efforts are drastically reduced and error debugging is easy. Since Simulink is a model operation program, the simulation model can be easily developed by addition of new sub-models to cater for various control functions. As a sub-model, for example, the induction motor could be incorporated in a complete electric motor drive system.

3.2 Modeling of Induction Motor

Three simulation models of induction motor are developed using MATLAB®/Simulink in this chapter. They are the current-input model, the voltage-input model, and the discrete-state model, which can be used to study different induction motor drive systems. The current-input model is more suitable for the study of a current-controlled drive system, while the voltage-input model and the discrete-state model are more suitable for the study of inverter-fed induction motor control.

The current-input model is based on Equation (2.15), the reduced-order equation of the 11th fifth-order equation in the excitation reference frame, while the discrete-state model is based on a discrete-time form of Equation (2.14), the 11th fifth-order equation in the stator reference frame. Modeling of the voltage-input model is based on the 8th fifth-order equation, which can be expressed in matrix form as follows (Trzynadlowski, 1994):

$$\frac{d}{dt}\begin{bmatrix} i_{ds}^s \\ i_{qs}^s \\ i_{dr}^s \\ i_{qr}^s \\ \omega_o \end{bmatrix} = \frac{1}{L_c^2}\begin{bmatrix} -R_sL_r & \frac{P}{2}\omega_oL_M^2 & R_rL_M & \frac{P}{2}\omega_oL_rL_M & 0 \\ -\frac{P}{2}\omega_oL_M^2 & -R_sL_r & -\frac{P}{2}\omega_oL_rL_M & R_rL_M & 0 \\ R_sL_M & -\frac{P}{2}\omega_oL_sL_M & -R_rL_s & -\frac{P}{2}\omega_oL_sL_M & 0 \\ \frac{P}{2}\omega_oL_sL_M & R_sL_M & \frac{P}{2}\omega_oL_sL_r & -R_rL_s & 0 \\ \frac{PL_ML_c^2}{3J}i_{qs} & -\frac{PL_ML_c^2}{3J}i_{ds} & 0 & 0 & -\frac{C_f}{J} \end{bmatrix}\begin{bmatrix} i_{ds}^s \\ i_{qs}^s \\ i_{dr}^s \\ i_{qr}^s \\ \omega_o \end{bmatrix} + \frac{1}{L_c^2}\begin{bmatrix} L_r & 0 & -L_M & 0 & 0 \\ 0 & L_r & 0 & -L_M & 0 \\ -L_M & 0 & L_s & 0 & 0 \\ 0 & -L_M & 0 & L_s & 0 \\ 0 & 0 & 0 & 0 & -\frac{L_c^2}{J} \end{bmatrix}\begin{bmatrix} V_{ds}^s \\ V_{qs}^s \\ 0 \\ 0 \\ T_L \end{bmatrix} \tag{3.1}$$

where $L_c = \sqrt{L_sL_r - L_M^2}$.

Rotating transformation from *abc* to *dq* coordinates, also called 3/2 rotating transformation, projects balanced three-phase quantities (voltages or currents) to rotating two-axis coordinates at a given angular velocity. The transformation is defined as follows:

$$\begin{bmatrix} V_d \\ V_q \end{bmatrix} = C_{dq} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ \sin(\theta) & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = C_{abc} \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \cos\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta - \frac{2\pi}{3}\right) \\ \cos\left(\theta + \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \begin{bmatrix} V_d \\ V_q \end{bmatrix} \tag{3.3}$$

where $C_{dq} \times C_{abc} = 2/3$.

Note that the transform matrices and multiplying factors are the same for both voltages and currents.

In the original Park's transformation, the factor C_{dq} for *abc* to *dq* transformation is 2/3, while the factor C_{abc} for *dq* to *abc* transformation is 1. In some publications, however, C_{dq} and C_{abc} are both defined to be $\sqrt{2/3}$. In this book, C_{dq} is defined as 1 and C_{abc} as 2/3 (Trzynadlowski, 1994). For example, the peak value of a line-line voltage of 220 V is $220 \times \sqrt{2} = 311.13$ V and the peak value of each phase voltage is $220 \times \sqrt{2}/\sqrt{3} = 179.63$ V. After a rotating transformation from *abc* to *dq* coordinate, the peak value of V_d or V_q is $220 \times \sqrt{2} \times C_{dq} \times \frac{3}{2} \times \frac{1}{\sqrt{3}} = 269.444 \times C_{dq}$. When $C_{dq} = 1$, the peak value of V_d or V_q is 269.444 V. Accordingly, the multiplying factor in the torque equation of the induction motor may be different depending on the value of C_{dq} defined.

Another transformation, which converts three phase quantities to two phase quantities without rotation of coordinates, is called the *abc* to $\alpha\beta$ coordinate transformation. It is used to simplify the analysis of three-phase quantities, and is defined as follows.

$$\begin{bmatrix} V_{ds} \\ V_{qs} \end{bmatrix} = C_{dqs} \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \tag{3.4}$$

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = C_{abcs} \begin{bmatrix} 1 & 0 \\ -1/2 & \sqrt{3}/2 \\ -1/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} V_{ds} \\ V_{qs} \end{bmatrix} \tag{3.5}$$

where $C_{dqs} \times C_{abcs} = 2/3$.

In this book, $\alpha\beta$ axes are defined as *ds-qs*. C_{dqs} is 1 and C_{abcs} in the $\alpha\beta$ transform is 2/3 (Trzynadlowski, 1994).

The various induction motor models being studied are summarized in Table 3.1.

Table 3.1 Induction motor models.

Models	Equivalent Circuit and Reference Frame	Modeling Equations	Computer Execution Time	Applications
Current-input model	'T' in excitation reference frame	Equation (2.15)	Fast	Current-controlled drive
Voltage-input Model	'T' in stator reference frame	Equation (3.1)	Medium	Voltage-source invert-fed drive
Discrete-state model	'T' in stator reference frame	Discrete form of Equation (2.14)	Slow	State-space expression

3.3 Current-Input Model of Induction Motor

The current-input model is based on the reference-frame theory of coordinate transformation (Krause, Wasynczuk and Sudhoff, 1995). The simulation model is implemented using MATLAB®/Simulink and it possesses the following characteristics:

1. It takes current source and load torque as inputs and gives the rotor speed as output.
2. The parameters may be continuously changed.
3. The model may be easily expanded and has a good user interface.

The current-input model of an induction motor consists of (1) a current (3/2) transformation sub-model, (2) an electrical sub-model, and (3) a mechanical sub-model, as shown in Figure 3.1.

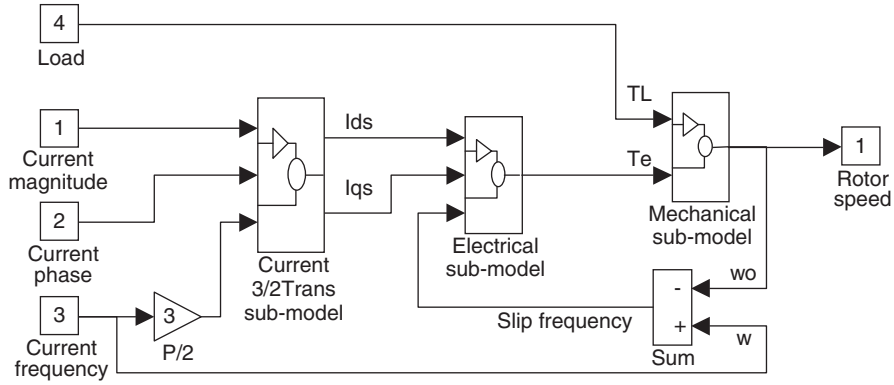


Figure 3.1 Current-input model of an induction motor in Simulink.

3.3.1 Current (3/2) Rotating Transformation Sub-Model

The current (3/2) rotating transformation sub-model consists of a current source block and a current 3/2 rotating transformation block. The current source block is used to build a current source based on input frequency, phase, and magnitude. The current (3/2) rotating transformation block is used to convert the three-phase stator current to the corresponding vectors in the excitation reference frame.

The current source block is based on the following current equation.

$$\begin{cases} i_{as} = A\cos\left(\int \omega dt\right) \\ i_{bs} = A\cos\left(\int \omega dt - 2\pi/3\right) \\ i_{cs} = A\cos\left(\int \omega dt + 2\pi/3\right) \end{cases} \tag{3.6}$$

The representation of one supply phase using Simulink blocks is shown in Figure 3.2.

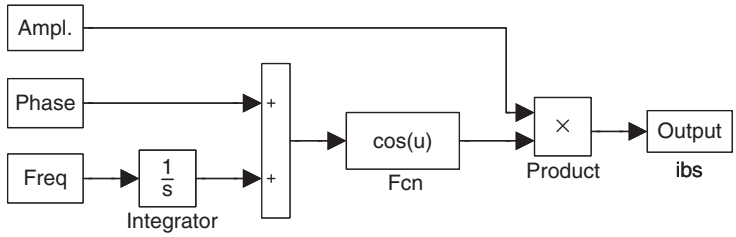


Figure 3.2 Simulink block diagram of one supply phase. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

The three-phase to two-phase (3/2) rotating transformation block is based on the current equations for phase transformation Equation (3.7).

$$\begin{bmatrix} i_{ds}^e \\ i_{qs}^e \end{bmatrix} = \begin{bmatrix} \cos\left(\int \omega dt\right) & \cos\left(\int \omega dt - 2\pi/3\right) & \cos\left(\int \omega dt + 2\pi/3\right) \\ \sin\left(\int \omega dt\right) & \sin\left(\int \omega dt - 2\pi/3\right) & \sin\left(\int \omega dt + 2\pi/3\right) \end{bmatrix} \begin{bmatrix} i_{as} \\ i_{bs} \\ i_{cs} \end{bmatrix} \tag{3.7}$$

The representation of 3/2 rotating transformation by Simulink blocks is shown in Figure 3.3.

The current source blocks and the 3/2 rotating transformation blocks are grouped together to form the current 3/2 rotating transformation sub-model in the induction motor model.

3.3.2 Electrical Sub-Model

Let the rotor time constant $\tau_r = L_r/R_r$. Equation (2.15) can be rewritten as:

$$T = \frac{P}{3R_r}\frac{L_M}{\tau_r}\left(i_{qs}^e \lambda_{dr}^e - i_{ds}^e \lambda_{qr}^e\right) \tag{3.8}$$

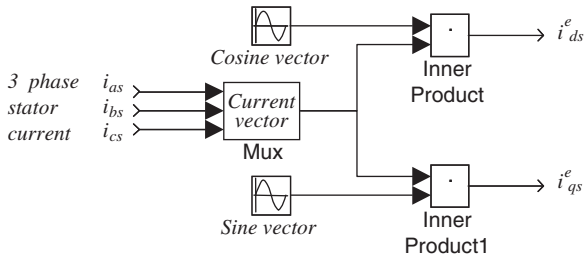


Figure 3.3 Simulink blocks of 3/2 rotating transformation sub-model. (Reproduced by permission of K. L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

$$\lambda^e_{dr} = \frac{1}{p}\left(\frac{L_M}{\tau_r} i^e_{ds} - \frac{1}{\tau_r}\lambda^e_{dr} + \omega_r \lambda^e_{qr}\right) \tag{3.9}$$

$$\lambda^e_{qr} = \frac{1}{p}\left(\frac{L_M}{\tau_r} i^e_{qS} - \frac{1}{\tau_r}\lambda^e_{qr} - \omega_r \lambda^e_{dr}\right). \tag{3.10}$$

Figure 3.4 shows the electrical sub-model of an induction motor as described by Equations (3.8), (3.9), and (3.10). Components i^e_{ds} and i^e_{qs} of the stator current vector and slip frequency ω_r are the input variables and the torque T is the output variable.

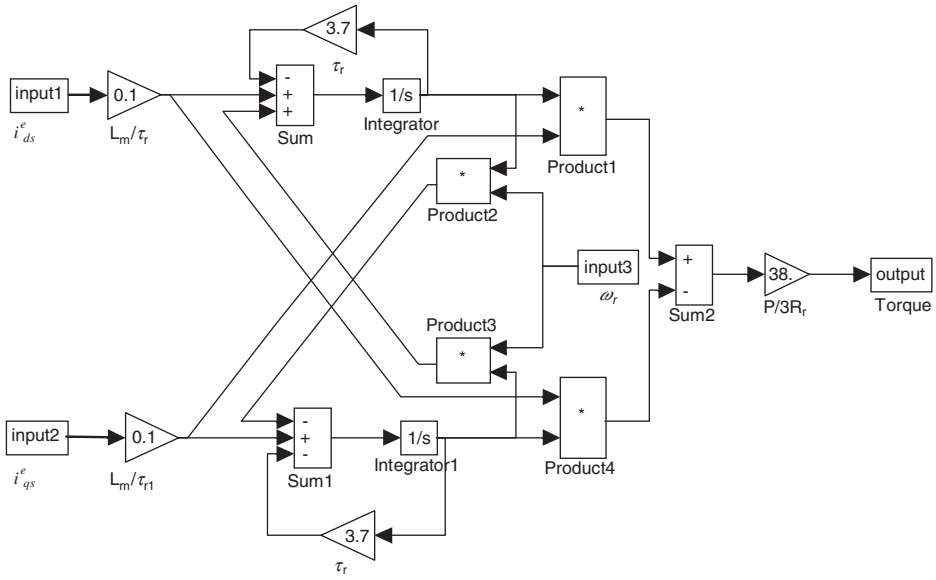


Figure 3.4 Electrical sub-model in Simulink. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

Construction of the electromechanical block in Simulink only requires the basic blocks of Sum, Gain, Product, and Integrator in the Simulink library. These blocks can be easily opened in separate windows for modification of parameters. The completed model is illustrated in Figure 3.4.

The electrical blocks are grouped into the electrical sub-model in the induction motor model.

3.3.3 Mechanical Sub-Model

When the friction coefficient of an induction motor is considered, the mechanical sub-model based on the first line of Equation (2.15) is expressed by Equations (3.11) and (3.12), which is constructed as shown in Figure 3.5.

$$\frac{d\omega_o}{dt} = -\frac{c_f}{J_m + J_L}\omega_o + \frac{T - T_L}{J_m + J_L} \tag{3.11}$$

$$\omega_o = \frac{(1/J_m + J_L)}{p + c_f/(J_m + J_L)}(T - T_L) \tag{3.12}$$

where p is a differential operator.

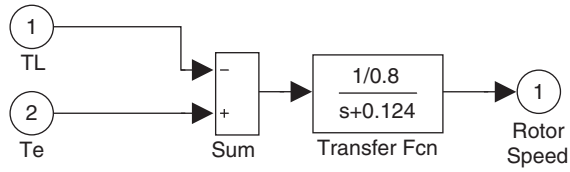


Figure 3.5 Simulink blocks of mechanical sub-model.

For the induction motor being studied, $J_m + J_L = 0.8$ and $c_f/(J_m + J_L) = 0.124$.

The slip speed ω_r can be calculated and fed back into the electromechanical block by Equation (3.13).

$$\omega_r = \omega - \frac{P}{2}\omega_o \tag{3.13}$$

The mechanical blocks are grouped into the mechanical sub-model in the induction motor model.

3.3.4 Simulation of Current-Input Model of Induction Motor

In order to test the current-input model, a power source block involving internal resistance and a calculation block of stator voltage are configured with the current-input model as shown in Figure 3.6.

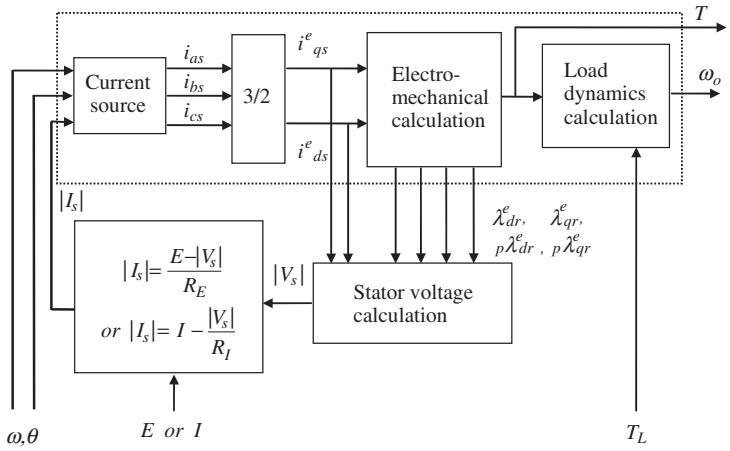


Figure 3.6 Block diagram of the current-input model.

In Figure 3.6, the block in dotted line is the current-input induction motor model which can be used alone when the motor operates in the current controlled mode. In addition, a power supply block is included to adapt the model for motor operation on a voltage source (E) or a current source (I).

In the excitation reference frame, stator flux equation and stator voltage equation can be expressed as (Trzynadlowski, 1994):

$$\lambda_s^e = \left(L_s - \frac{L_M^2}{L_r}\right) i_s^e + \frac{L_M}{L_r} \lambda_r^e \tag{3.14}$$

$$V_s^e = R_s i_s^e + (p + j\omega) \lambda_s^e \tag{3.15}$$

where $i_s^e = i_{ds}^e + j i_{qs}^e$, $V_s^e = V_{ds}^e + j V_{qs}^e$ and $\lambda_s^e = \lambda_{ds}^e + j \lambda_{qs}^e$.

The stator voltage calculation block is based on Equations (3.14) and (3.15). The flux $\lambda_r^e = \lambda_{dr}^e + j\lambda_{qr}^e$ and $p\lambda_r^e$ can be obtained directly from the outputs of the electromechanical block. When the two-phase quantities (voltages or currents) are transformed to three-phase coordinates, the peak value of the three phase quantities is 2/3 times of the two-phase quantities. Hence, the peak value of $|V_s|$ in three-phase coordinates in Figure 3.6 is $|V_s| = \frac{2}{3}\sqrt{(V_{ds}^e)^2 + (V_{qs}^e)^2}$.

The parameters of the induction motor used for simulation are listed under 'Motor 1' of Appendix B. To illustrate the application of the dynamic model of the induction motor to transient motor operation, a simulation study of direct-on-line starting is demonstrated. At the initial time instant ($t = 0$), the motor, previously de-energized and at standstill, is connected to a three-phase, 220 V (line-to-line voltage) and 60 Hz supply. The peak value of each phase voltage equals $\sqrt{2} \times \left(220/\sqrt{3}\right) = 179.63$ V. The power source is simulated by a signal generator block in the Simulink library. It is assumed that the load $T_L = 0.5\omega_o$ (N.m). The moment of inertia J_L of the load equals 0.4 kg m^2.

Figures 3.7–3.12 show the results of computer simulation using the Simulink model which also accounts for the effect of internal resistance in the power source. When the power supply has a large internal resistance, the torque oscillations in the torque/speed characteristic are reduced and damped more rapidly, but the run up time of the motor is longer.

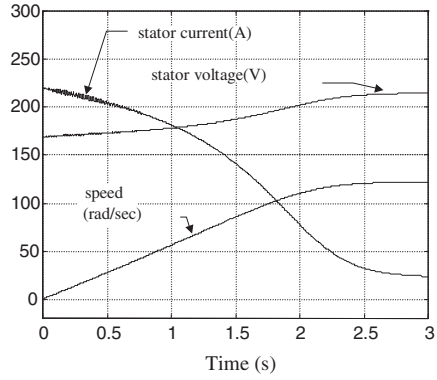


Figure 3.7 Speed response, stator current, and stator voltage when internal resistance of power supply is 0.2 Ω. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

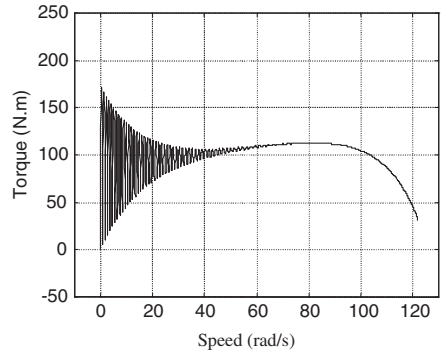


Figure 3.8 Dynamic torque-speed characteristic when internal resistance of power supply is 0.2 Ω. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

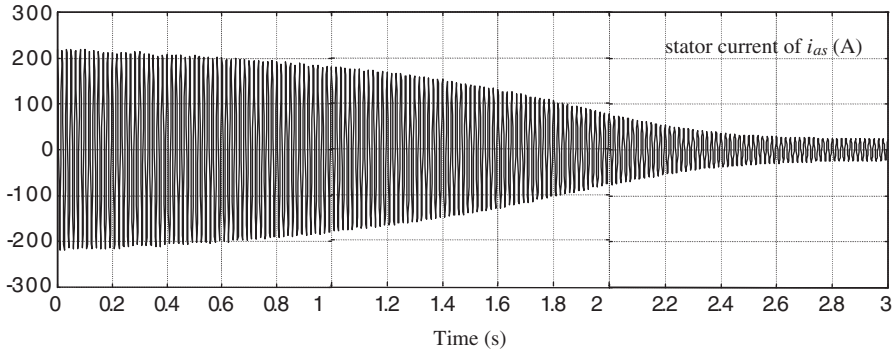


Figure 3.9 Stator current when internal resistance of power supply is 0.2 Ω. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

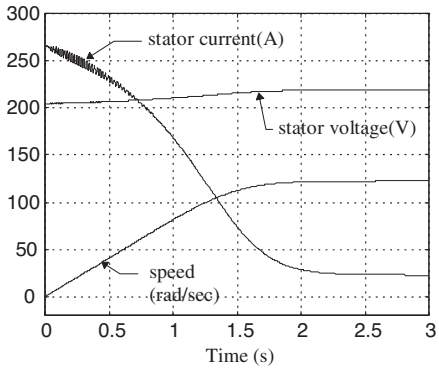


Figure 3.10 Speed response, stator current, and stator voltage when internal resistance of power supply is 0.05 Ω. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

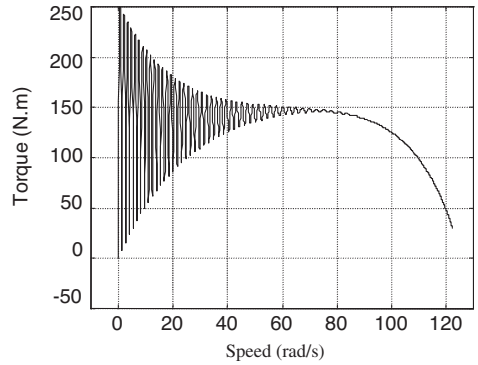


Figure 3.11 Dynamic torque-speed characteristic when internal resistance of power supply is 0.05 Ω. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK", *The 1997 IEEE Biennial International Electrical Machines and Drives Conference*, Paper WB3-6, 18–21 May 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

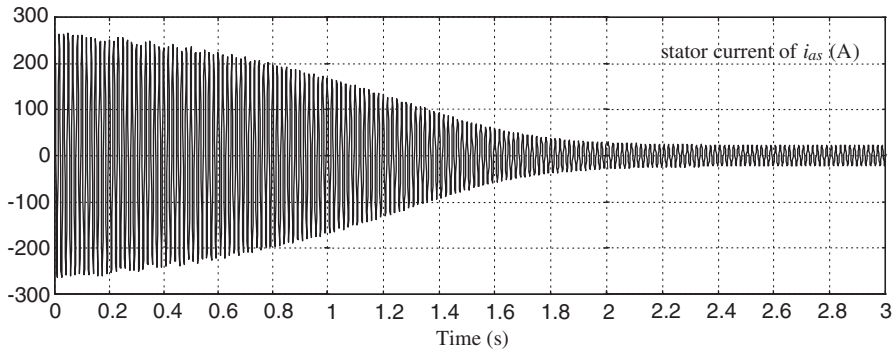


Figure 3.12 Phase current when internal resistance of power supply is 0.05 Ω. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Modelling of the three-phase induction motor using SIMULINK," The 1997 IEEE Biennial International Electrical Machines and Drives Conference, Paper WB3-6, May 18–21, 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE.)

3.4 Voltage-Input Model of Induction Motor

The voltage-input model is based on the 8th fifth-order equation of induction motor. In this modeling scheme, the induction motor consists of an electrical sub-model, a torque sub-model and a mechanical sub-model as shown in Figure 3.13. It is more convenient to split the induction motor model into smaller sub-models for drive design purpose.

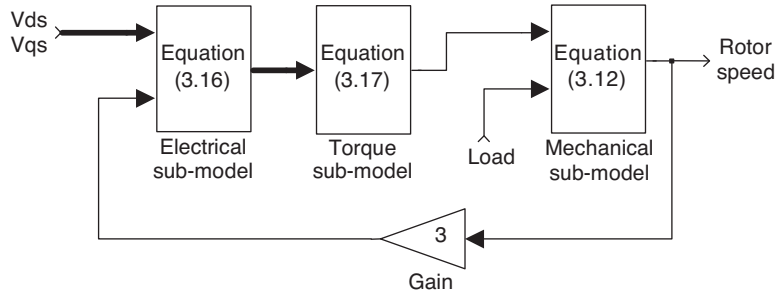


Figure 3.13 Modeling of an induction motor in Simulink (thick arrows representing vector inputs or outputs).

The equation for computing the current vector may be derived from Equation (3.1).

$$\begin{bmatrix} i_{ds}^s \\ i_{qs}^s \\ i_{dr}^s \\ i_{qr}^s \end{bmatrix} = \int_0^t \left\{ \begin{bmatrix} L_s & 0 & L_M & 0 \\ 0 & L_s & 0 & L_M \\ L_M & 0 & L_r & 0 \\ 0 & L_M & 0 & L_r \end{bmatrix}^{-1} \right.$$

$$\left. \left(\begin{bmatrix} V_{ds}^s \\ V_{qs}^s \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} R_s & 0 & 0 & 0 \\ 0 & R_s & 0 & 0 \\ 0 & \frac{P}{2}\omega_o L_M & R_r & \frac{P}{2}\omega_o L_r \\ -\frac{P}{2}\omega_o L_M & 0 & -\frac{P}{2}\omega_o L_r & R_r \end{bmatrix} \begin{bmatrix} i_{ds}^s \\ i_{qs}^s \\ i_{dr}^s \\ i_{qr}^s \end{bmatrix} \right) \right\} d\tau \qquad (3.16)$$

Matrix [A] and matrix [B] are defined as follows.

$$[A] = \begin{bmatrix} R_s & 0 & 0 & 0 \\ 0 & R_s & 0 & 0 \\ 0 & \frac{P}{2}\omega_o L_M & R_r & \frac{P}{2}\omega_o L_r \\ -\frac{P}{2}\omega_o L_M & 0 & -\frac{P}{2}\omega_o L_r & R_r \end{bmatrix} \quad [B] = \begin{bmatrix} L_s & 0 & L_M & 0 \\ 0 & L_s & 0 & L_M \\ L_M & 0 & L_r & 0 \\ 0 & L_M & 0 & L_r \end{bmatrix}^{-1}$$

By using the matrices [A] and [B], we can implement Equation (3.16) by the Simulink blocks as shown in Figure 3.14.

In Figure 3.14, the voltage vector [V_{ds}, V_{qs}] is the input vector and the current vector [i_{ds}, i_{qs}, i_{dr}, i_{qr}] is the output vector. Mux1, Mux2, Sum, Integrator, and Matrix Gain [B] are basic blocks

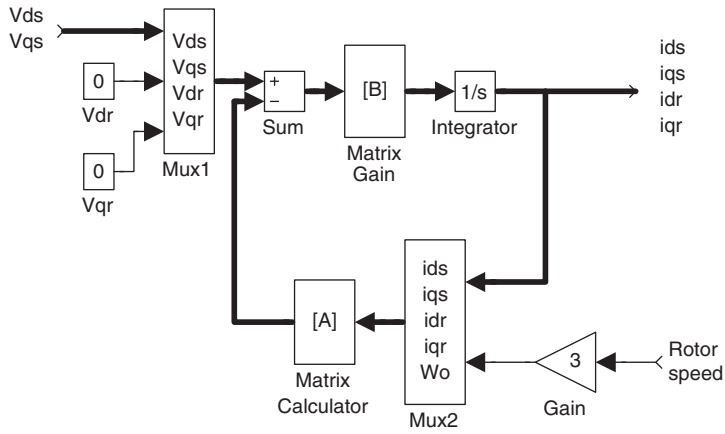


Figure 3.14 Electrical model of an induction motor in Simulink (thick arrows represent vector inputs or outputs).

of Simulink. The function blocks of electrical sub-model in Figure 3.14 may be grouped together and then included in the motor model in Figure 3.13.

1. Torque calculation model of induction motor
   The torque equation of an induction motor is expressed as:

$$T = \frac{PL_M}{3}(i_{dr}i_{qs} - i_{qr}i_{ds}). \tag{3.17}$$

   By using Equation (3.17), we can construct the torque calculation blocks as shown in Figure 3.15.

   In Figure 3.15, Demux, Product, Sum, and Gain are basic blocks of Simulink. The torque calculation blocks in Figure 3.15 may be grouped together and then included in the motor model in Figure 3.13.
2. Mechanical model of induction motor
   The mechanical model is same as that in the current-input model, as expressed in Equation (3.8) and illustrated in Figure 3.5.

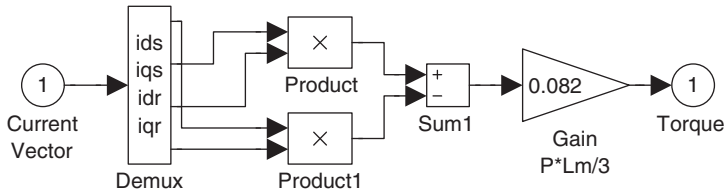


Figure 3.15 Torque calculation blocks in Simulink.

All function blocks of the induction motor in Figure 3.13 may be grouped together to form an induction motor block as shown in Figure 3.16. The two scope blocks enable the speed and torque of the motor to be observed.

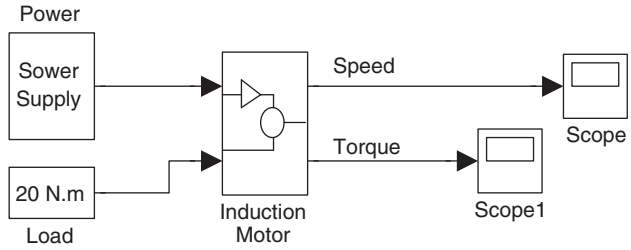


Figure 3.16 Simulink model of an induction motor.

3.4.1 Simulation Results of 'Motor 1'

The parameters of the induction motor used for simulation are listed under 'Motor 1' of Appendix B, but $c_f = 0$. To illustrate the transient operation of the induction motor, a simulation study of direct-on-line starting is demonstrated. At the initial time instant ($t = 0$), the motor, previously de-energized and at standstill, is connected to a three-phase, 220 V (line-to-line) and 60 Hz supply with an internal resistance of 0.05 Ω per phase. The power source is constructed using a signal generator block from the Simulink library. The load torque T_L is assumed to be 20 N m, and is independent of speed. The moment of inertia J_L of the load is 0.4 kg m^2. Figures 3.17 and 3.18 show the results of computer simulation using the Simulink model.

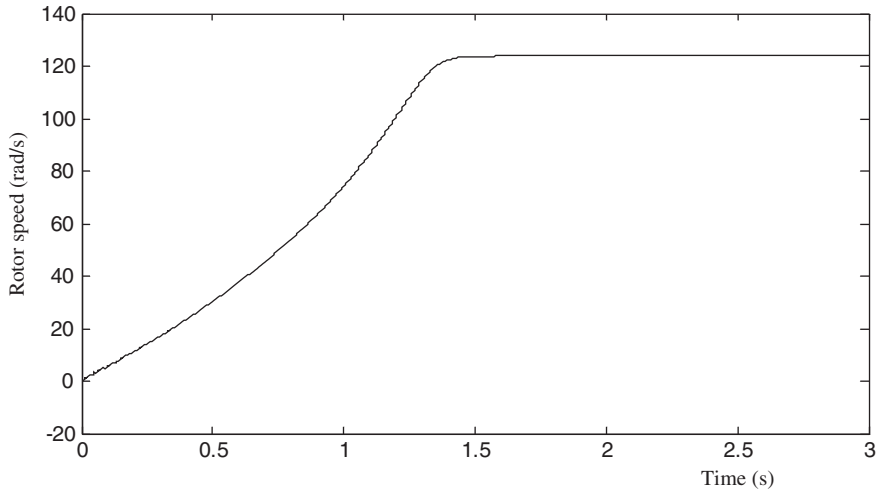


Figure 3.17 Speed response of the induction motor with direct-on-line starting.

3.4.2 Simulation Results of 'Motor 2'

The parameters of the induction motor are listed in 'Motor 2' of Appendix B. A simulation study of direct-on-line starting is carried out. At the initial instant of time, $t = 0$, the motor, previously de-energized and at standstill, is connected to a 220 V, 60 Hz supply. The power

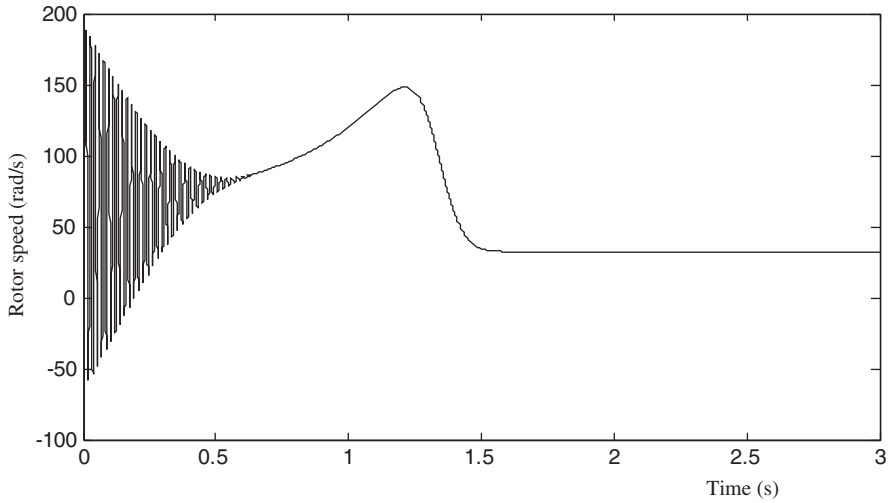


Figure 3.18 Torque response of the induction motor with direct-on-line starting.

source is constructed using a signal generator block from the Simulink library. The load torque T_L is assumed to be 2 N m, and independent of the speed. The moment of inertia J_L of the load is 0.05 kg m^2. Figures 3.19–3.21 show the results of computer simulation using the voltage-input model.

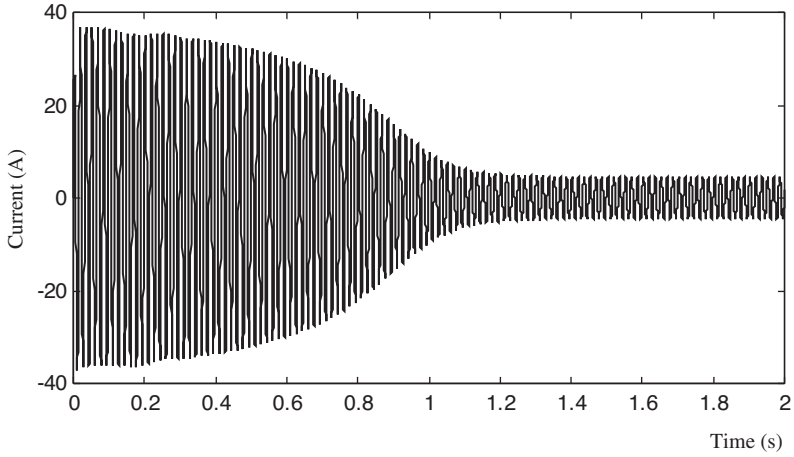


Figure 3.19 Stator current of the induction motor with direct-on-line starting.

3.4.3 Simulation Results of 'Motor 3'

'Motor 3' is an experimental motor (model 295) manufactured by Bodine Electric Company. The motor parameters listed in 'Motor 3' of Appendix B have been obtained from standard tests as outlined in Appendix H. To illustrate the transient operation of the induction motor,

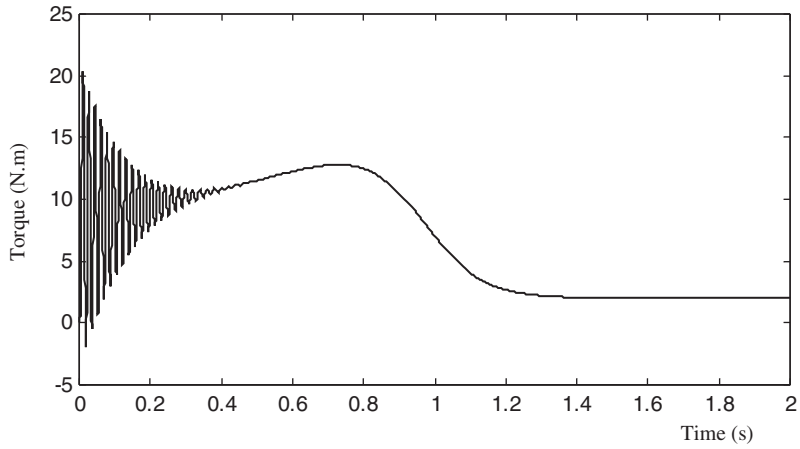


Figure 3.20 Torque response of the induction motor with direct-on-line starting.

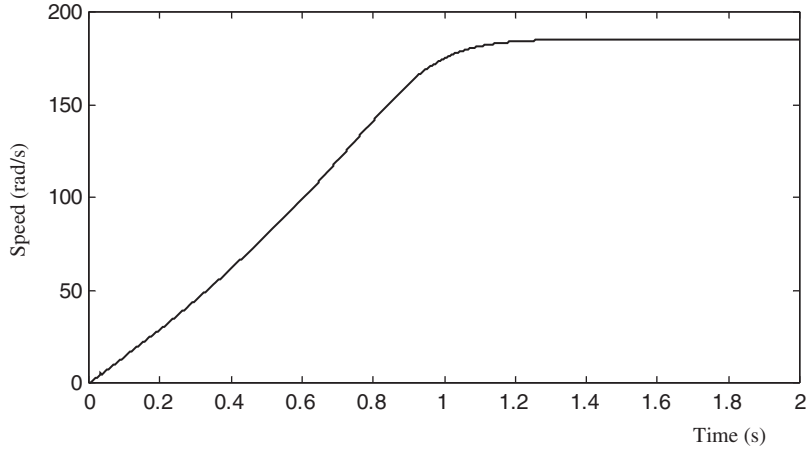


Figure 3.21 Speed response of the induction motor with direct-on-line starting.

a simulation study of direct-on-line starting is carried out. At the initial time instant, $t = 0$, the motor, previously de-energized and at standstill, is connected to a 220 V, 60 Hz supply. The power source is constructed using a signal generator block from the Simulink library. The load torque T_L is assumed to be 0.1 N m, and independent of speed. The moment of inertia J_L of the load is assumed to be 0.001 kg m^2. Figures 3.22–3.24 show the results of computer simulation over a period of 0.4 s using the voltage-input model.

3.5 Discrete-State Model of Induction Motor

Equation (2.14) expresses a state-space equation of induction motor. The solution to the differential equation $\dot{x} = Ax + Bu$ is given by (Lewis, 1992):

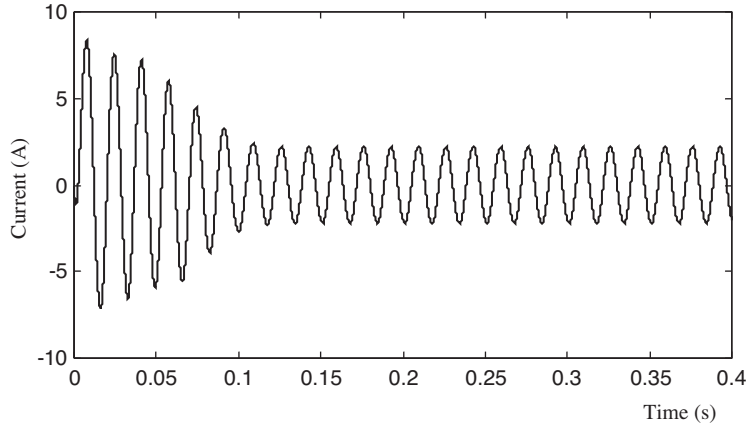


Figure 3.22 Phase-A stator current of the induction motor with direct-on-line starting.

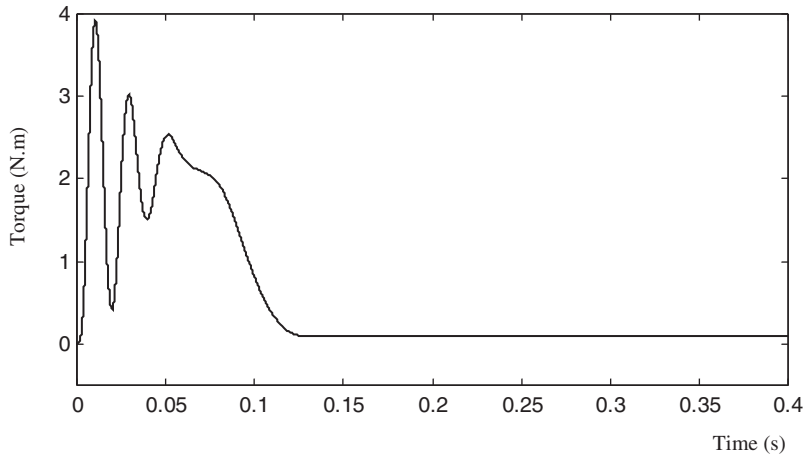


Figure 3.23 Torque response of the induction motor with direct-on-line starting.

$$x(t) = e^{A(t-t_0)}x(t_0) + \int_{t_0}^{t} e^{A(t-t_0)}Bu(\tau)d\tau. \tag{3.18}$$

In digital computation, the measurements are taken only at integral multiples of the sampling period M. Thus, we can convert the continuous-time model of an induction motor to the discrete-time model:

$$x_{n+1} = A_n x_n + B_n u_n \tag{3.19}$$

$$y_n = C_n x_n. \tag{3.20}$$

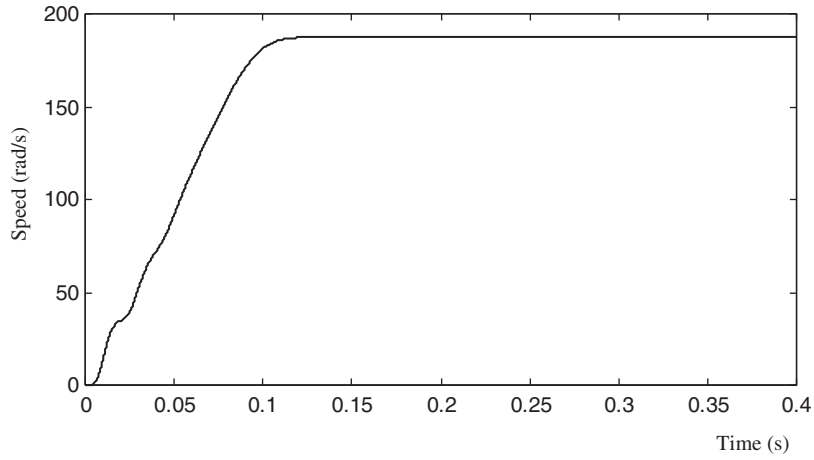


Figure 3.24 Speed response of the induction motor with direct-on-line starting.

Defining $t_0 = nM$, $t = (n + 1)M$, and the sampled sequence $x_n = x(nM)$, we may write

$$x_{n+1} = e^{AM} x_n + \int_{nM}^{(n+1)M} e^{A[(n+1)M-\tau]} Bu(\tau) d\tau. \tag{3.21}$$

Assuming that $u(t)$ is constant between the samples, the parameters in Equations (3.19) and (3.20) become

$$A_n = e^{AM}; \quad B_n = \int_0^M e^{AM} B d\tau; \quad C_n = C. \tag{3.22}$$

The conversion is accomplished by the following approximate formulas:

$$A_n = e^{AM} \approx I + AM \tag{3.23}$$

$$B_n = \int_0^M e^{AM} B d\tau \approx BM \tag{3.24}$$

$$C_n = C. \tag{3.25}$$

where we denote the system matrix, the input and output matrices of the continuous system with A, B, and C, and those of the discrete system with A_n, B_n, and C_n. We also assume that our sampling time is very short compared with the dynamics of the system. Making use of the rotor time constant $\tau_r = L_r/R_r$, $K_r = R_s + L_M^2 R_r / L_r^2$ and $K_l = (1 - L_M^2 / L_r / L_s) * L_s$, the discrete form of induction motor equation is derived from Equation (2.14) as follows:

$$\begin{bmatrix} i_{ds}^{(n+1)} \\ i_{qs}^{(n+1)} \\ \lambda_{dr}^{(n+1)} \\ \lambda_{qr}^{(n+1)} \\ \omega_o^{(n+1)} \end{bmatrix} = \begin{bmatrix} 1-\frac{K_r}{K_l}M & 0 & \frac{L_M R_r}{L_r^2 K_l}M & \frac{PL_M\omega_o^{(n)}}{2L_r K_l}M & 0 \\ 0 & 1-\frac{K_r}{K_l}M & \frac{PL_M\omega_o^{(n)}}{2L_r K_l}M & \frac{L_M R_r}{L_r^2 K_l}M & 0 \\ \frac{L_M}{\tau_r}M & 0 & 1-\frac{1}{\tau_r}M & -\frac{P}{2}\omega_o^{(n)}M & 0 \\ 0 & \frac{L_M}{\tau_r}M & \frac{P}{2}\omega_o^{(n)}M & 1-\frac{1}{\tau_r}M & 0 \\ -\frac{PL_M}{3JL_r}\lambda_{qr}^{(n)}M & \frac{PL_M}{3JL_r}\lambda_{dr}^{(n)}M & 0 & 0 & 1-\frac{MC_f}{J} \end{bmatrix} \begin{bmatrix} i_{ds}^{(n)} \\ i_{qs}^{(n)} \\ \lambda_{dr}^{(n)} \\ \lambda_{qr}^{(n)} \\ \omega_o^{(n)} \end{bmatrix}$$

$$+ \begin{bmatrix} \frac{M}{K_L} & 0 & 0 \\ 0 & \frac{M}{K_L} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\frac{M}{J} \end{bmatrix} \begin{bmatrix} V_{ds}^{(n)} \\ V_{qs}^{(n)} \\ T_L^{(n)} \end{bmatrix} \tag{3.26}$$

Equation (3.26) can be implemented by an 'S-function' block of MATLAB®/Simulink. The S-function is a program description of a dynamic system. With the power source block, the load block, and the scope block, the S-function block of induction motor is configured in Simulink as shown in Figure 3.25.

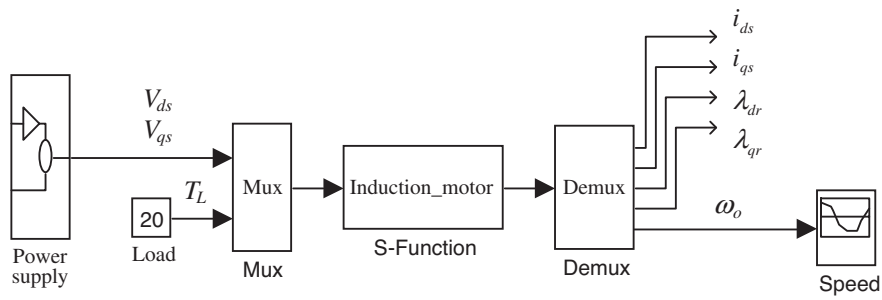


Figure 3.25 Discrete-state model of induction motor in Simulink.

In the discrete-state model of induction motor, the discrete state is $[i_{ds}^s\ i_{qs}^s\ \lambda_{dr}^s\ \lambda_{qr}^s \omega_o]$, the input is $[V_{ds}^s\ V_{qs}^s\ T_L]$, and the output is $[i_{ds}^s\ i_{qs}^s\ \lambda_{dr}^s\ \lambda_{qr}^s\ \omega_o]$. The power source which produces the stator voltage vectors is constructed using a signal generator block in the Simulink library. The 'S-function' block accommodates a short program called M-file which implements Equation (3.26). The M-file is given in Appendix C.

To illustrate the transient operation of the induction motor, a simulation of direct-on-line starting is done. The induction motor parameters for the computer simulation are listed under 'Motor 1' of Appendix B. The load torque, T_L, is assumed to be 20 N m, and independent of speed. The moment of inertia J_L of the load equals that of the motor. At the initial time instant ($t = 0$), the motor, previously de-energized and at standstill, is connected to a three-phase, 220 V (line-to-line), 60 Hz supply with an internal resistance of 0.05 Ω per phase.

Figure 3.26 shows the results of computer simulation using the discrete-state model with sampling time $M = 0.0005$s. The speed-time curve is similar to that obtained using the voltage-input model (Figure 3.17).

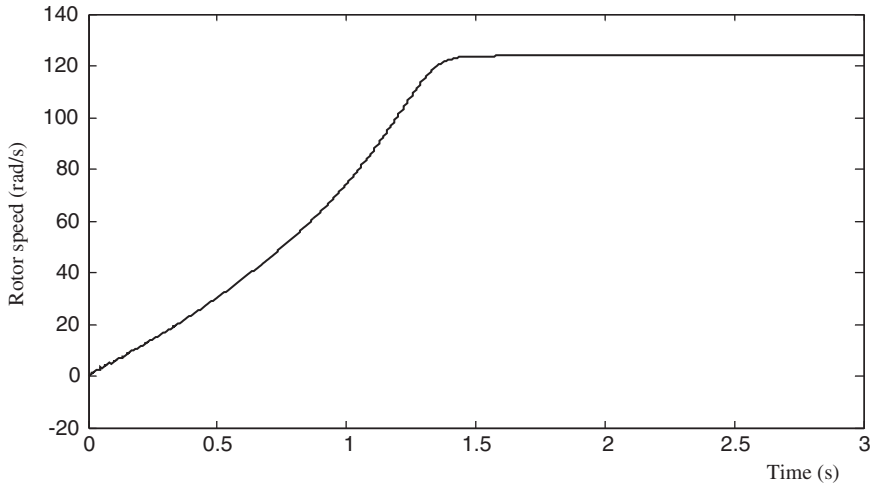


Figure 3.26 Speed response of the induction motor with direct-on-line staring.

3.6 Modeling and Simulation of Sinusoidal PWM

Adjustable frequency operation of motors requires a symmetrical set of three-phase sine modulated PWM (pulse-width-modulated) waveforms, adjustable in both amplitude and frequency. The reference voltages (which are sinusoidal waves) should be adjustable in the full speed range, normally in the area from several Hz to several hundred Hz. The PWM patterns were typically generated by comparing these analog reference voltages with a high-frequency triangular carrier wave at the desired switching frequency (typically between 5 and 20 kHz). The results of the comparisons between the sinusoidal references and the carrier waveform are the PWM signals used to control the power devices of the voltage source inverter that supplies the motor. The desired pulse widths are usually calculated in a microprocessor or DSP chip and then a PWM generation unit is used to produce the output patterns. In order to supply the motor, a power converter is needed that translates the low level PWM signals from the processor to the appropriate high voltage levels. The most common of such a power converter is the voltage source inverter that comprises six power switching devices such as MOSFETs or IGBTs.

To study the sinusoidal PWM for induction motor control, a Simulink program module of the sinusoidal PWM shown in Figure 3.27 is used.

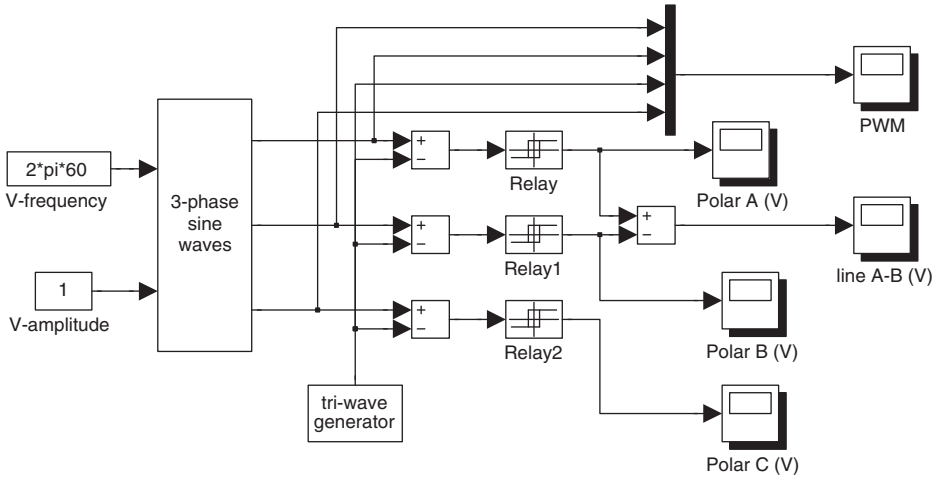


Figure 3.27 Simulation program of the sinusoidal PWM in Simulink.

In the simulation program, two 'Constant' blocks are used to produce the frequency and the amplitude of the reference voltage. The block '3-phase sine waves' yields three-phase sinusoidal signals according to the frequency and amplitude of the reference voltage, while the block 'tri-wave generator' yields a high-frequency triangular carrier wave. By using three 'Relay' blocks (thresholds of switch on and switch off in the 'Relay' blocks are set as 10^{-4} and -10^{-4}), we can output the three-phase sinusoidal PWM waveforms. The five 'Scope' blocks are used for viewing the simulation results.

The reference voltages and the triangular wave carrier of the three-phase sinusoidal PWM are shown in Figure 3.28, which can be observed from the 'PWM' scope shown in Figure 3.27.

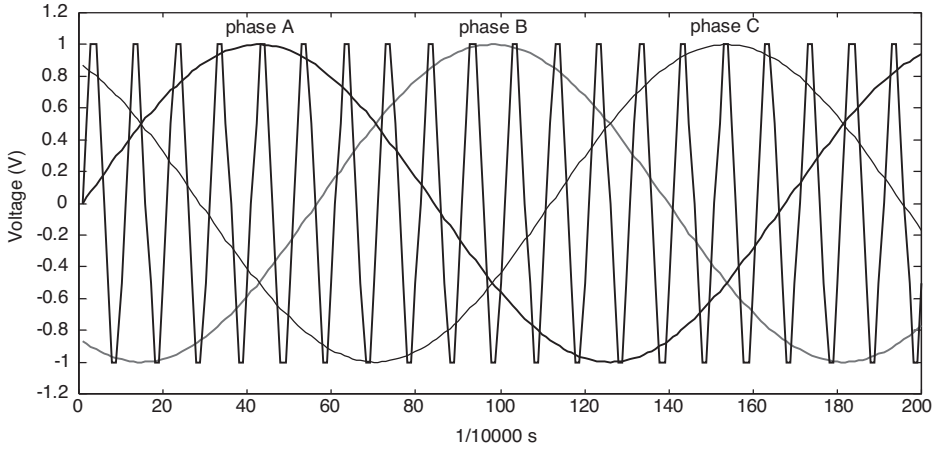


Figure 3.28 Reference voltages and triangular wave carrier.

Pole-A and pole-B voltages of the three-phase sinusoidal PWM are shown in Figures 3.29 and 3.30 and these can be observed from the 'Polar A (V)' and 'Polar B (V)' scopes shown in Figure 3.27.

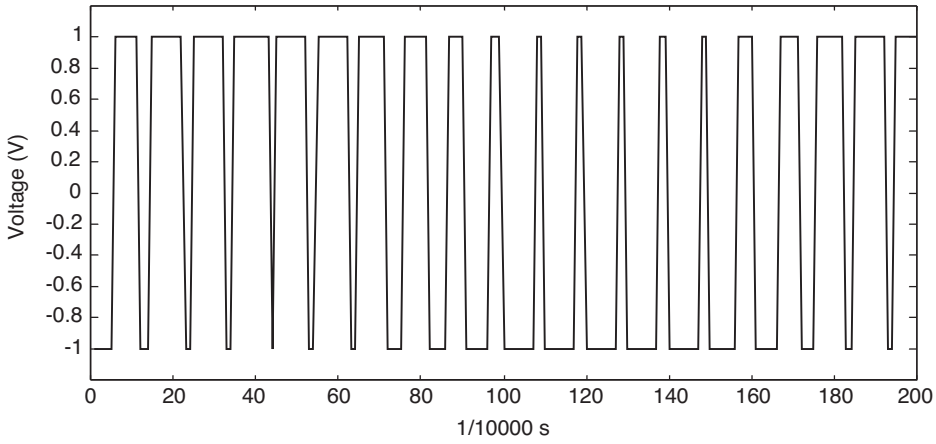


Figure 3.29 Pole-A voltage of three-phase sinusoidal PWM.

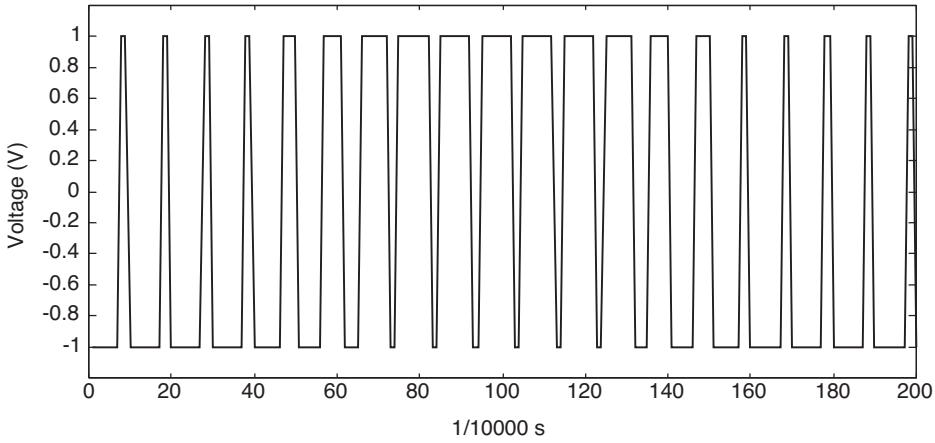


Figure 3.30 Pole-B voltage of three-phase sinusoidal PWM.

The line voltage of the three-phase sinusoidal PWM is shown in Figure 3.31, which can be observed by the 'line A-B (V)' scope in Figure 3.27.

3.7 Modeling and Simulation of Encoder

The encoder for measuring rotor speed is an optoelectronic feedback device that uses a patterned optical mask and an LED light source and transistor photo-sensor pairs. As the induction motor shaft rotates, the light source either passes through the disk, or is blocked by

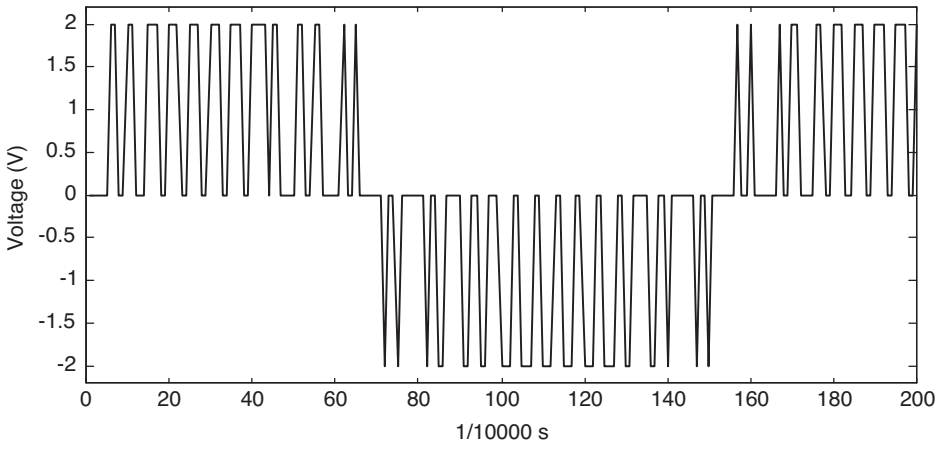


Figure 3.31 Line A-B voltage of three-phase sinusoidal PWM.

the disk. The emitter/detector pairs produce a digital output waveform. The encoder can be modeled by Simulink as shown in Figure 3.32.

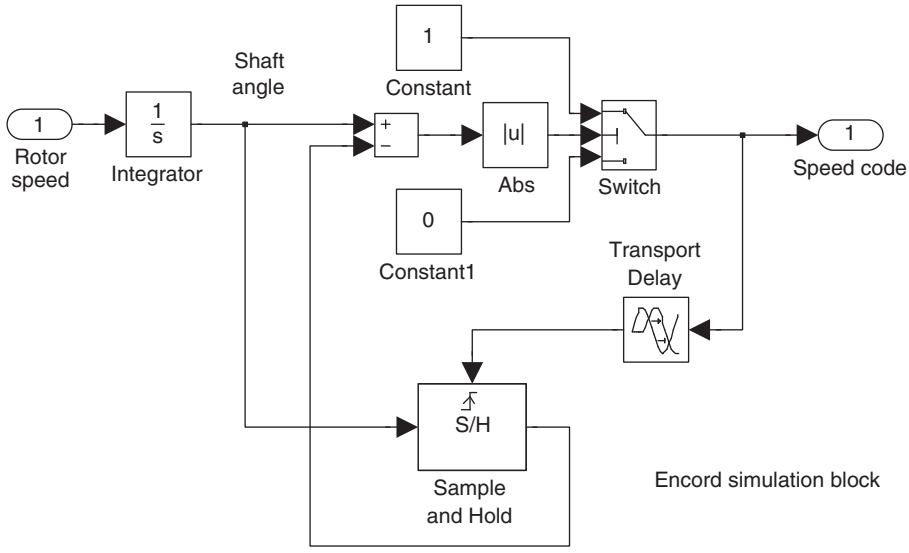


Figure 3.32 Simulation model of speed encoder in Simulink.

In the encoder model, the input is rotor speed of the induction motor and the output is a pulse string of the angular speed code. The rotor speed is transformed to a shaft angle by using an 'Integrator' block of Simulink, while a 'Transport Delay' block is used to control the pulse width of the encoder output. Two 'Constant' blocks are used to control the magnitude of the encoder output pulse. The 'Switch' block is used to control the encoder precision. If the encoder

precision is defined as N pulses/revolution, the threshold of the 'Switch' is set as $2^{*}\pi/N$. Figure 3.33 shows the rotor speed of the induction motor and Figure 3.34 shows the corresponding pulse string of the rotor speed code when the encoder precision is set to be 200 pulses/revolution.

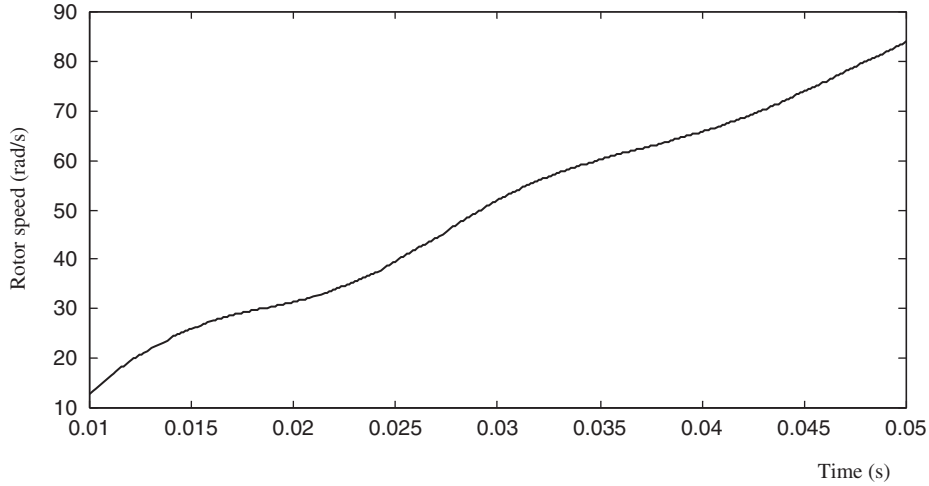


Figure 3.33 Rotor speed of induction motor input to the encoder model.

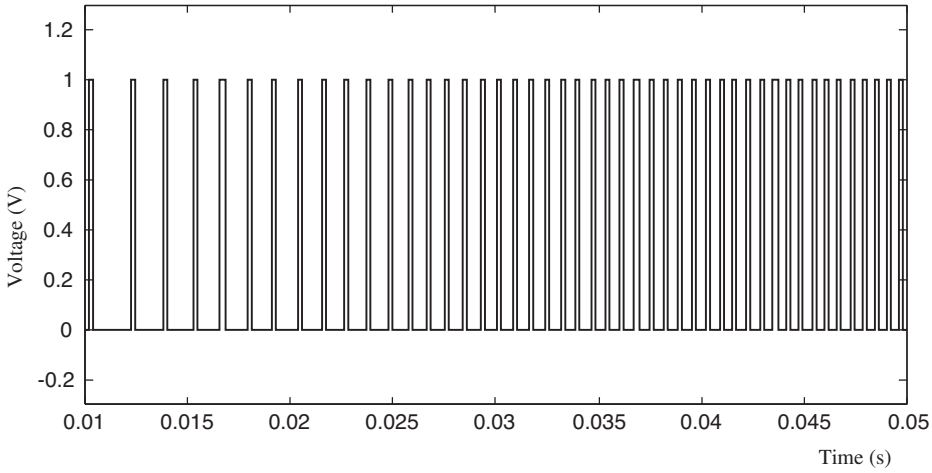


Figure 3.34 Pulse string output from the encoder model with a precision of 200 pulses/revolution.

3.8 Modeling of Decoder

The function of the decoder is to transform the output pulses of an encoder into the rotor speed signal. It can be implemented by special decoder hardware or DSP processor in a practical application. A mathematical model of the decoder may be expressed by Equations (3.27) and (3.28):

$$\theta_o(t) = f(S_{code}) \tag{3.27}$$

$$\omega_o = \frac{\theta_o(t) - \theta_o(t - \Delta t)}{\Delta t}. \tag{3.28}$$

where S_{code} is output pulse of an encoder, θ_o is the shaft angle.

A Simulink model of the decoder is shown in Figure 3.35.

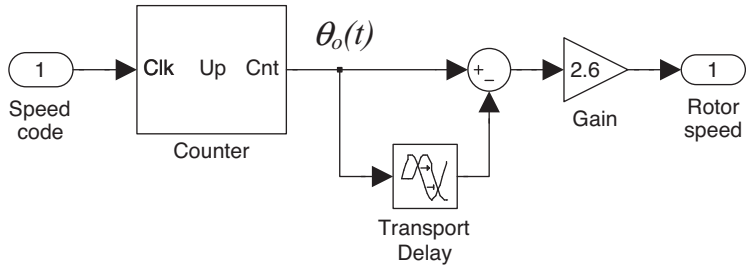


Figure 3.35 Decoder model in Simulink window.

In this decoder model, a 'Counter' block implements Equation (3.27). A 'Transport Delay' block, a 'Sum' block, and a 'Gain' block implement the difference calculation according to Equation (3.28). The value of the 'Gain' block equals $2^*\pi^*\Delta t/N$, where N is the encoder precision in pulses/revolution.

3.9 Simulation of Induction Motor with PWM Inverter and Encoder/Decoder

To simulate the induction motor with a PWM inverter power supply and an encoder device, the PWM model built in Section 3.6, the encoder model built in Section 3.7, and the decoder model built in Section 3.8 are configured with the voltage-input model of induction motor in Section 3.4. A 'Gain' block is used to simulate an inverter whose input is the PWM model's output and whose output is the polar voltages for an induction motor.

The parameters of an experimental induction motor used for the simulation are listed under 'Motor 3' of Appendix B. The load torque T_L is assumed to be 0.1 N m, and independent of speed. The moment of inertia J_L of the load is assumed to be 0.001 kg m^2, and the PWM switch voltage is set as $\pm$ 180 V. Figure 3.36 shows the pole-A output voltage of the inverter and Figure 3.37 shows the line voltage V_{AB} supplied to the induction motor. Figure 3.38 shows the phase-A stator current of the induction motor and Figure 3.39 shows the torque response of the induction motor.

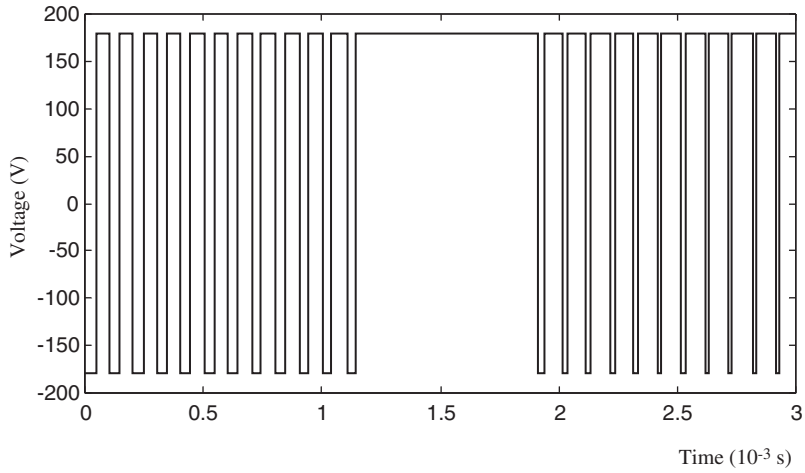


Figure 3.36 Pole-A voltage of the inverter output to the induction motor.

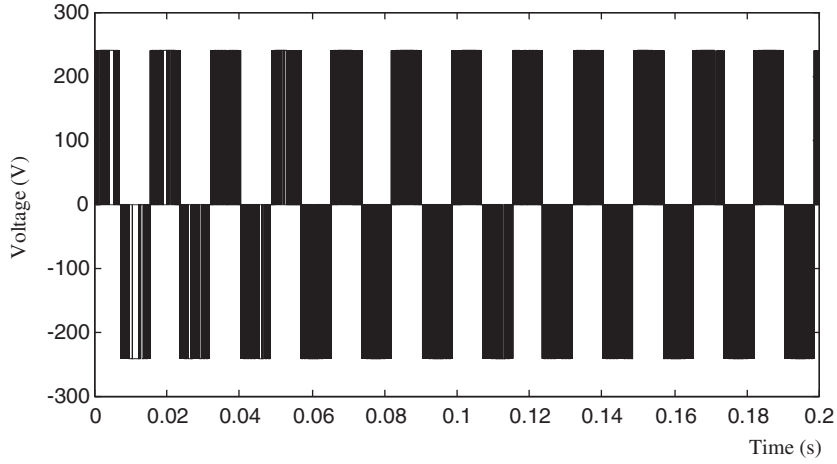


Figure 3.37 Line voltage V_{AB} supplied to the induction motor.

Figure 3.40 shows the rotor speed measured by the encoder and decoder models built in Sections 3.7 and 3.8 respectively.

Figure 3.41 shows the rotor speed measured by the encoder and decoder models, when the encoder precision is set as 2000 pulses/revolution.

3.10 MATLAB®/Simulink Programming Examples

MATLAB® (The MathWorks, Inc., 2008a) is a popular computing software. Using easy-to-understand program language, MATLAB® can manipulate matrices, functions, algebraic

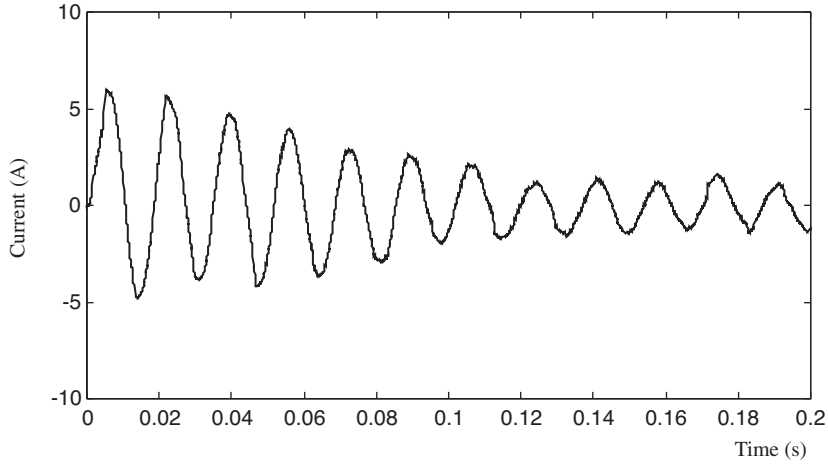


Figure 3.38 Phase-A stator current of the induction motor.

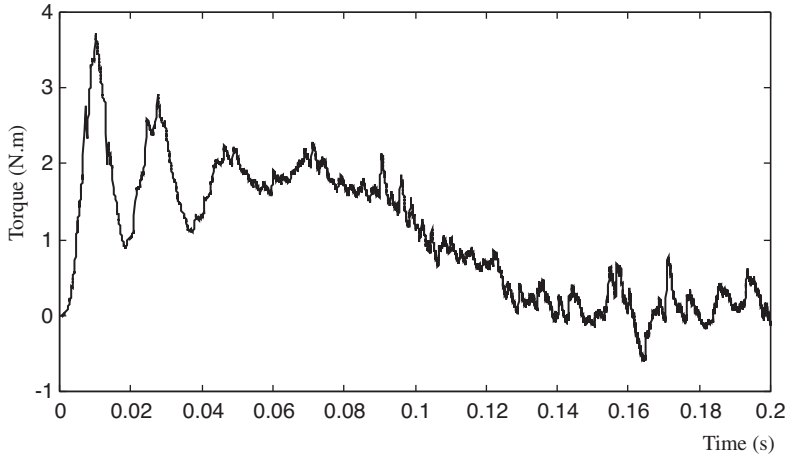


Figure 3.39 Torque response of the induction motor with PWM inverter supply.

expressions, and it is provided with facilities for plotting of computed data. Simulink (The MathWorks, Inc., 2008b) is a graphical programming tool that must run in the MATLAB® environment. Simulink (The MathWorks, Inc., 2008c) can be used to model, simulate and analyze dynamic systems by using graphical blocks. It should be mentioned that the functions supported by Simulink is only a subset of those supported by MATLAB®. Some complex algorithms, such as Genetic Algorithm or algebra operation, can only be performed by MATLAB® codes. Nevertheless, graphical programming proves very helpful for a nonprofessional programmer. Basically, there are three ways to simulate a dynamic system on

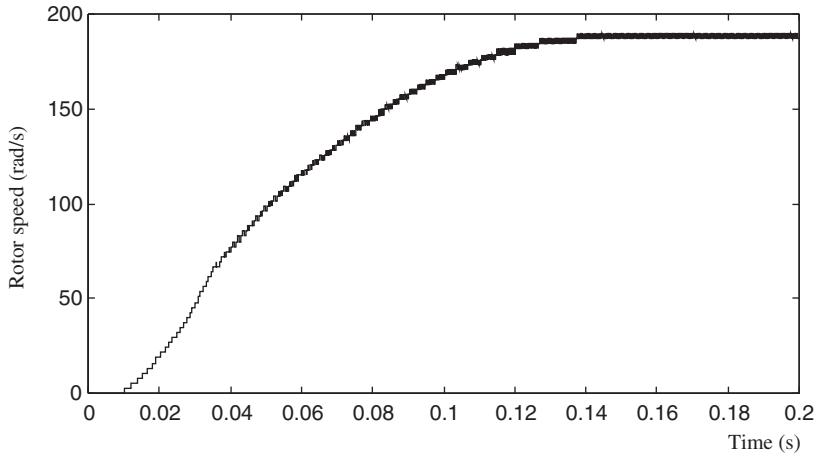


Figure 3.40 Rotor speed measured by an encoder with a precision of 200 pulses/revolution.

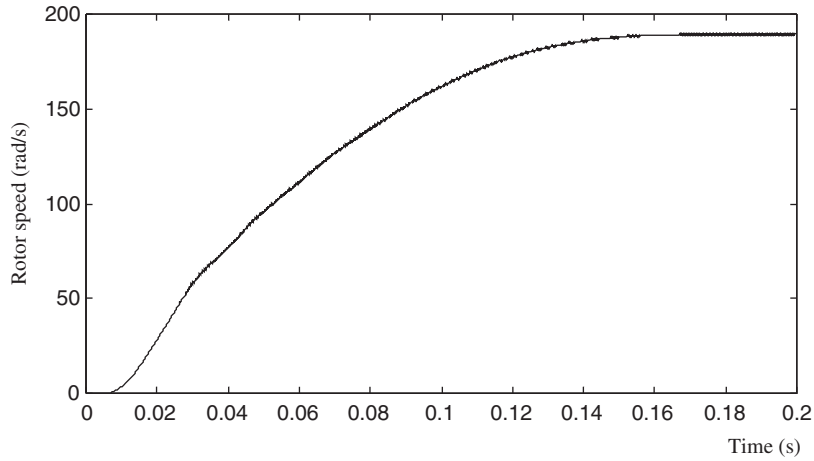


Figure 3.41 Rotor speed of the induction motor with an encoder of 2000 pulses/revolution.

MATLAB®/Simulink. The first method is to write program codes of MATLAB® language and to perform the codes in the MATLAB® command window. The second method is to create a simulation model with various blocks from the Simulink library and to execute the simulation of the model in Simulink. The third method is simulation using hybrid MATLAB® codes and Simulink blocks.

When MATLAB® is started, a main window will appear as shown in Figure 3.42.

In the MATLAB® window, various calculation and simulation may be performed by typing corresponding commands into the command line. The commands may be an equation, a function, or a sequence of codes.

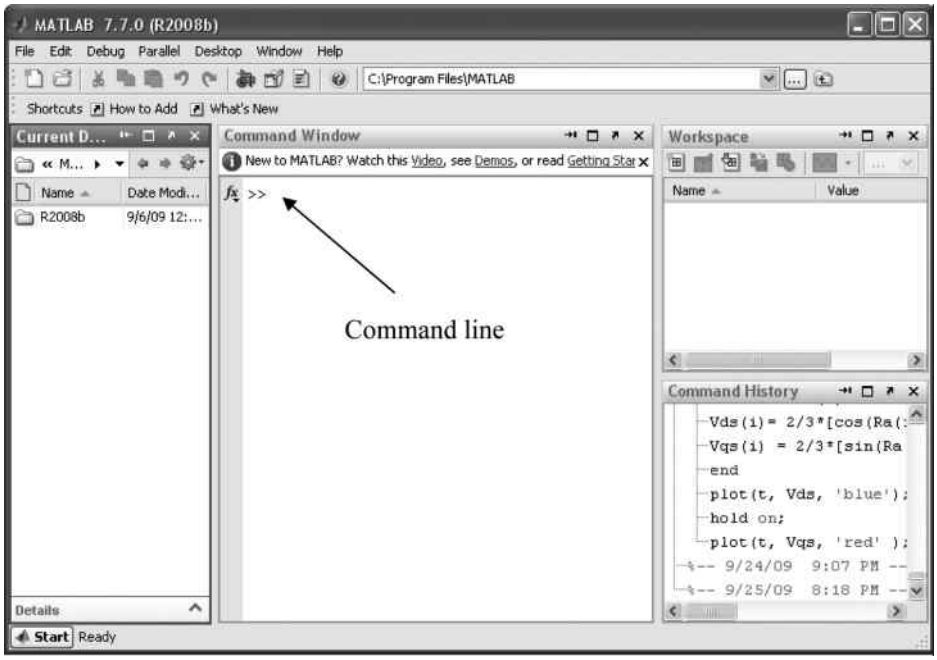


Figure 3.42 MATLAB® window.

In this section, six examples will be given to illustrate programming of Park's transform with MATLAB® codes, Simulink blocks, and hybrid methods. The six different programming examples are useful for mastering the basic steps of MATLAB®/Simulink programming.

Example 3.1 Programming by MATLAB® commands
Example 3.2 Programming by Simulink blocks
Example 3.3 Programming by Simulink blocks with self-defined expression
Example 3.4 Programming by Simulink blocks which calls a MATLAB® function
Example 3.5 Programming by MATLAB® function which calls a Simulink model
Example 3.6 Programming by calling an 'abc_to_qd0' block

Example 3.1 uses only MATLAB® commands and Example 3.2 uses only Simulink blocks. Example 3.3 illustrates using self-defined expression to simplify a Simulink model. The Examples (3.4–3.6) illustrate how to program with hybrid MATLAB® function and Simulink blocks. The hybrid programming method is useful in genetic algorithm and extended Kalman filter simulations, as these are functions of MATLAB® in the present MATLAB®/Simulink version. When a Simulink model is optimized by the Genetic Algorithm, it needs to call the Simulink model from MATLAB® function. When an extended Kalman filter of MATLAB® function is embedded in a Simulink model, the Simulink model needs to call the MATLAB® function.

Example 3.1 Programming of Park's Transformation by MATLAB® Commands

The original Park's transformation of voltage is expressed as

$$\begin{bmatrix} V_{ds} \\ V_{qs} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ \sin(\theta) & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \begin{bmatrix} V_{as} \\ V_{bs} \\ V_{cs} \end{bmatrix} \tag{3.29}$$

In Park's transformation, the inputs are the vector [*Vas, Vbs, Vcs*] and the reference angle θ, while the output vector is [*Vds, Vqs*]. To simulate Park's transformation, the input data needs to be created first. Then the transformation function needs to be expressed by MATLAB codes. Finally, the results of the transformation need to be plotted.

Step 1 Create Data of Three Phase Voltages

The three phase voltages [V_{as}, V_{bs}, V_{cs}] with a frequency of 60 Hz and amplitude of 1 V may be expressed as follows:

$$V_{as} = \sin(2\pi \times 60t) \tag{3.30}$$

$$V_{bs} = \sin\left(2\pi \times 60t - \frac{2}{3}\pi\right) \tag{3.31}$$

$$V_{cs} = \sin\left(2\pi \times 60t + \frac{2}{3}\pi\right). \tag{3.32}$$

In the example, simulation time is set as 0.03 s and step size is set as 0.0001s. Type the following code in the MATLAB® command line to create the time variable, '*t*'.

```
>> t = [0:0.0001:0.03];
```

Type the following code to create data of the three phase voltages.

```
>> Vas = sin(2*pi*60*t);
>> Vbs = sin(2*pi*60*t - 2/3*pi);
>> Vcs = sin(2*pi*60*t + 2/3*pi);
```

Type the following code for plotting the waveforms of the three phase voltages.

```
>> plot(t, Vas, 'blue');
>> hold on;
>> plot(t, Vbs, 'red');
>> plot(t, Vcs, 'black');
>> hold off;
```

The 'plot' command is used to show the waveforms of the three phase voltages. *Vas* is plotted in blue, *Vbs* is plotted in red, and *Vcs* is plotted in black. Use the 'hold on' and 'hold off' commands to keep the waveforms of the three phase voltages in one picture frame.

The above codes may be combined as follows:

```
clear
t = [0:0.0001:0.03];
Vas = sin(2*pi*60*t);
Vbs = sin(2*pi*60*t - 2/3*pi);
Vcs = sin(2*pi*60*t + 2/3*pi);
plot(t, Vas, 'blue');
hold on;
plot(t, Vbs, 'red');
plot(t, Vcs, 'black');
axis([0 0.03 -1.2 1.2]);
hold off;
```

The entire code sequence may be copied into the MATLAB® command line. The results of executing the code are shown in Figure 3.43.

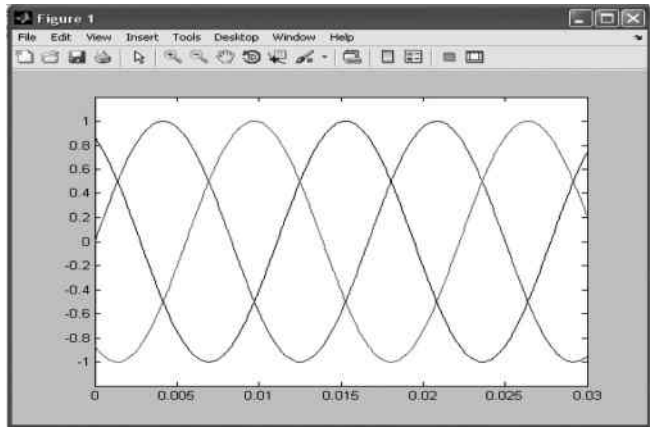


Figure 3.43 Waveforms of three phase voltages V_{as}, V_{bs}, and V_{cs}.

Step 2 Create Reference Angle of Transformation

The reference angle of transformation is represented by a 100 Hz sawtooth wave with amplitude equal to 2π rad. Type the following code in the MATLAB® command line to create data of the reference angle which is saved in variable 'Ra'. The time variable '*t*' in the codes has been created in **Step 1**.

```
>> Ra = pi + pi*sawtooth(2*pi*100*t);
```

The reference angle created may be plotted by following commands.

```
>>plot(t, Ra);
>>axis([0 0.03 0 2*pi]);
```

The command 'axis' is used to set the x-axis and y-axis limits. In the example, the x-axis is time 't' with range from 0 to 0.03 s and the y-axis is angle with range from 0 to 2π rad. The waveform of the reference angle is shown in Figure 3.44.

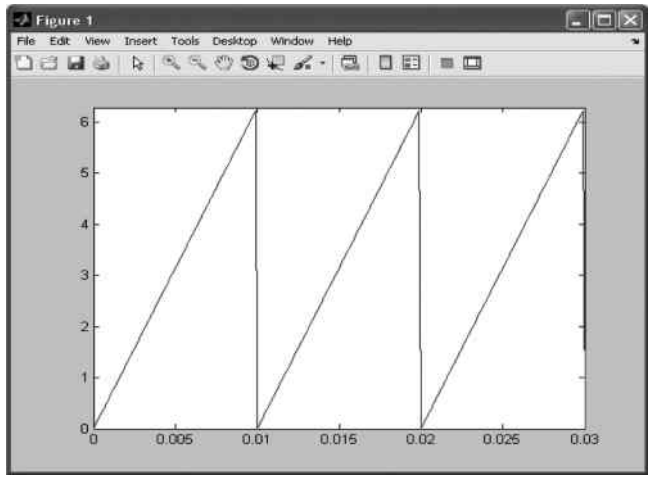


Figure 3.44 Waveform of the reference angle.

Step 3 MATLAB® Code of Park's Transformation

In **Step 1**, the simulation time is set as 0–0.03 s with a step size of 0.0001s, hence, the data size of the simulation is 301 (including one data at $t = 0$ s).

Park's transformation is implemented by the following code, the results being saved in the variables *Vds* and *Vqs*.

```
>>for i = 1:301;
>>Vds(i) = 2/3*[cos(Ra(i))', cos(Ra(i)-2/3*pi)',
cos(Ra(i) + 2/3*pi)']*[Vas(i);Vbs(i);Vcs(i)];
>>Vqs(i) = 2/3*[sin(Ra(i))', sin(Ra(i)-2/3*pi)',
sin(Ra(i) + 2/3*pi)']*[Vas(i);Vbs(i);Vcs(i)];
>>end
```

It is easily seen that the Park's transformation given by Equation (3.29) is implemented by the looping statements. The results may be plotted by following commands.

```
>>plot(t, Vds, 'blue');
>>hold on;
>>plot(t, Vqs, 'red');
>>hold off;
```

Step 4 Create a New MATLAB® Function of Park's Transformation

A new MATLAB® file may be created by clicking 'File' and selecting 'New' in MATLAB® window. Copy all the codes in Steps 1∼3 into the new file and save it with the name 'Example1_Park_transform' with suffix '.m' (the extension 'm' means MATLAB® file), that is, 'Example1_Park_transform.m'. The file may be executed by typing the filename

'Example1_Park_transform' in the MATLAB® command line.
The file 'Example1_Park_transform.m' is listed as follows:

```
% Clear all variables in workspace
clear

% Step 1
t = [0:0.0001:0.03];
Vas = sin(2*pi*60*t);
Vbs = sin(2*pi*60*t - 2/3*pi);
Vcs = sin(2*pi*60*t + 2/3*pi);
subplot(3,1,1);
plot(t, Vas, 'blue');
hold on;
plot(t, Vbs, 'red');
plot(t, Vcs, 'black');
axis([0 0.03 -1.2 1.2]);
hold off;

% Step 2
Ra = pi + pi*sawtooth(2*pi*100*t);
subplot(3,1,2);
plot(t, Ra);
axis([0 0.03 0 2*pi]);

% Step 3
for i = 1:301;
Vds(i)= 2/3*[cos(Ra(i))', cos(Ra(i)-2/3*pi)',
cos(Ra(i) + 2/3*pi)']*[Vas(i);Vbs(i);Vcs(i)];
Vqs(i) = 2/3*[sin(Ra(i))', sin(Ra(i)-2/3*pi)',
sin(Ra(i) + 2/3*pi)']*[Vas(i);Vbs(i);Vcs(i)];
end
subplot(3,1,3);
plot(t, Vds, 'blue');
hold on;
plot(t, Vqs, 'red');
axis([0 0.03 -1.2 1.2]);
hold off;
```

Note:

1. '%' is comment line.
2. Command 'subplot' is used to place multiple plots in same figure frame. Command 'subplot (n, m, k)' place $m \times n$ pictures in a figure frame where m is size of rows, n is size of columns and k is the index of current plot.

The results are shown in Figure 3.45.

Example 3.2 Programming of Park's Transformation by Simulink Blocks

Step 1 Creating a New Simulink Model
From the MATLAB® window, Simulink may be called by typing 'Simulink' in the MATLAB® command line or by clicking the 'Simulink' icon on the upper left corner of the MATLAB® window. The 'Simulink Library Browser' window shown in Figure 3.46 will appear.

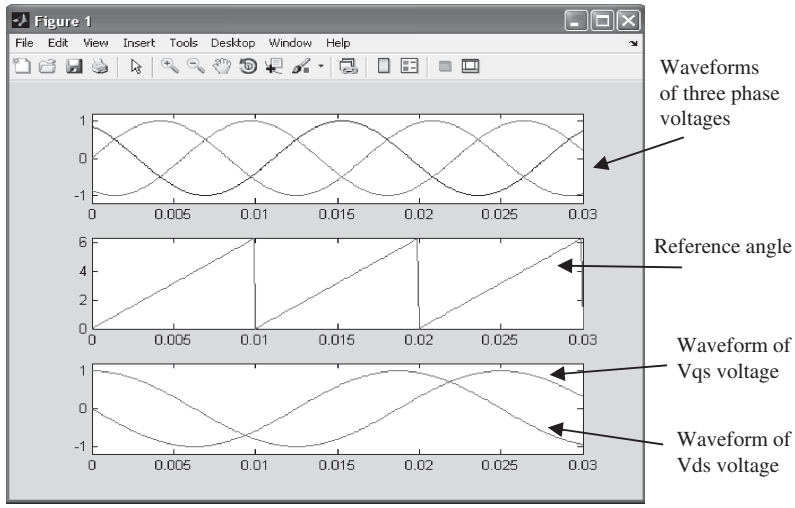


Figure 3.45 Simulation results of Example 3.1.

Click the icon as indicated in Figure 3.46 (or select 'New' and 'Module' from 'File') to create a blank model as shown in Figure 3.47.

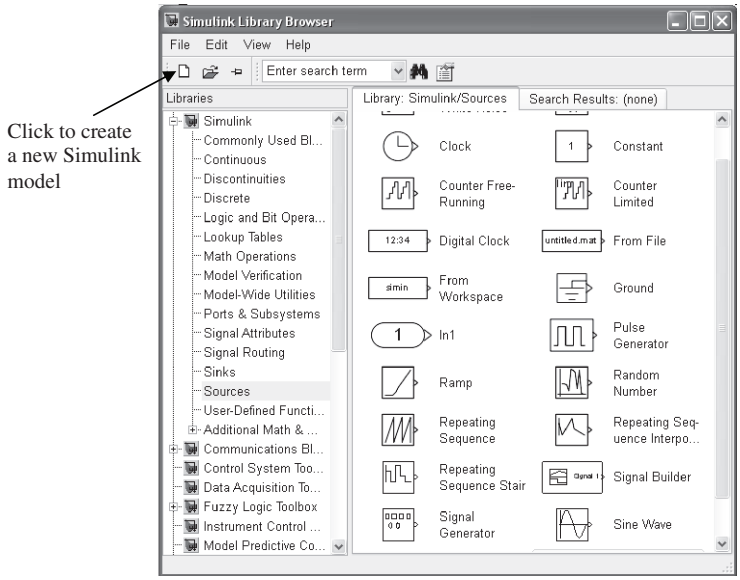


Figure 3.46 Simulink library browser.

Step 2 Building a Simulink Model of Park's Transformation

Figure 3.48 shows the library of 'Simulink/Source'. In the library, the 'Constant block', 'Sine wave' block, and 'Repeating Sequence' block can be found. Paths of other blocks can be found

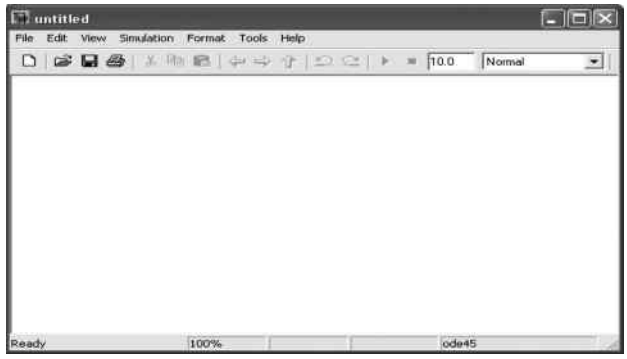


Figure 3.47 A blank Simulink model.

in Table 3.2 (in the rightmost column named 'Libraries and path'). Drag these blocks into the new model and connect them to build the Simulink model as shown in Figure 3.48.

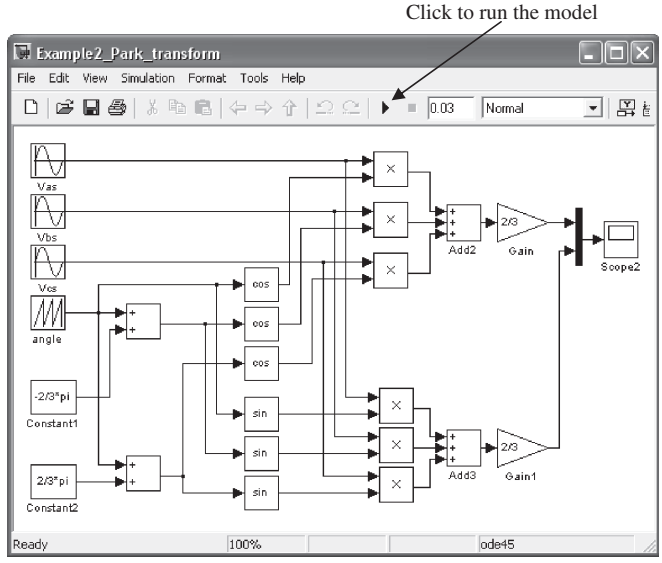


Figure 3.48 Simulink model of Park's transformation.

Step 3 Parameter Configuration of Simulink Model

The Simulink block parameters and simulation parameters need to be configured, separately. The parameters of the Simulink blocks are listed in Table 3.2. Click to open each block to edit the parameters.

The simulation parameters need to be configured with the following values.

1. Simulation time is from 0 to 0.03 s.
2. Max step size is 0.0001s.

Table 3.2 Parameters of Simulink blocks.

Blocks	Parameters (unspecified is default value)	Libraries and path
Vas	Amplitude: 1; Frequency: 60*2*pi; Phase: 0	Simulink/Sources/Sine wave
Vbs	Amplitude: 1; Frequency: 60*2*pi; Phase: −2/3*pi	Simulink/Sources/Sine wave
Vcs	Amplitude: 1; Frequency: 60*2*pi; Phase: 2/3*pi	Simulink/Sources/Sine wave
angle	Time values: [0 0.01]; Output values: [0 2*pi]	Simulink/Sources/Repeating Sequence
Constant1	Constant value: −2/3*pi	Simulink/Sources/Constant
Constant2	Constant value: 2/3*pi	Simulink/Sources/Constant
sin	Function: sin	Simulink/Math Operations/ Trigonometric Function
cos	Function: cos	Simulink/Math Operations/ Trigonometric Function
product	Default value	Simulink/Math Operations/Product
Add	List of signs: + + +	Simulink/Math Operations/Add
Gain	Gain: 2/3	Simulink/Math Operations/Gain
Scope2	Default value	Simulink/Sinks/Scope

3. Other parameters are default values.

Press down 'Ctrl + E' keys to open and edit the simulation parameters as shown in Figure 3.49.

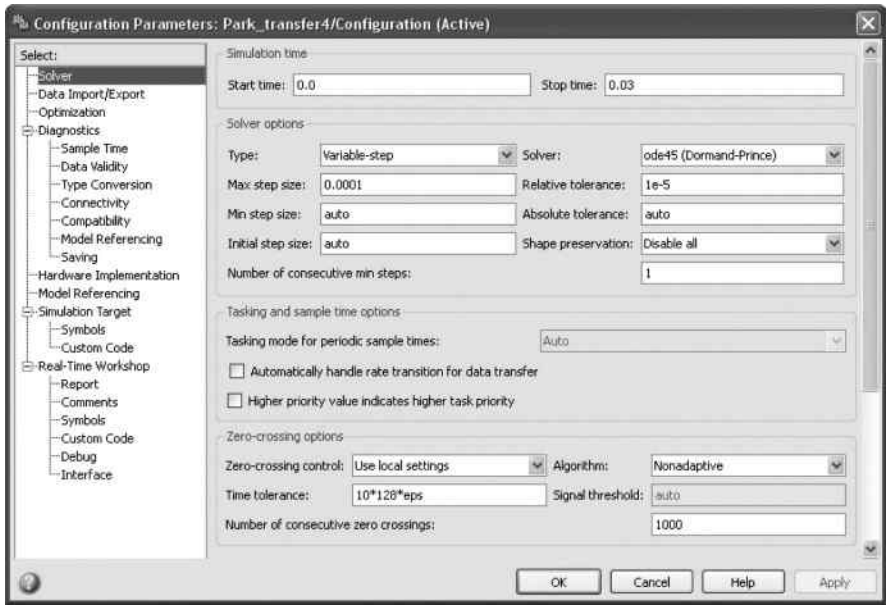


Figure 3.49 Configuration parameters of the simulation.

Step 4 Running the Simulink Model of Park's Transformation

Click the run icon shown in Figure 3.49 to start the simulation of Park's transformation. The simulation results may be observed by clicking the 'Scope2' block as shown in Figure 3.50.

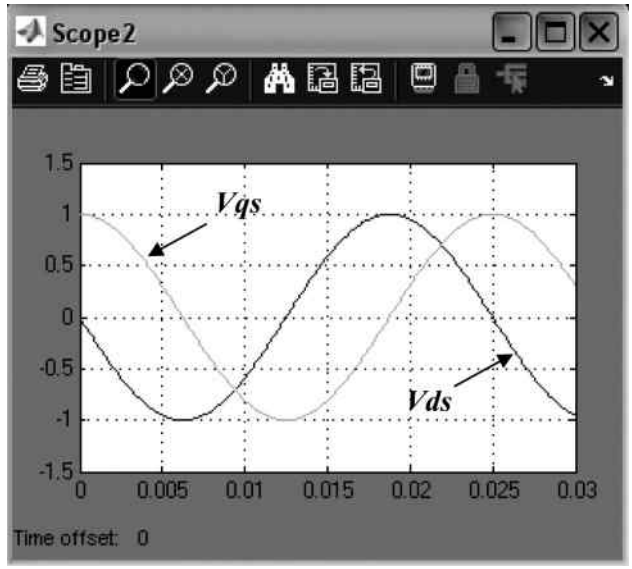


Figure 3.50 Simulation results of Example 3.2.

Example 3.3 Programming of Park's Transformation using Simulink Blocks and Self-Defined Expression

In order to simplify the Simulink model of Park's transformation, a user-defined function block, named 'Fcn', is employed. The block may be found through the library path 'Simulink/User-Defined Functions'. Drag the block from the library into the Simulink model to build the model of Park's transformation as shown in Figure 3.51.

Click the two 'Fcn' blocks as shown in Figure 3.51 separately to input the self-defined functions into the expression line.

In 'Fcn1' block, 'Sample time' is the default value and the following expression is input: 2/3*(cos(u(4))* u(1) + cos(u(4) −2/3*pi)* u(2) + cos(u(4) + 2/3*pi)* u(3)). Figure 3.52 shows how the expression is input into the 'Fcn1' block.

For 'Fcn2' block, the following expression is input:

```
2/3*(sin(u(4))* u(1) + sin(u(4)-2/3*pi)*
u(2) + sin(u(4) + 2/3*pi)* u(3))
```

In Example 3.3, simulation parameters, such as simulation time and step size are same as those in Example 3.2. The simulation results may be observed by clicking 'Scope2' in the Simulink model as shown in Figure 3.51. The simulation results of Example 3.3 are same as those in Example 3.2.

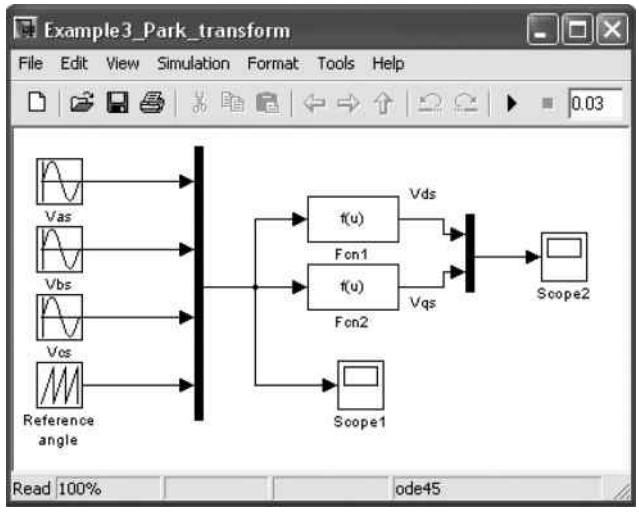


Figure 3.51 Simulink model of Example 3.3.

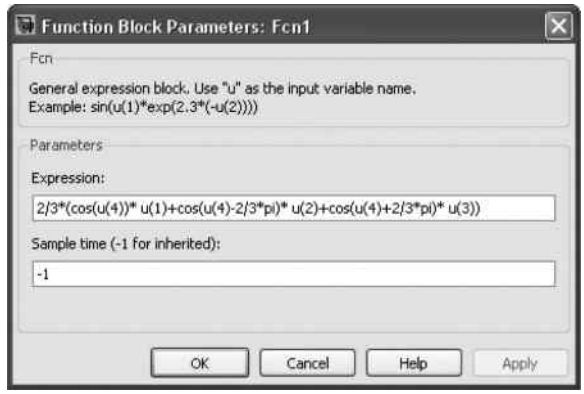


Figure 3.52 Self-defined function being entered into 'Fcn1'.

Example 3.4 Programming of Park's Transformation Using Simulink Blocks which Calls a MATLAB® Function

In this example, a MATLAB® function is called by a Simulink model to implement Park's transformation. MATLAB® function calls are useful in some applications, for example, in the implementation of extended Kalman filter in a Simulink model.

Step 1 Create a MATLAB® Function of Park's Transformation which may be Called by Simulink Model

As in Example 3.1, a new MATLAB® file may be created by clicking 'File' and selecting 'New' in MATLAB® window. Input the following code into the new file and save it with name

'Example4_Park_Transform.m'.

```
function Vdqs=Example4_Park_Transform(x)
Vdqs(1) = 2/3*(cos(x(4))*x(1) + cos(x(4)-
2/3*pi)*x(2) + cos(x(4) + 2/3*pi)*x(3));
Vdqs(2) = 2/3*(sin(x(4))*x(1) + sin(x(4)-
2/3*pi)*x(2) + sin(x(4) + 2/3*pi)*x(3));
end
```

The above code defines a MATLAB® function whose name is 'Example4_Park_Transform.m'. The output of the function is a vector 'Vdqs' which includes two elements, Vdqs(1) and Vdqs(2), which represent Vds and Vqs, respectively. 'x' is input of the function. Elements of 'x', x(1), x(2), and x(3) represent Vas, Vbs, and Vcs separately, while x(4) represents the transformation reference angle.

Step 2 Building a Simulink Model to Call the MATLAB® Function

A 'MATLAB Fcn' block is used to call a MATLAB® function. The block may be found through library path 'Simulink/User-Defined Functions'. Drag the block from the library into a Simulink model to build the model of Park's transformation as shown in Figure 3.53.

Click the 'MATLAB Fcn' block in Figure 3.53 to open it as shown in Figure 3.54. In the 'MATLAB function' window, type in 'Example4_Park_Transform' and other parameters are default values.

Click the 'OK' icon to finish the block configuration.

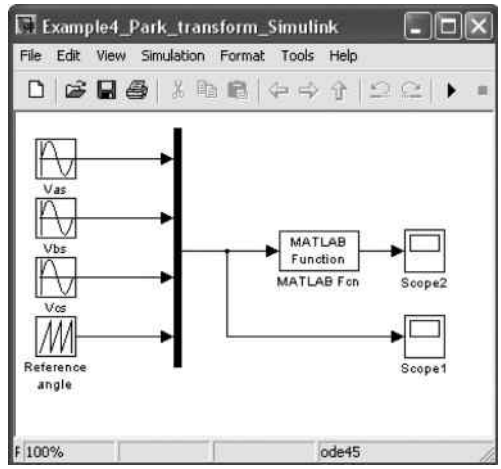


Figure 3.53 Simulink model of Example 3.4.

Step 3 Running the Simulink Model of Example 3.4

The simulation results of Example 3.4 are the same as those in previous examples, and they may be observed by clicking 'Scope1' and 'Scope2' in the Simulink model as shown in Figure 3.53. The waveforms in 'Scope1' are the same as those in Figure 3.43 in Example 3.1 and the waveforms in 'Scope2' are same as those in Figure 3.50.

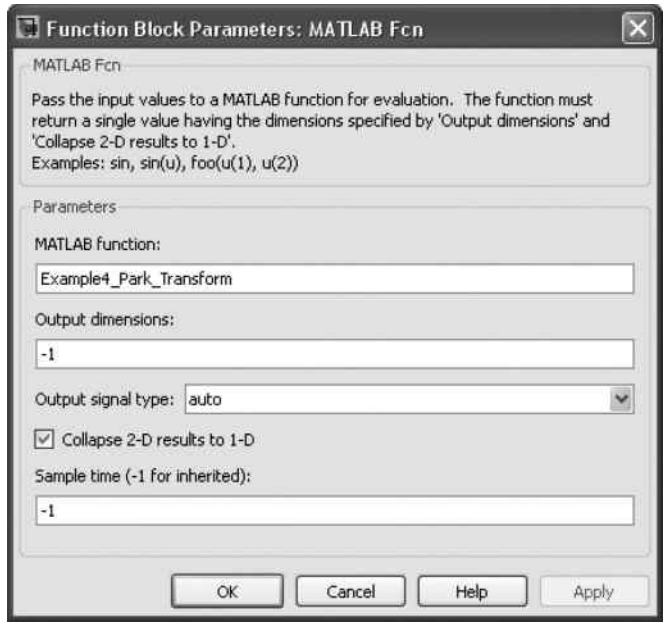


Figure 3.54 'MATLAB Fcn' block configuration.

Example 3.5 Programming of Park's Transformation by a MATLAB® Function which Calls a Simulink Model

This example illustrates how to call a Simulink model from a MATLAB® code. This operation is convenient for some applications, for example, optimization of a Simulink model using genetic algorithm (which is a MATLAB® function in the present MATLAB®/Simulink version).

Step 1 Building a Simulink Model

A 'To Workspace' block is employed to store data into workspace and the data will be called by MATLAB® function. The block may be found through library path 'Simulink/Sinks'. Drag the block from the library into the Simulink model to build the Simulink model of Example 3.5, as shown in Figure 3.55.

Click to open the 'To Workspace' block in Figure 3.55. Enter 'x' into the 'Variable name' window and select 'Array' in the 'Save format' window. Other parameters are the default values as shown in Figure 3.56.

Click the 'OK' icon to finish the block configuration. The new Simulink is saved with name 'Example5_Park_transform.mdl', where the suffix '.mdl' means Simulink model.

Step 2 Creating a New MATLAB® Function which Calls the Simulink Model

A new MATLAB® file may be created by clicking 'File' and selecting 'New' in the MATLAB® window. Enter the following code sequence and save it with the filename 'Example5_Park_transform.m'.

```
clear
sim('Example5_Park_transform',0.03);
Vas = x(:,1);
Vbs = x(:,2);
Vcs = x(:,3);
Ra = x(:,4);
t = tout;
A = size(x);
for i = 1:A(1);
Vds(i)= 2/3*[cos(Ra(i))', cos(Ra(i)-2/3*pi)', cos(Ra(i) + 2/3*pi)']*
[Vas(i);Vbs(i);Vcs(i)];
Vqs(i) = 2/3*[sin(Ra(i))', sin(Ra(i)-2/3*pi)',
sin(Ra(i) + 2/3*pi)']*[Vas(i);Vbs(i);Vcs(i)];
end
plot(t, Vds, 'blue');
hold on;
plot(t, Vqs, 'red');
```

In the file, 'sim' is a MATLAB® function which is used to simulate a Simulink model. It takes two parameters: one is name of the Simulink model ('Example5_Park_transform') and the other is the simulation time (0.03 s).

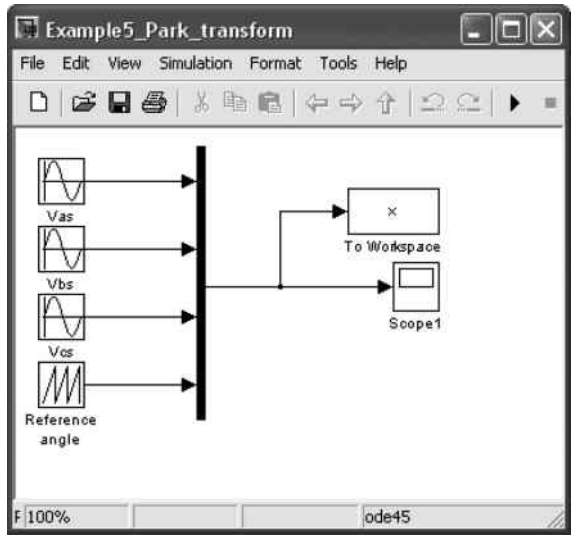


Figure 3.55 Simulink model of Example 3.5.

Step 3 Running the MATLAB® Function of Example 3.5

In MATLAB® command line, type 'Example5_Park_transform' to run Example 3.5 and the results are obtained automatically, as shown in Figure 3.57.

The simulation results shown in Figure 3.57 are same as the previous examples.

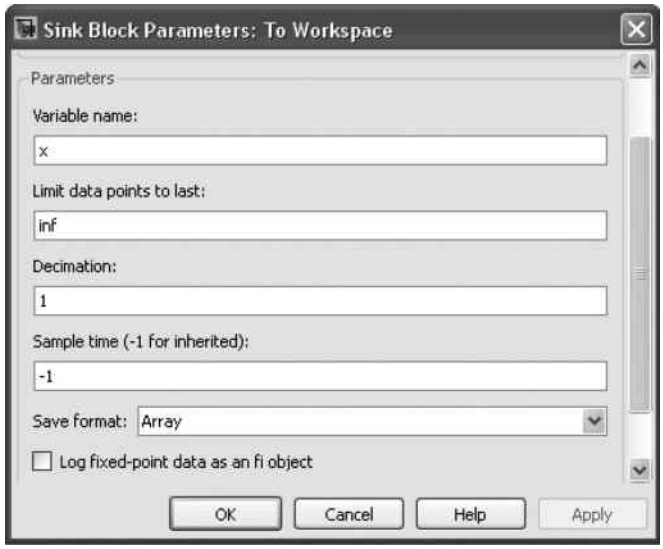


Figure 3.56 Parameter configuration.

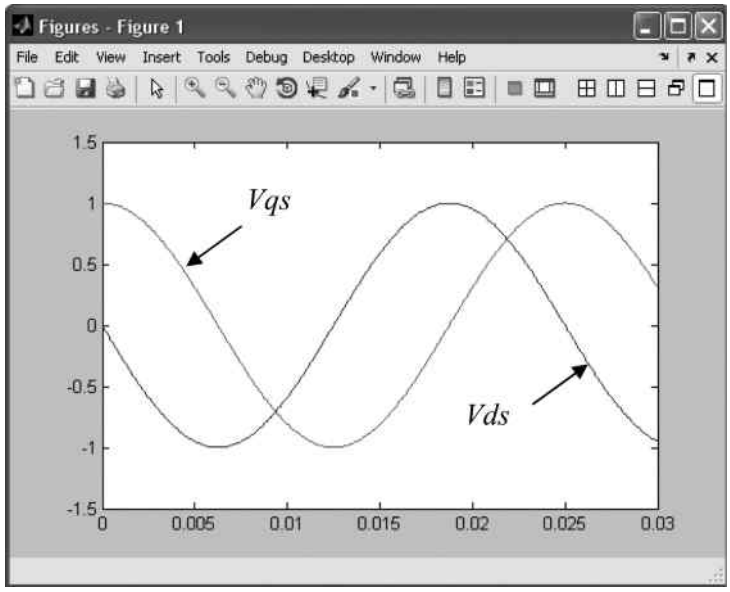


Figure 3.57 Simulation results of Example 3.5.

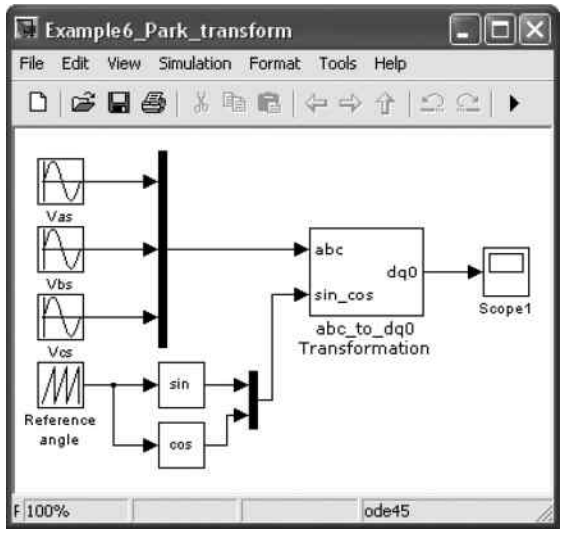


Figure 3.58 Simulink model of Park's transformation using 'abc_to_dq0' block.

Example 3.6 Programming of Park's Transformation by Calling 'abc_to_dq0 Transform' Block

In this example, a simulink model of Park's transformation is built by employing the block 'abc_to_dq0 Transform' in Simulink library. The block may be found from path 'SimPowerSystem/Extra_Library/Measurements'. Drag the block into the Simulink model as shown in Figure 3.58.

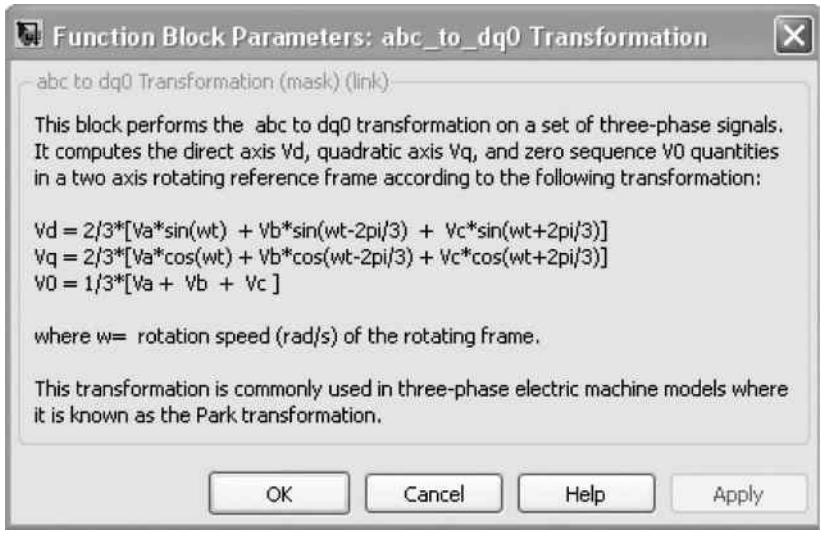


Figure 3.59 Descriptions of 'abc_to_dq0 Transformation' block.

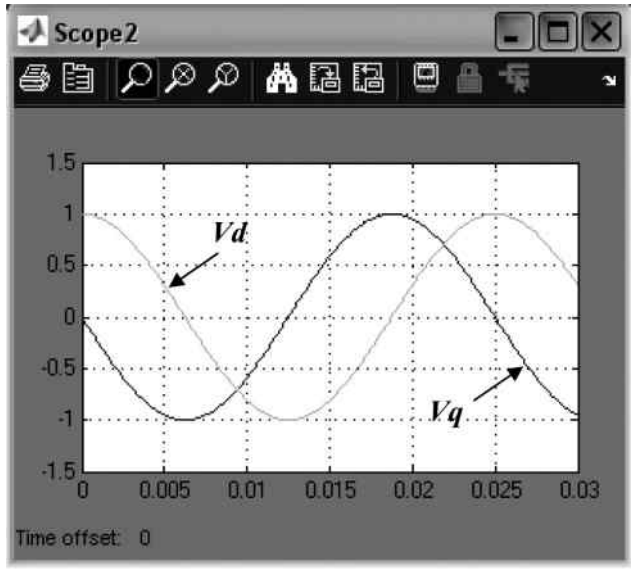


Figure 3.60 Simulation results of Example 3.6.

As shown in Figure 3.58, input of transformation reference in the block is 'sin_cos'. In order to match the input format, two trigonometric function blocks, 'sin' and 'cos' are used. Click the 'abc_to_dq0 Transformation' block to see the description as shown in Figure 3.59.

There are differences between the 'abc_to_dq0' transformation and the original Park's transformation by comparing Equation (3.26). *Vd* defined in the Simulink block is *Vqs* in the original Park's transformation while *Vq* defined in the Simulink block is *Vds*. In using the 'abc_to_dq0' block, *Vd* has to be interpreted as *Vqs* in the original Park's transformation and *Vq* has to be interpreted as *Vds*.

Figure 3.60 shows the simulation results of Example 3.6 which calls the 'abc_to_dq0' block. Compared with Figure 3.50, it is seen that *Vd* lags *Vds* by $2/3\pi$ rad and *Vq* lags *Vqs* by the same angle.

3.11 Summary

Based on the 8th and 11th fifth-order equations, the Simulink models of induction motor presented in this chapter are useful for transient analysis of the induction motor. The current-input model of induction motor involves the least amount of calculations and, hence, is computationally efficient and can be used for the study of a current-controlled induction motor drive system. The voltage-input model is realized using the matrix calculation blocks and can be used for the study of a voltage-controlled induction motor drive system. The discrete-state model uses an 'S-function' block with a short M-file (which consists of about twenty program lines) to implement the discrete fifth-order equation of a three-phase induction motor, giving the most compact modeling form. Because the discrete-state model involves lengthy calculations and gives accurate simulation results only when the sampling time *M* is less than 0.8 ms, a long

program execution time is required. Each block of the models may be connected and modified easily in the MATLAB®/Simulink environment. Some limit conditions, such as saturation of magnetic circuit and stator current limit, may be easily inserted into the function blocks. The object modeling methods of the current-input model and voltage-input model are convenient because program compilation is not required. As a subsystem, the three models may be easily incorporated into a sophisticated control system of the induction motor. The sinusoidal PWM model and encoder model may be used to simulate a PWM generator and a speed measurement device making the system model of the induction motor drive more realistic. Further work on induction motor modeling may include: (1) incorporating more practical factors in the model, such as magnetic saturation, temperature rise, motor vibration, and unbalanced parameters, (2) automatically generating a computer model based on measured data of the actual induction motor. This chapter has also provided a primer for readers to familiarize with the MATLAB®/Simulink programming environment. The techniques introduced will be employed in the intelligent control of induction motor drives detailed later in the book.

References

Chan, C.C., Jiang, J.Z., and Chen, G.H. (1993) Computer simulation and analysis of a new polyphase multipole motor drive. *IEEE Transactions on Industrial Electronics*, **40**(1), 62–77.

Domijan, A. Jr and Yin, Y. (1994) Single phase induction machine simulation using the electromagnetic transients program: theory and test cases. *IEEE Transactions on Energy Conversion*, **9**(3), 535–542.

Krause, P.C. and Thomas, C.H. (1965) Simulation of symmetrical induction machinery. *IEEE Transactions on Power Apparatus and System*, **PAS-84**(11), 1038–1053.

Krause, P.C., Wasynczuk, O., and Sudhoff, S.D. (1995) *Analysis of Electric Machinery*, IEEE Press, New Jersey.

Lewis, F.L. (1992) *Applied Optimal Control and Estimation*, Prentice-Hall, Inc., New Jersey.

The MathWorks, Inc. (2008a) MATLAB® Getting Started Guide. The MathWorks, Inc.

The MathWorks, Inc. (2008b) Simulink Getting Started Guide. The MathWorks, Inc.

The MathWorks, Inc. (2008c) Simulink Graphical User Interface. The MathWorks, Inc.

Mohan, N., Robbins, W.P. *et al.* (1997) Simulation of power electronic and motion control system, in *Power Electronics and Variable Frequency Drives: Technology and Applications* (ed. B.K. Bose), IEEE Press, New Jersey.

Trzynadlowski, A.M. (1994) *The Field Orientation Principle in Control of Induction Motors*, Kluwer Academic Publishers, Boston.

4

Fundamentals of Intelligent Control Simulation

4.1 Introduction

Computer simulation provides a fast way to verify new designs and improvements at the algorithm level, without incurring hardware cost and possible device damage. Currently, MATLAB® (The MathWorks, Inc., 2008a) software is an ideal tool for simulating most intelligent control systems. A large number of function blocks have been developed in the Simulink libraries (The MathWorks, Inc., 2008b) by Mathworks Inc. Various mathematical operations, engineering models, and software tools are packaged into the blocks. With the aid of these blocks, users can avoid repeated effort in modeling when verifying or developing a custom design.

In this chapter, readers will learn how to model and simulate a fuzzy logic based PI controller, neural network based Park's transformation, signal measurement using Kalman filter, and a genetic algorithm optimized PID controller using MATLAB®/Simulink. Control System Toolbox, Fuzzy Logic Toolbox, Neural Network Toolbox, Signal Processing Blockset in Simulink libraries will be employed for the simulation examples.

4.2 Getting Started with Fuzzy Logic Simulation

MATLAB® (version R2008b) delivers a 'Fuzzy Logic Toolbox' (The MathWorks, Inc., 2008c). In the toolbox, a block named 'Fuzzy Logic Controller' may be used to simulate a fuzzy logic controller in Simulink platform. The MATLAB® command 'Fuzzy' calls the FIS (Fuzzy Inference System) editor that enables membership functions and fuzzy inference rules to be input and edited. In this section, an example of fuzzy PI control system is presented.

4.2.1 Fuzzy Logic Control

A basic fuzzy inference system is described as shown in Figure 4.1.

The basic fuzzy inference system may be simulated in MATLAB®, and the operations are summarized in Table 4.1.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

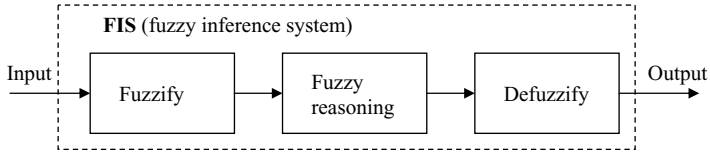


Figure 4.1 A basic fuzzy inference system.

Table 4.1 Operations of fuzzy inference system in MATLAB®.

Name	Action	Parameter	Edit Window	How to Enter
FIS (fuzzy inference system)	Fuzzy inference	Membership functions and rulebase	FIS Editor	Entering command 'fuzzy' on the MATLAB® main window
Fuzzify	Map inputs to membership values	Input membership function	Membership Function Editor	Enter from FIS Editor window (see Figure 4.9)
Fuzzy reasoning	Fuzzy reasoning based on fuzzy rulebase and if-then logic	Output membership function	FIS Editor/ Membership Function Editor	Enter from FIS Editor window (see Figure 4.9)
		Fuzzy rulebase	FIS Editor/Rule Editor	Enter from FIS Editor window (see Figure 4.10)
Defuzzify	Transfer fuzzy values to crisp outputs	N/A	N/A	N/A

The steps of developing a fuzzy simulation model are shown in Figure 4.2.

Figure 4.2 shows the basic steps of developing a fuzzy inference system in MATLAB®/Simulink. Firstly, the fuzzy inference system is created and its parameters are configured by the 'FIS Editor' of MATLAB®. Then, the fuzzy inference system is exported under MATLAB® path with its own filename. After the filename is entered into a 'Fuzzy Logic Controller' block, the fuzzy inference system may be called by the Simulink model.

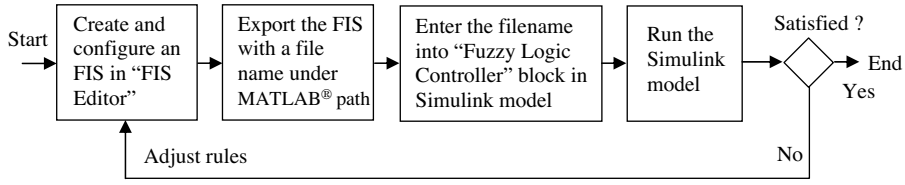


Figure 4.2 Steps of developing a fuzzy model in MATLAB®/Simulink.

4.2.2 *Example: Fuzzy PI Controller*

An example of a basic fuzzy controller (Sousa and Bose, 1995) is presented in this section. The purpose is to demonstrate the programming process of fuzzy logic simulation in MATLAB®/Simulink. For comparison purpose the performance of a PI controller is also studied.

Step 1 Building a PI Control System in Simulink

A PI controller is described by the expression $u(t) = K_p e(t) + K_i \int e(t)dt$, where $e(t)$ is input of the controller, $u(t)$ is output of the controller, K_p is the proportional gain, and K_i is the integral gain. A Simulink model of PI control system may be constructed as shown in Figure 4.3.

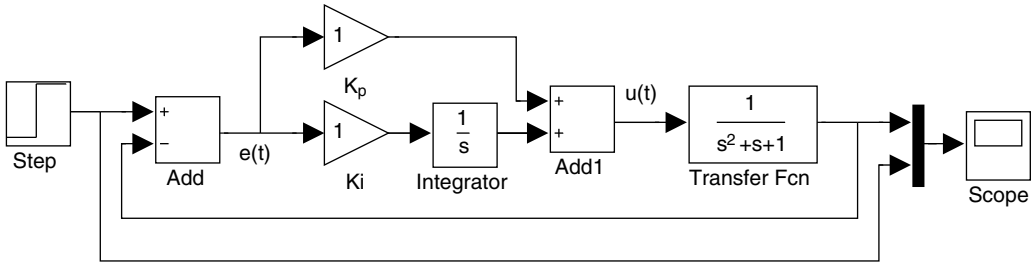


Figure 4.3 A PI control system in Simulink.

The PI control system in Figure 4.3 consists of 'Step', 'Add', 'Gain', 'Integrator', 'Transfer Fcn', and 'Scope' blocks. These blocks may be found in 'Simulink Library'. The properties of the blocks are summarized in Table 4.2. For example, the block 'Step' is used to generate a step function from the Sources Group in the Simulink libraries. The 'Transfer Fcn' is provided in

Table 4.2 Simulation blocks in PI control system.

Block	Functions	Parameters	Libraries/Group
Step	Generate a step function	Step time = 5 s Initial value = 0 Final value = 1	Simulink/Sources
Add	Add or subtract inputs	+ − + +	Simulink/Math Operations
Gain	Multiply input by a constant	Proportional Gain = 1 Integral Gain = 1	Simulink/Math Operations
Integrator	Integrate signal	N/A	Simulink/Continuous
Transfer Fcn	Transfer function	Numerator coefficient = 1 Denominator coefficient = [1 1 1]	Simulink/Continuous
Scope	Display signals generated during simulation	Y_min = − 0.5 Y_max = 1.5	Simulink/Sinks

Continuous Group in the Simulink libraries. The parameters of the blocks may be set by clicking the individual blocks.

Step 2 Running the Simulation for the PI Control System
After running the Simulink model of PI control system as shown in Figure 4.3, the simulation results of controller command (dot line) and system output (solid line) are obtained as shown in Figure 4.4.

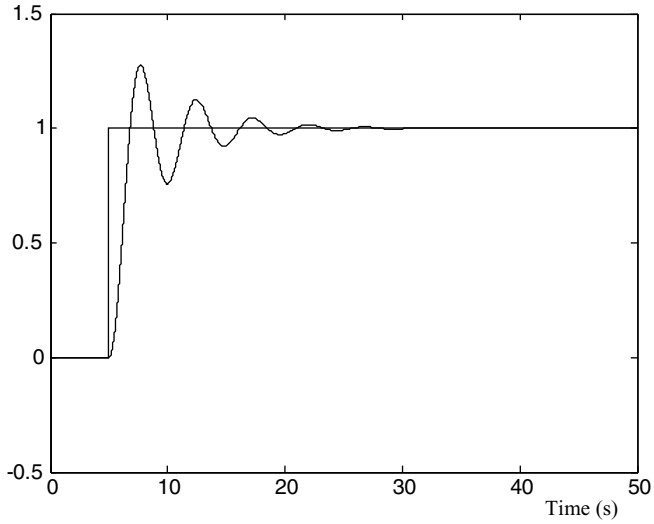


Figure 4.4 Simulation results of PI controller.

The performance of the PI controller may be further optimized by tuning the proportional gain K_p and the integral gain K_i.

Step 3 Fuzzy Logic Control Modeling in Simulink
In the previous PI controller, both the proportional gain K_p and the integral gain K_i are constant. In order to improve the controller performance, the gains should be varied as $e(t)$ varies. This is accomplished by varying K_p and K_I according to some fuzzy rules when the absolute value of error of 'Transfer Fcn' block output and input command, $e(t)$ (controller's input) varies. For example, the rules may be defined as follows:

If abs($e(t)$) is large, then K_p is large and K_i is large.
If abs($e(t)$) is small, then K_p is large and K_i is zero.
If abs($e(t)$) is zero, then K_p is large and K_i is small.

A Simulink model of fuzzy PI control system is shown in Figure 4.5 and parameters of the Simulink blocks are listed in Table 4.3.

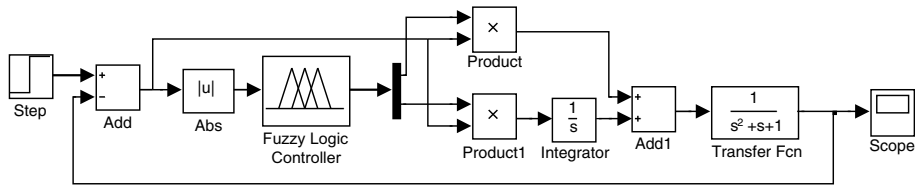


Figure 4.5 Modeling fuzzy PI control system in Simulink.

Table 4.3 Simulation blocks in Fuzzy PI control system.

Name	Function	Libraries/Group	Parameter
Abs	Output absolute value of input	Simulink/Math Operations	N/A
Fuzzy Logic Controller	Fuzzy inference system	Fuzzy Logic Toolbox	FIS file is 'fis2'
Product	Multiply the inputs	Simulink/Math Operations	Number of inputs is 2

Step 4 Configuring the Fuzzy Inference System in MATLAB®

After the command 'Fuzzy' is entered in the MATLAB® command window, a FIS (Fuzzy Inference System) editor window will appear as shown in Figure 4.6.

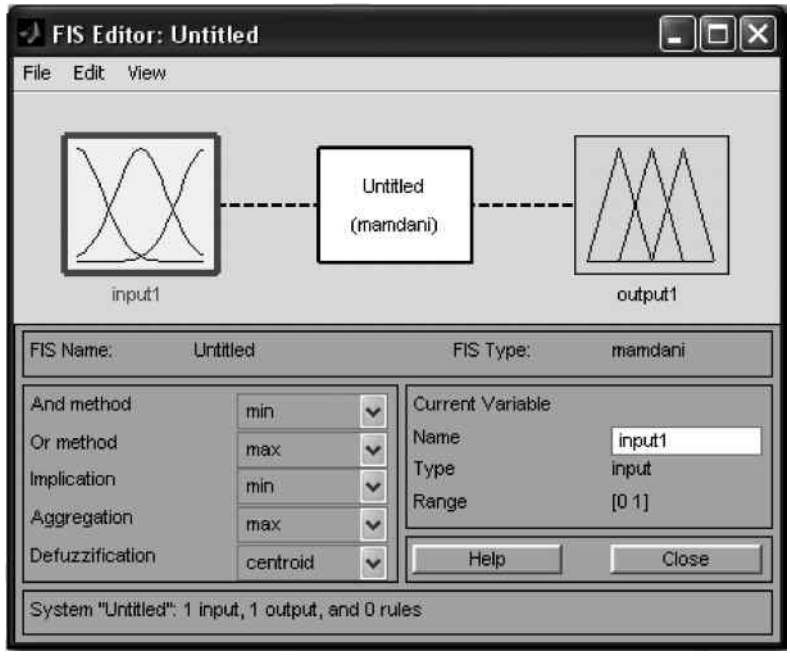


Figure 4.6 FIS (Fuzzy Inference System) editor.

There is only one input and one output in the newly opened FIS by default, whereas the proposed fuzzy controller has two outputs, one for adjusting K_p and another for adjusting K_i. One more output should thus be added. This is done by clicking 'Edit' on the FIS Editor and selecting 'Add Variable' and 'Output', as shown in Figure 4.7.

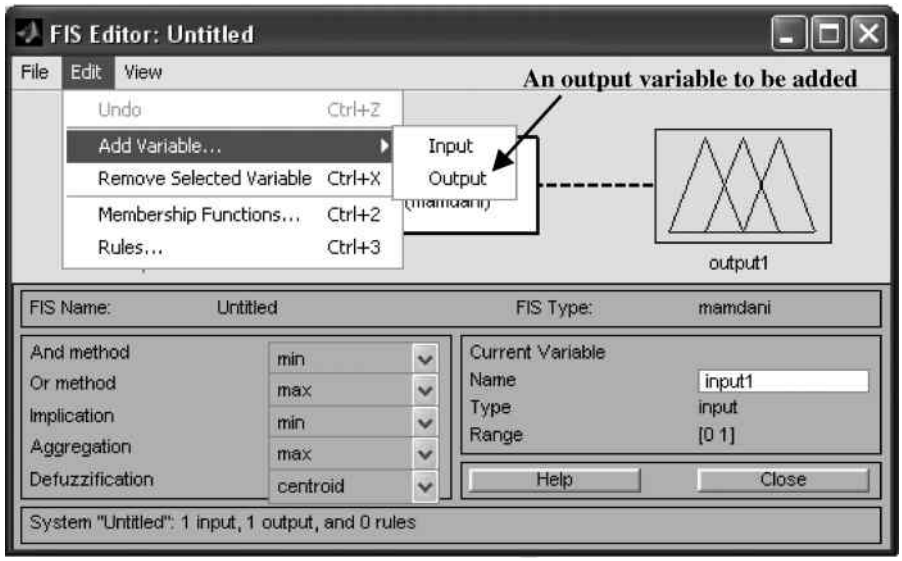


Figure 4.7 Adding an output variable in FIS editor.

After the above operation, a new output, 'output2', is added in the FIS editor as shown in Figure 4.8.

Double-click any one input or output box in the 'FIS Editor'. A 'Membership Function Editor' window appears as shown in Figure 4.9.

The membership function of the input and output need to be configured manually in the 'Membership Function Editor' with parameter values given in Table 4.4.

Double-click the fuzzy rule box 'Untitled mamdani' in the center of the FIS editor to bring up the 'Rule Editor' window as shown in Figure 4.10. Fuzzy inference rules may be configured manually in the 'Rule Editor' with the contents in Table 4.5.

When the editing of the Fuzzy Inference System is completed, click the 'File' icon and export the Fuzzy Inference System with a filename. The filename should match the parameter in the 'Fuzzy Logic Controller' block as listed in Table 4.3. In this example, the filename is 'fis2'. Also, the file should be saved to current MATLAB® paths, in order that it can be called by the Simulink model of Fuzzy based PI controller as shown in Figure 4.5.

Step 5 Running the Simulation for the PI Controller and Fuzzy Controller

Run the Simulink model of the PI control system as shown in Figure 4.3 and the Simulink model of the fuzzy PI control system as shown in Figure 4.5. The simulation results of PI controller (dot line) and fuzzy controller (solid line) are displayed, as shown in Figure 4.11.

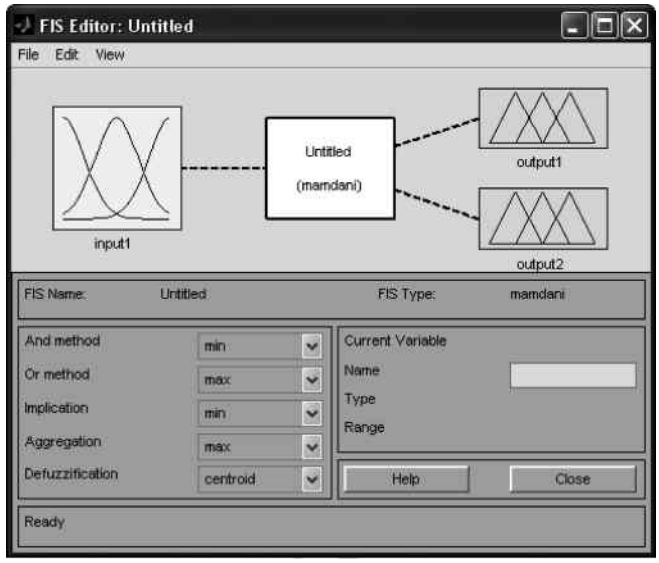


Figure 4.8 One output added in FIS editor.

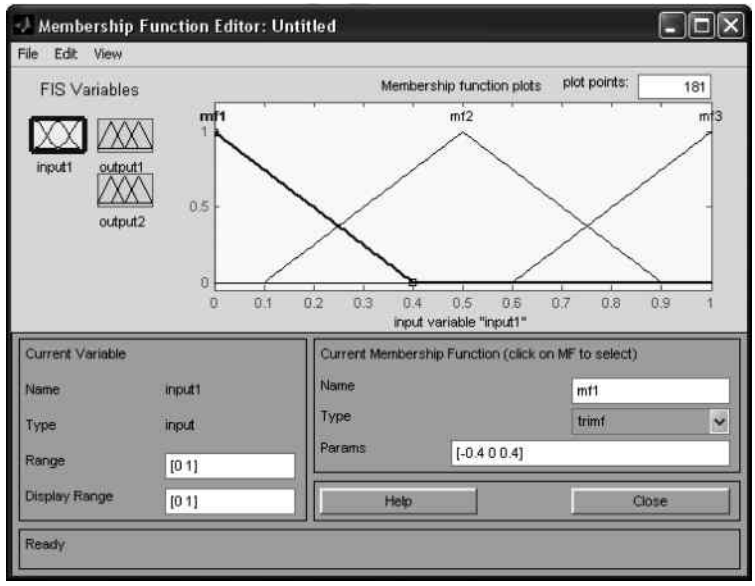


Figure 4.9 Membership function editor.

Table 4.4 Parameters of membership function.

Membership Function	Range	Type	Parameters
Input1	[0 1]	Trimf	[−0.5 0 0.5]
Output1	[0 1]	Trimf	[−0.5 0 0.5]
Output2	[0 1]	Trimf	[−0.5 0 0.5]

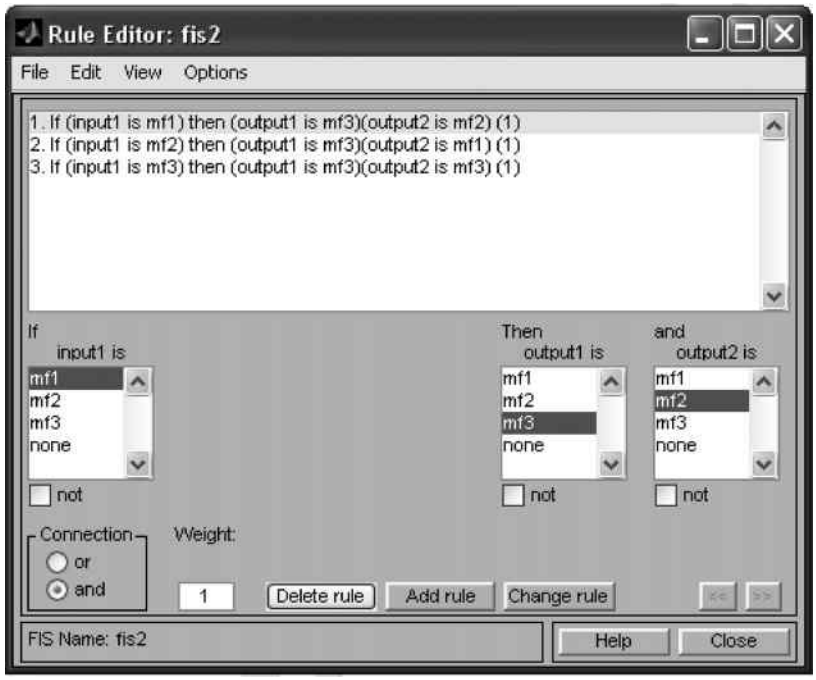


Figure 4.10 Rule editor.

Table 4.5 Fuzzy inference rules.

Rule No	Conditions		Actions	
1	If input is zero	abs($e(t)$) is zero	output1 is large and output2 is small	K_p is large and K_I is small
2	If input is small	abs($e(t)$) is small	output1 is large and output2 is zero	K_p is large and K_I is zero
3	If input is large	abs($e(t)$) is large	output1 is large and output2 is large	K_p is large and K_I is large

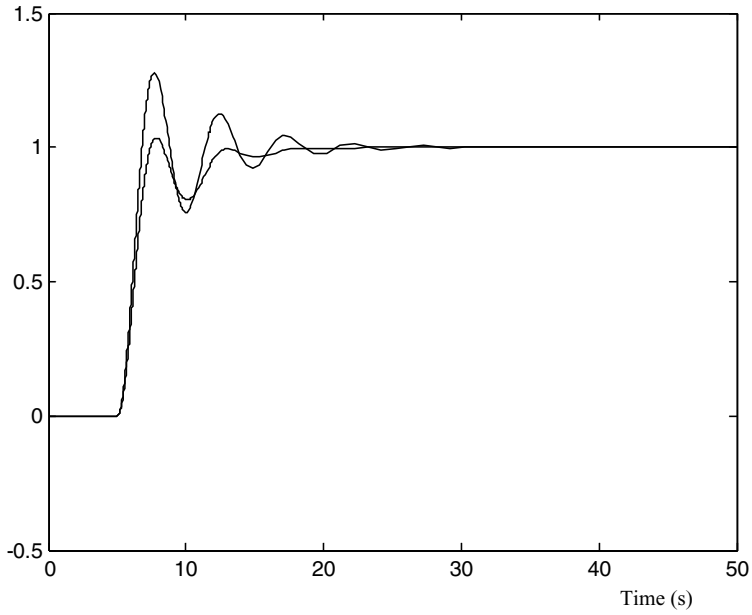


Figure 4.11 Performance of PI controller and fuzzy PI controller.

The above simulation results show that the basic fuzzy controller presented gives better performance than a conventional PI controller with fixed gains. It should be noted that optimization of structure and parameters has not been performed on the fuzzy controller yet.

4.3 Getting Started with Neural-Network Simulation

MATLAB®/Simulink delivers a 'Neural Network Toolbox' and some learning functions for simulating various neural networks (The MathWorks, Inc., 2008d). A neural network may be built and trained in MATLAB® command line or in a graphic-guide window. In the toolbox, various neuron transfer functions are packaged into blocks that can be dragged into a Simulink block diagram for building a neural-network model. Alternatively, a neural-network model may be built by entering appropriate commands into MATLAB® command window. In this section, neural-network based Park's transformation will be implemented by MATLAB®/Simulink.

4.3.1 Artificial Neural Network

An artificial neural network consists of neurons and connection lines. A two-layer neural network is shown in Figure 4.12, where layer 1 is also called the hidden layer and layer 2 is also called the output layer. In a practical application, a network may have several layers each having a different number and type of neurons.

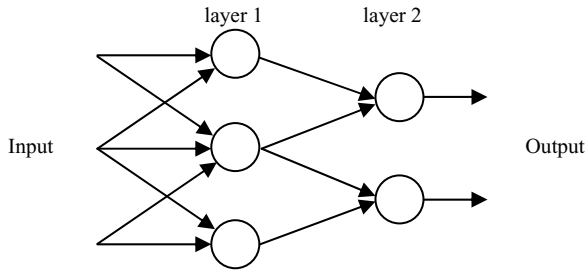


Figure 4.12 A two-layer network consisting of five neurons and connection lines.

A neuron consists of net weight (product operator), net bias (constant), net sum (add operator), and a transfer function as shown in Figure 4.13. Parameters of a neuron are listed in Table 4.6.

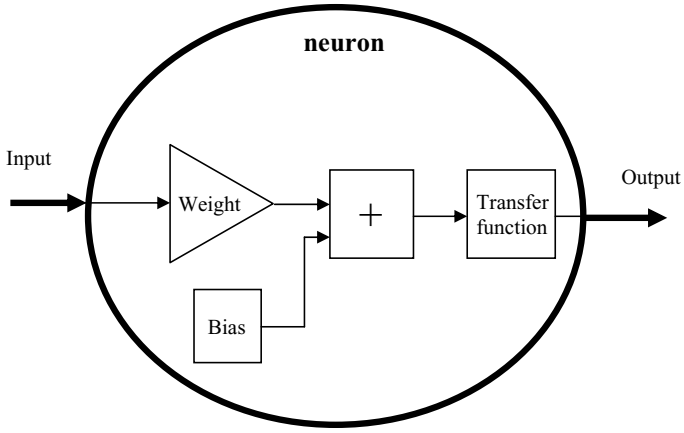


Figure 4.13 Internal structure of artificial neuron.

Table 4.6 Parameters of an artificial neuron.

Name	Properties	Achieved by
Weight	Scalar or vector gain	Learning
Bias	Scalar or vector constant	Learning
Transfer function	linear or nonlinear function	Depends on function relationship of input and output

The weights and bias of neurons may be obtained by a training process as shown in Figure 4.14.

The initial weights and biases for a back-propagation network may be created randomly. Update equation of the weight and bias of neurons is given as:

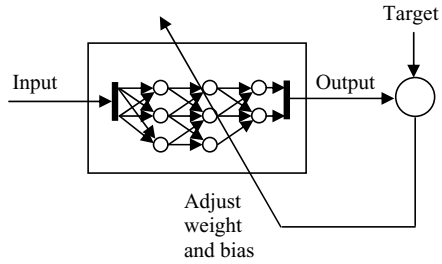


Figure 4.14 Training process of neural network.

$$w(t+1) = w(t) + \eta \frac{\partial E}{\partial w(t)}. \tag{4.1}$$

where $w(t+1)$ is a vector of the new weight and bias, $w(t)$ is a vector of the old weight and bias, η is learning rate, $\frac{\partial E}{\partial w(t)}$ is gradient descent, and E is sum squared error of the output and the target of the neural network as shown in Figure 4.14.

4.3.2 *Example: Implementing Park's Transformation Using ANN*

Park's transformation is also called *a-b-c* to *d-q* rotating transformation. With a rotating reference angle, the three-phase variables in stationary frame are transferred to the rotating *d-q* reference frame by Park's transformation. In this example, Park's transformation will be implemented by a neural network.

Park's transformation may be expressed as

$$\begin{bmatrix} V_d \\ V_q \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ \sin(\theta) & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \tag{4.2}$$

where V_a, V_b, V_c, and θ are the input vector, and V_d and V_q are the output vector.

Step 1 Design of Neural Network of Park's Transformation

Equation (4.2) expresses a product of the transformation matrix and vector $[V_a, V_b, V_c]$. Product operation is defined as an input function and no training is necessary in MATLAB®. (Due to no suitable neuron support, training a product operation is difficult in MATLAB®.) However, the transformation matrix may be implemented by a neural network with one input (angle θ) and six outputs (elements of the transformation matrix). The neural network thus consists of four 'tansig' type neurons and six 'purelin' type neurons. Using the neural network, the transformation matrix is implemented with a parallel calculation structure as shown in Figure 4.15.

Park's transformation in Equation (4.2) may be implemented by a product of $[V_a, V_b, V_c]$ and the outputs of the neural network in Figure 4.15.

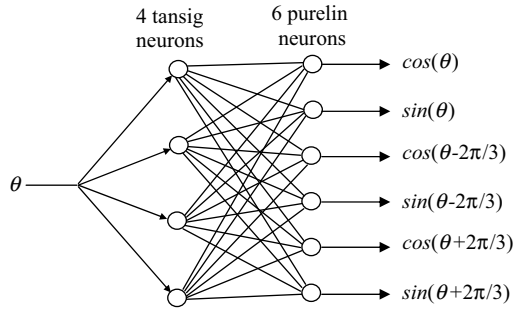


Figure 4.15 Neural network of Park's transformation matrix.

Step 2 Acquiring Samples for Training Neural Network

A Simulink model is designed to yield input and target sample sets which will be used to train a neural network, as shown in Figure 4.16. In the model, 'Repeating Sequence' block outputs repeating sequence with amplitude varying from zero to 4π rad and period equal to 2 s, which yields the rotor angle θ. The sample data of the rotor angle θ is sent to workspace with name 'sin_in' as the neural network input. The six elements of the transformation matrix in Equation (4.2) are given by six 'sin' and 'cos' blocks, separately, whose sample data are stored to 'Workspace' with name 'sin_out' as the target of training neural network. The sample rate is set as 0.001 s, so 1000 data pairs (input and target samples) are obtained by running the model for 1 second. Further details of the blocks in the model are given in Table 4.7.

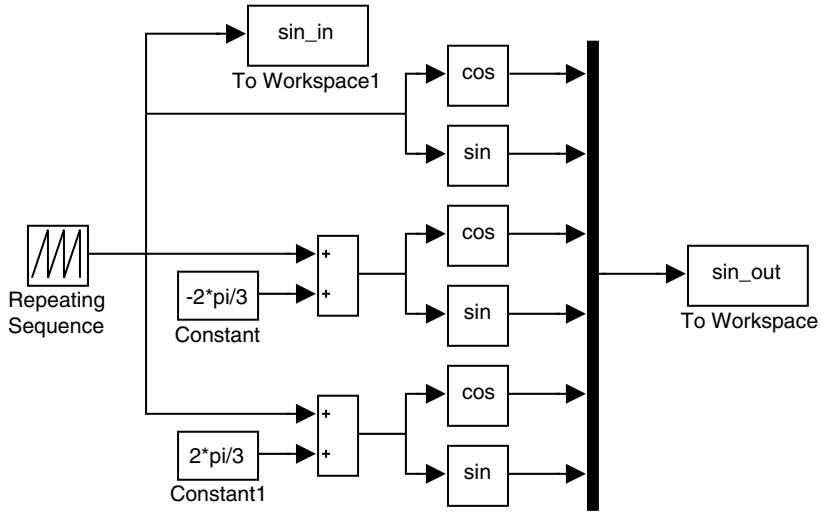


Figure 4.16 Simulink model for acquiring samples.

It is emphasized that the data in 'sin_in' and 'sin_out' blocks are stored with column formats in the workspace, while input and target arrays of the network training function in MATLAB®

Table 4.7 Simulation blocks in Park's transformation matrix.

Name	Function	Libraries/Group	Parameter
Repeating Sequence	Generate angle reference	Simulink/Sources	Time values [0,2] Output values [0,4*pi]
Trigonometric	Perform trigonometric function	Simulink/Math Operations	Sin Cos
To Workspace	Write data to MATLAB® workspace	Simulink/Sinks	Save format: Array
Simulation parameters	Manage simulation process	N/A	Start time: 0 s End time: 1s Sample time: 0.001 s

are in row formats. In order to match the network training function, the input and target samples have to be transposed. This is achieved by entering the following commands:

```
>> sinin = sin_in';
>> sinout = sin_out';
```

Step 3 Creating a New Neural Network

After **Step 2**, the sample sets of input and target are stored in row format with the new names 'sinin' and 'sinout'. Enter the following command to create a feed-forward backpropagation network:

```
>> net = newff(sinin,sinout,4);
```

The parameters of the command are listed in Table 4.8.

Table 4.8 Parameters of a feed-forward backpropagation network.

Name	Function
newff()	Create a new feed-forward backpropagation network default is 'tansig' for hidden layer and 'purelin' for output layer
net	Name of new network
sinin	Input of the network
sinout	Output of the network
4	4 'tansig' neurons in hidden layer
sinout is 6-dimension vector	6 'purelin' neurons in output layer

Step 4 Training the Neural Network

Enter the following command in order to train the new network with name 'net':

```
>> [net,tr] = train(net,sinin,sinout);
```

The parameters of the command are listed in Table 4.9.

Table 4.9 Parameters for network training.

Name	Function
train()	create a new feed-forward backpropagation network default transfer function is 'tansig'
net	Name of created network
sinin	input of the network
sinout	Train target
tr	Store training records

The training function will return new weight and bias values and these are stored in the network 'net', while the training records are stored in 'tr'. After entering the training command, a 'Neural Network Training' window as shown in Figure 4.17 appears and a training process starts automatically.

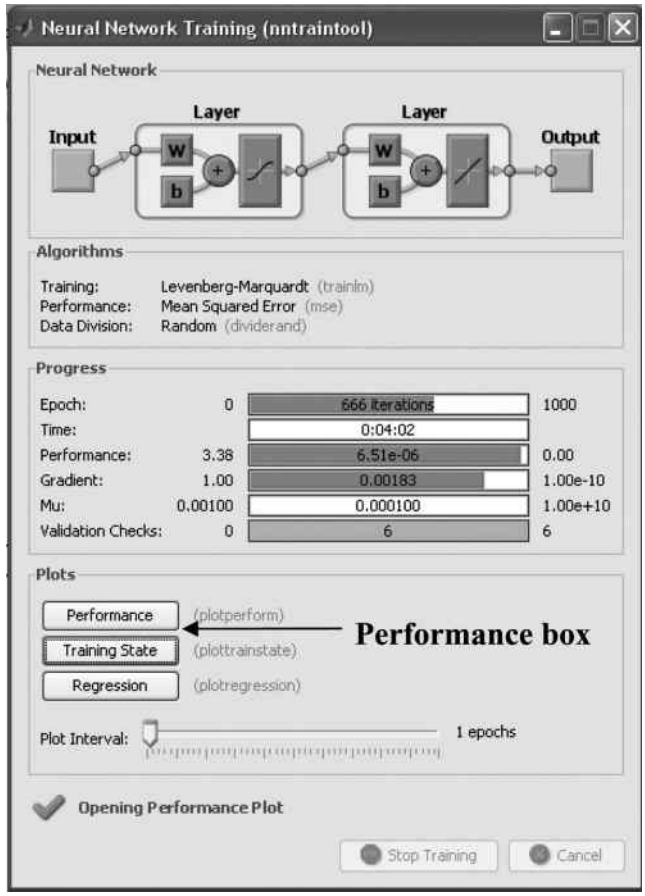


Figure 4.17 Training window.

After the training of network completed, clicking the 'Performance' box in the 'Neural Network Training' window may display the epoch process as shown in Figure 4.18. If the result is unsatisfactory, the training process may be repeated by entering the following commands again until a satisfactory result is obtained:

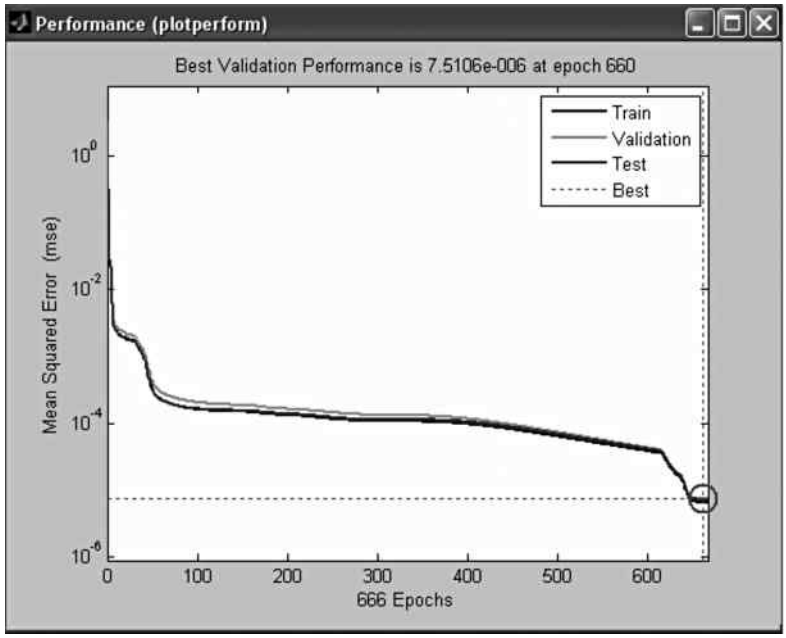


Figure 4.18 Network training performance: the mean squared error is less than 10^{-5} after 660 training epochs.

```
>> net = newff(sinin,sinout,4);
>> [net,tr] = train(net,sinin,sinout);
```

It should be noted that the results are different for each training process because the initial weight and bias of the network being trained are randomly created.

Step 5 Generating a Simulink Block of the Trained Neural Network

Enter the following command to generate a Simulink block for simulating the trained neural network 'net' for further use:

```
>> gensim(net,-1);
```

where -1 means inherited sample time.

After entering the above command, a Simulink model is automatically created as shown in Figure 4.19.

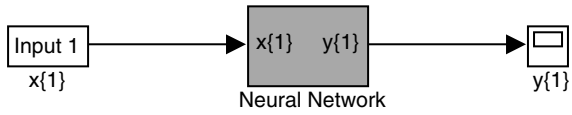


Figure 4.19 Creating a neural network by the command 'gensim(net,-1)'.

Copy the 'Neural Network' block in Figure 4.19 and paste it into a new Simulink model named as Park's Transform as shown in Figure 4.20. In the model, the 'Vabc' block yields three phase voltage [V_a, V_b, V_c] with frequency 60 Hz and magnitude 1 V. The 'Neural Network' block yields Park's transformation matrix. The 'Matrix Product' block implements product operation of [V_a, V_b, V_c] and the Park's transformation matrix to give V_d and V_q.

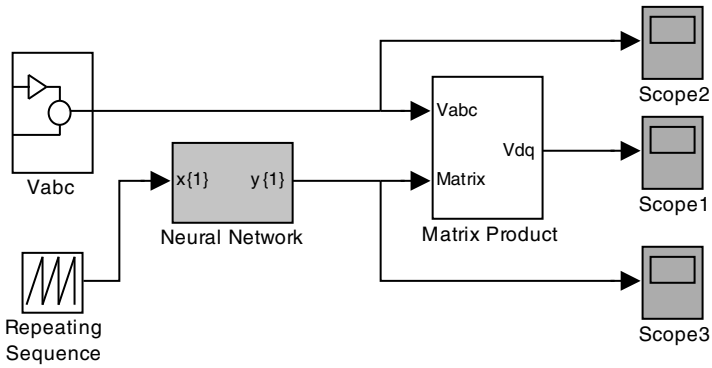


Figure 4.20 Simulink model of Park's transformation.

Step 6 Running the Simulink Model
After running the Park_ANN model, 'Scope2' displays the three phase voltage [$V_aV_bV_c$], 'Scope3' displays the output of neural work based Park's transformation, and 'Scope1' displays the voltage [V_dV_q] of rotating *d-q* reference frame. The simulation results are shown in Figures 4.21–4.23.

4.4 Getting Started with Kalman Filter Simulation

By using recursive calculations, the Kalman filter is capable of estimating the state of a linear dynamic system from a series of noisy measurements. In this section, the theory of the Kalman filter is briefly reviewed. An example of signal measurement in the presence of noise will then be presented to show how the Kalman filter can be modeled.

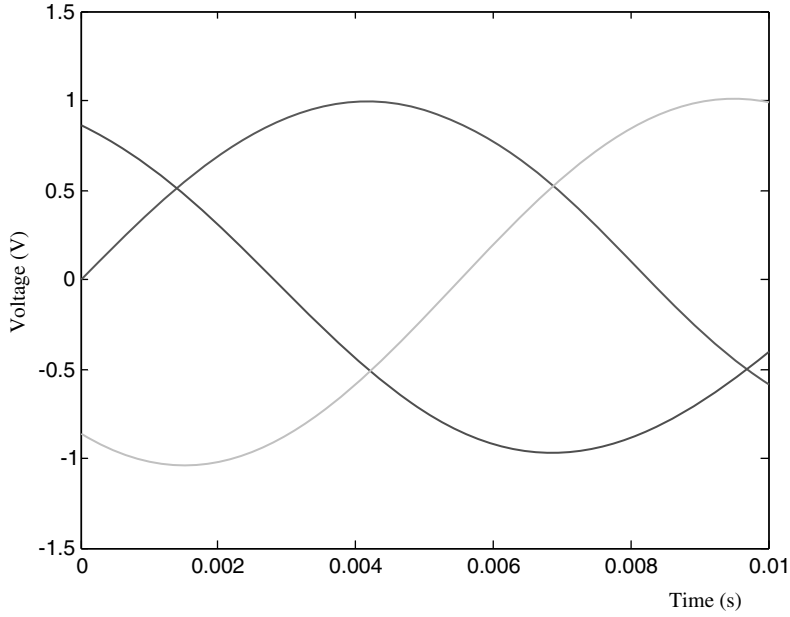


Figure 4.21 Three phase voltage $[V_a V_b V_c]$.

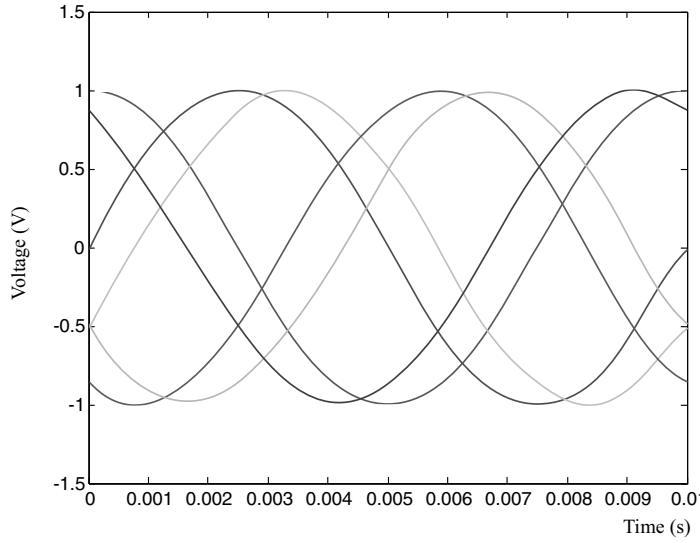


Figure 4.22 Output of neural-network based Park's transformation.

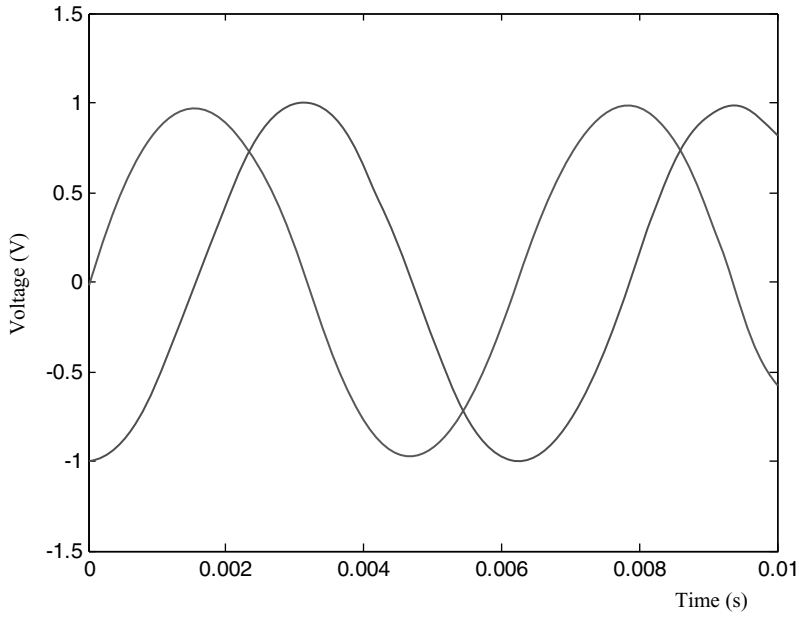


Figure 4.23 Voltage $[V_d V_q]$ output of rotating d-q reference frame.

4.4.1 Kalman Filter

A discrete state-space representation of a linear dynamic system with noises may be written in the following form.

$$\begin{aligned} x_n &= Ax_{n-1} + Bu_{n-1} + w \\ y_n &= Cx_n + v \end{aligned} \tag{4.3}$$

The discrete model may be constructed as a flowchart as shown in Figure 4.24.

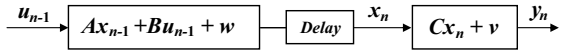


Figure 4.24 Discrete model of a linear dynamic system.

u (input of system) and y (output of system) are the two input variables of a basic Kalman filter and estimated state x and estimated y are the two output variables, as illustrated in Figure 4.25.

The equations of state estimation of the basic discrete Kalman filter may be expressed as follows:

1. *Prediction of state:*

$$\bar{x}_n = A\hat{x}_{n-1} + Bu_{n-1} \tag{4.4}$$

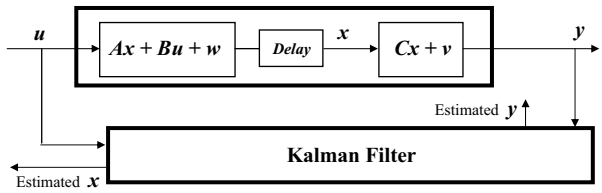


Figure 4.25 A system model and a basic Kalman Filter.

Prediction of error covariance matrix:

$$\bar{P}_n = A\hat{P}_{n-1}A^T + Q \tag{4.5}$$

2. *Computation of Kalman filter gain:*

$$K_n = \bar{P}_n C^T (C\bar{P}_n C^T + R)^{-1} \tag{4.6}$$

State estimation:

$$\hat{x}_n = \bar{x}_n + K_n(y_n - C\bar{x}_n) \tag{4.7}$$

Update of the error covariance matrix:

$$\hat{P}_n = \bar{P}_n - K_n C\bar{P}_n \tag{4.8}$$

The recursive calculation process of the basic Kalman filter shown in Figure 4.26 is useful for understanding the Kalman filter at work.

The definitions of symbols in the Kalman filter are listed in Table 4.10.

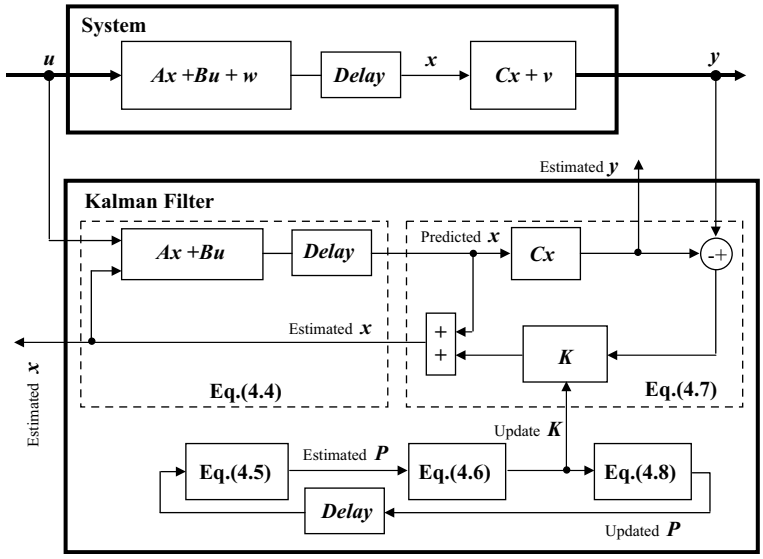


Figure 4.26 Kalman filter calculations.

Table 4.10 Parameters of the Kalman filter.

Symbol	Definition	Initial Value	Update
A	State transition matrix	Constant matrix	No
B	Input matrix	Constant matrix	No
C	Measurement matrix	Constant matrix	No
K	Kalman gain	N/A	Yes
$\bar{P}$	Predicted error covariance	N/A	Yes
$\hat{P}$	Estimated error covariance	Matrix	Yes
Q	Process noise covariance	Constant matrix	No
R	Measurement noise covariance	Constant matrix	No
u	System input	N/A	Yes
v	Measurement noise	N/A	N/A
w	Process noise	N/A	N/A
x	State	N/A	N/A
$\bar{x}$	Predicted state	N/A	Yes
$\hat{x}$	Estimated state	Matrix	Yes
y	Measurement	N/A	Yes

4.4.2 *Example: Signal Estimation in the Presence of Noise by Kalman Filter*

In this example, measured random DC voltage accompanying various noises is estimated by a Kalman filter in order to demonstrate how to simulate a Kalman filter using MATLAB®/Simulink.

Step 1 Mathematical Model of the Signal Measurement

In processing measurement data, Kalman filter may be used to separate signal from random noises (Kalman, 1960). In this example, it is reasonable to assume that the measured signal is from a linear dynamic system with noises as described in Equation (4.3). If we assume that the measured signal is same as the system state, then the measurement matrix C is equal to 1. The measured data is a one-dimensional array, and the state has no change from step $n-1$ to step n, that is, $x_n = x_{n-1}$. Hence, the state transition matrix A is 1. There is no control input in the system, hence u equals 0. The state-space representation in Equation (4.3) becomes

$$\begin{aligned} x_n &= x_{n-1} + w \\ y_n &= x_n + v \end{aligned} \tag{4.9}$$

Step 2 Modeling the Signal Measurement Process

MATLAB® software delivers a 'Kalman Filter' block in 'Signal Processing Blockset'. The block can simulate a simple Kalman filter action (The MathWorks, Inc., 2008e). In the example, value of DC voltage is arbitrarily selected to be measured and it is accompanied by random measurement noise. Mean of the noise is set as 0 and variance of the noise is set as 0.1, hence the measurement noise variance R is set as 0.1. Since only measurement noise is injected into the DC voltage in this example, the process noise covariance Q is set as 0, exactly. It should be noted that the noise covariance is unknown in practical measurements.

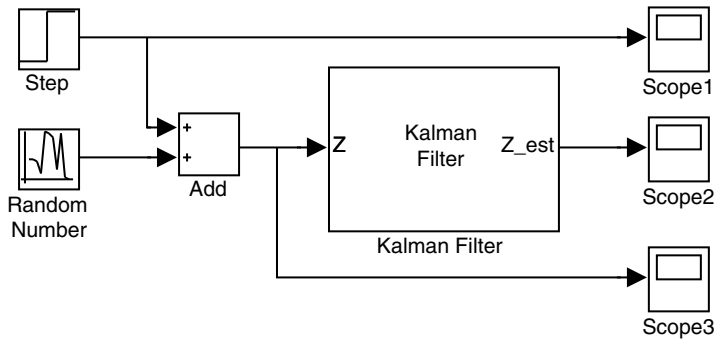


Figure 4.27 Simulink model of Kalman Filter for signal measurement.

The simulation model is constructed in Simulink as shown in Figure 4.27 and the model parameters as listed in Table 4.11.

Table 4.11 Parameters of blocks in Simulink model of Kalman filter.

Name	Function	Parameter	Libraries/Group
Kalman Filter	Estimating measured signal under noise condition	Initial estimated state = 0; Initial estimated error covariance = 1; State transition matrix $A = 1$; Process noise covariance $Q = 0$; Measurement matrix $C = 1$; Measurement noise covariance $R = 0.1$	Signal Processing Blockset/Filtering/ Adaptive Filters
Step	Generating a step DC signal	Step time = 0 Initial value = 0 Final value = 10	Simulink/Sources
Random Number	Generating a random number to simulate measurement noise	Mean = 0; Variance = 0.1; Seed = 0.	Simulink/Sources
Scope 1	Show original signal without noise	N/A	Simulink/Sinks
Scope 2	Show estimated signal by Kalman filter	N/A	Simulink/Sinks
Scope 3	Show measured signal with noise	N/A	Simulink/Sinks
Parameters of simulation	Manage simulation	Fixed-step with step size = 0.02 Start time = 0.0 Stop time =1	N/A

To observe the output of Kalman filter in each iteration, the simulation step is set as 0.02 s and the simulation time is set as 1 s. Thus, the simulation calculation will yield 50 measurement samples and 50 outputs.

The parameter configuration window of the 'Kalman Filter' block is shown in Figure 4.28.

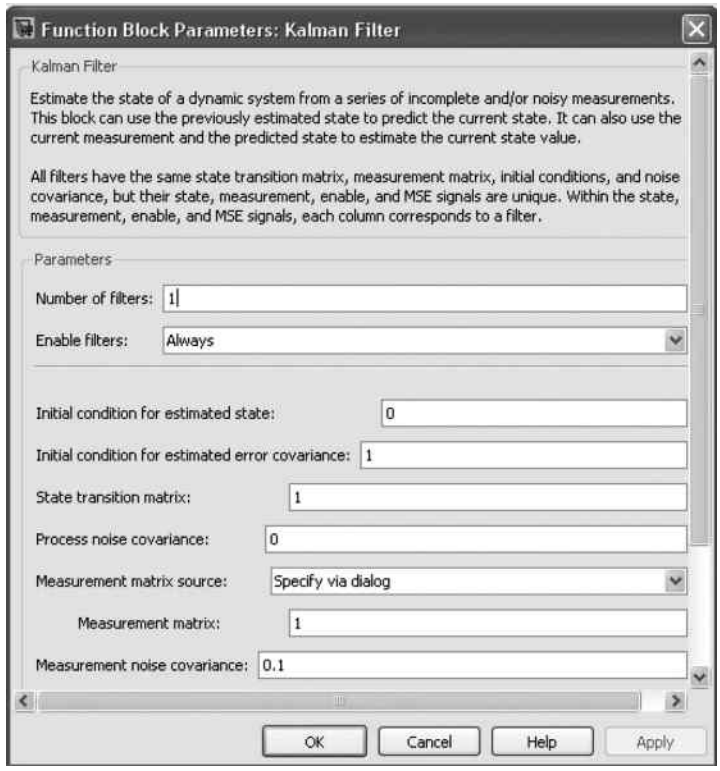


Figure 4.28 Configuring parameters of Kalman filter for the example.

Step 3 Simulation of the Kalman Filter Based Signal Measurement

In a practical application, the measurement noise covariance R may be unidentified. Hence, R may be arbitrarily set. To observe the convergence process of Kalman filter calculation in the presence of various noises, the simulation is performed three times with the following measurement noise covariances which are input into the 'Kalman Filter' blocks in each simulation, separately.

$R = 0.1$ (true value of simulating the measurement system)
$R = 0.001$ (arbitrary value)
$R = 1$ (arbitrary value)

Run the Simulink model shown in Figure 4.27. The simulation results are summarized in Figures 4.29 and 4.30.

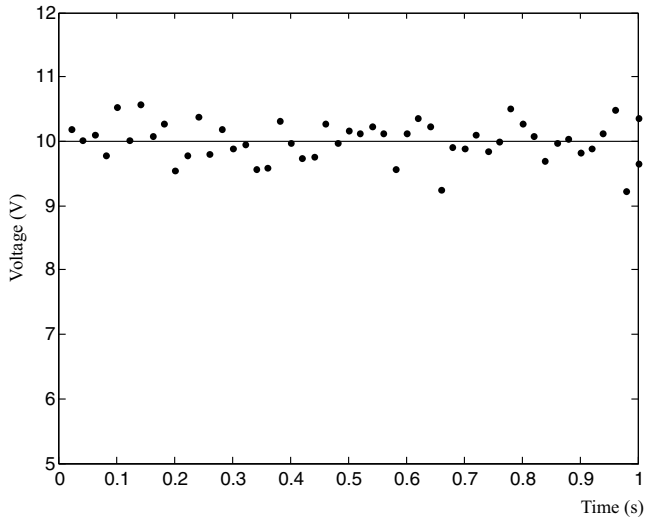


Figure 4.29 Simulation results: original voltage (solid line) and 50 measured samples (dots).

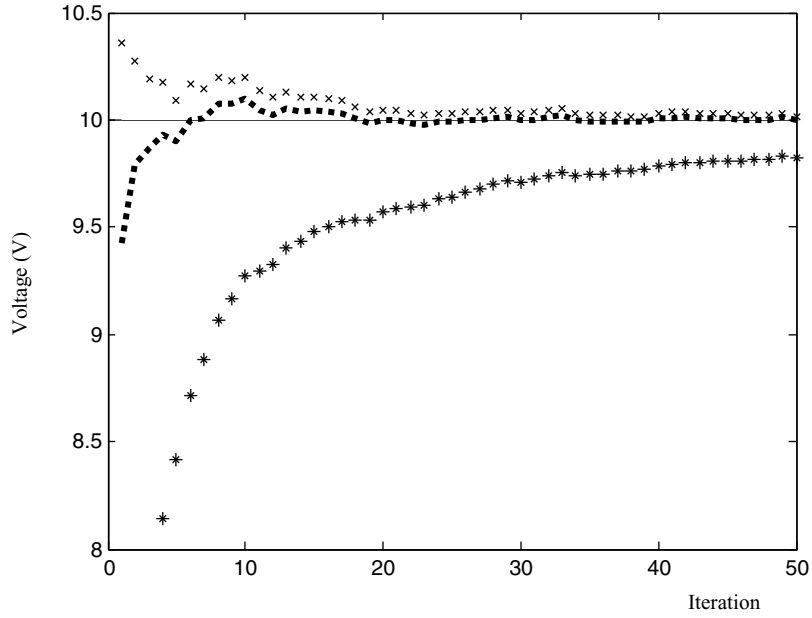


Figure 4.30 Simulation results: original voltage (solid line) and estimated voltages by Kalman filter in 50 iterations, with noise covariance $R = 0.1$ (dots), with $R = 0.001$ (crosses), and with $R = 1$ (asterisks).

The simulation results show that Kalman filter has the ability to separate signal from random noises. However, Kalman filter is sensitive to the measurement noise covariance R. Various R values will result in different convergence results as shown in Figure 4.30. Because noise covariance is usually unknown in practical applications, it will be of interest to investigate further how the noise covariance in the Kalman filter can be tuned in order to yield the best estimation result.

4.5 Getting Started with Genetic Algorithm Simulation

MATLAB® (version R2008b) delivers a 'Genetic Algorithm and Direct Search Toolbox' (The MathWorks, Inc., 2008f). In the toolbox, the 'ga' function may be used to optimize a MATLAB® function. In the toolbox User's Guide, some MATLAB® functions are used as GA's fitness function and they can be optimized by the 'ga' function. In this section, an example is presented to optimize a Simulink model by the 'ga' function.

4.5.1 Genetic Algorithm

Genetic Algorithm (GA) may be employed to optimize a model or a function. The model or a function is called a fitness function in a GA optimization problem.

The basic steps of genetic algorithm are as follows.

Step 1 Start: Generate a random population of n chromosomes which have a proper form for the problem.

Step 2 Run Fitness Function: Evaluate the fitness function using each chromosome in the population.

Step 3 Result Test: Judge the n evaluated results. If conditions are satisfied, then GA stops and outputs the best chromosome of current population.

Step 4 Selection: Select two or more parent chromosomes from a population according to their fitness (the better the fitness of a chromosome, the greater is its chance of being selected).

Step 5 Crossover: With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, the offspring is an exact copy of the parents.

Step 6 Mutation: With a mutation probability mutate new offspring at each locus (position in chromosome).

Step 7 Replace: Place new offspring in the old population to create a new population and use a new generated population for a further run of the algorithm.

Step 8 Loop: Go to **step 2**.

The calculation procedure of basic genetic algorithm may be summarized in Figure 4.31.

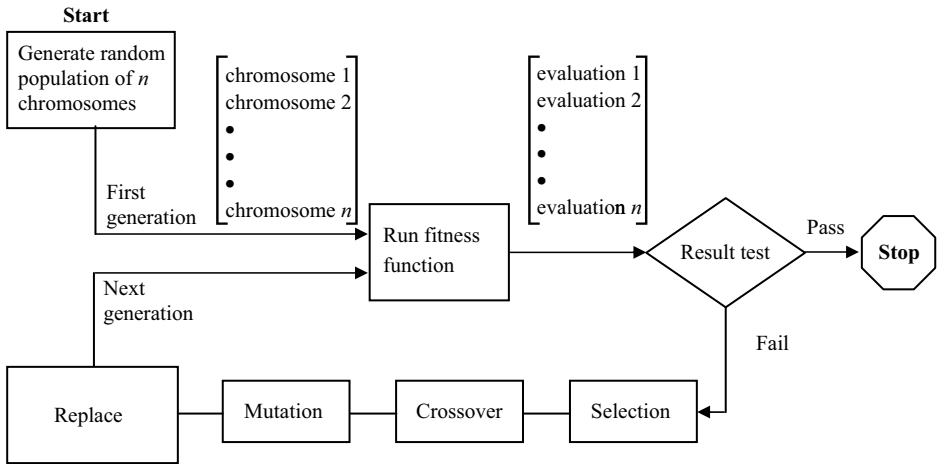


Figure 4.31 Calculation process of basic genetic algorithm.

The parameters of genetic algorithm are listed in Table 4.12.

Table 4.12 Parameters of genetic algorithm.

Name	Function	Input	Output
Start	Initialize *n* random chromosomes	N/A	First generation of population has *n* chromosomes
Run fitness function	Evaluate fitness function	*n* chromosomes	*n* evaluated results
Result test	Judge *n* evaluated results If conditions satisfied, then stop	*n* chromosomes	Pass or Fail
Selection	Select good parent chromosomes according to rank of evaluation	*n* chromosomes	Good chromosomes
Crossover	Create new chromosomes	Parent chromosomes	Child chromosomes
Mutation	Improve chromosomes	Child chromosomes	Improved child chromosomes
Replace	Create next generation of population by replacing old chromosomes	Child chromosomes and parent chromosomes	*n* chromosomes of next generation of population

4.5.2 *Example: Optimizing a Simulink Model by Genetic Algorithm*

This example demonstrates that a PID controller is optimized by Genetic Algorithm function 'ga' of MATLAB® when the control system is a Simulink model. In order to call the Simulink model from MATLAB®, the Simulink model must first be built, and then a MATLAB® programming function is used to call the Simulink model.

Step 1 Building a PID Control Model in Simulink
A Simulink model of PID (proportional–integral–derivative) control system is built as shown in Figure 4.32.

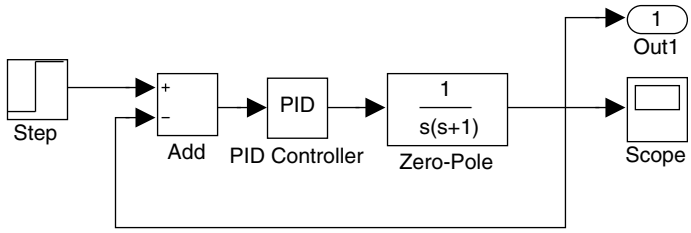


Figure 4.32 A Simulink model of PID control system.

This PID control system will be optimized by genetic algorithm. It is simulated by a 'Zero-pole' block. A 'Step' block is used to create a control reference. An 'Out1' block is used to export calculation results of the Simulink model. The Simulink model is saved with the filename 'PID_controller.mdl'. Parameters of the PID control system are listed in Table 4.13.

Table 4.13 Parameters of Simulink model 'PID_controller.mdl'.

Name	Function	Parameter
Plant	Simulate a plant	$\frac{1}{s(s+1)}$
PID Controller	Feedback control	$K_p = \text{x1}$, $K_i = \text{x2}$, $K_d = \text{x3}$
Step	Create control reference	step time = 0 initial value = 0 and final value = 10
Out1	Export calculation results of the Simulink model	N/A
Simulation Parameters	Manage simulation process	Type of solver is Fixed-step Fixed-step size 0.001 s Start time 0.0 s Stop time 10 s

Parameters of the PID controller consist of three variables, namely, proportional gain $K_p = \text{x1}$, integral gain $K_i = \text{x2}$, and derivative gain $K_d = \text{x3}$, which are set in the 'PID Controller' block as shown in Figure 4.33.

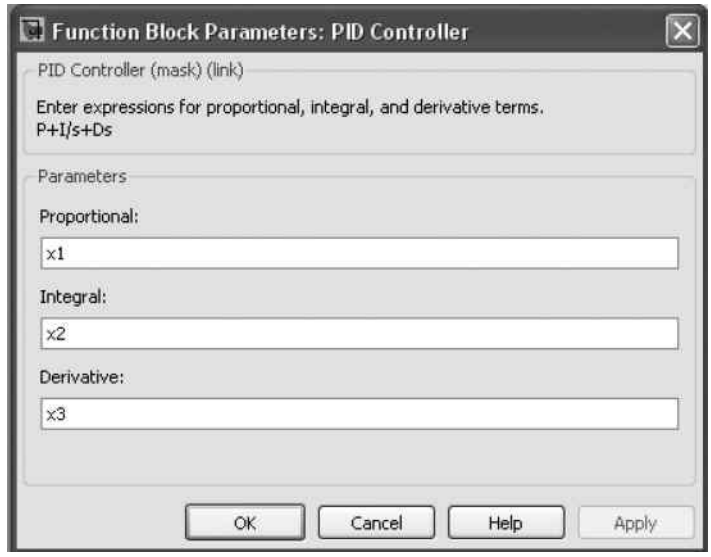


Figure 4.33 Configuration of PID parameters.

Step 2 Programming a Function to Run the Simulink Model

In order to call the Simulink model from MATLAB® platform, a MATLAB® function 'Call_PID.m' is programmed as follows.

```
function s = Call_PID(x)
assignin('base','x1',x(1));
assignin('base','x2',x(2));
assignin('base','x3',x(3));
[tout,xout,yout] = sim('PID_controller',10);
z = yout;
[m,n] = size(z);
V = 0;
R = 10; %command reference
for i = 1:m
V = V + (R-z(i))^2;
end
s = V/m;
end
```

In MATLAB®/Simulink, there two workspaces, one is named as 'caller' which stores variable values of MATLAB® function. Another workspace is named as 'base' which stores

variable values of input and output of Simulink model. Because the 'ga' function is a MATLAB® function, an 'assignin' function is employed to transfer data in the two workspaces. In this way, the variable values may be exchanged between function 'Call_PID.m' and model 'PID_controller.mdl'. Descriptions of the program Call_PID.m are listed in Table 4.14.

Table 4.14 Descriptions of Programming Call_PID.m.

Description	Function	Parameter
function s = Call_PID(x)	Function head	x: input variable of function s: output variable of function Call_PID: function name
assignin	Exchange variable value in 'caller' and 'base' workspaces	x1, x2, x3: variables in 'base' workspace may be called by Simulink model x: array in 'base' workspace used by 'ga' function
sim	Perform a Simulink model	Output: [tout,xout,yout] PID_controller: Simulink model 10: simulation time (secs)
siz	Calculating number of rows and number of columns	input: a matrix output: m is number of rows and n is number of columns of the matrix
'for' cycle sentence and s = V/m sentence	Calculating mean square error of the plant output and command reference	s: output of Call_PID function R = 10: command reference

Step 3 MATLAB® Programming of Genetic Algorithm

In the example, the fitness function is the Call_PID.m which calls Simulink model 'PID_controller.mdl'. The chromosome is the PID controller's parameters [x1, x2, x3]. Parameters of GA performance use default values of the 'ga' function which are listed in Table 4.15. When the performance needs to be improved, the parameters may be changed by 'gaoptimset' function.

Table 4.15 Elements of GA.

Elements of GA	Representation	Reference
Fitness function	Call_PID.m	Function Call_PID.m
Evaluate value	Mean square error of Simulink model output and command reference	Function Call_PID.m and 'PID_controller.mdl'
Chromosome	PID controller's parameters x1, x2, and x3 in Simulink model	Figure 4.33
GA parameters	Default value	See 'Options of GA' later

Enter the following commands to run the simulation on genetic algorithm.

```
clc
clear
options = gaoptimset(@ga);
options = gaoptimset(options,'PlotFcns',{@gaplotbestf},'Display',
'iter');
lb = [0 0 0];
ub = [100 100 100];
[x,fval] = ga(@Call_PID,3,[],[],[],[],lb, ub,[],options);
```

Descriptions of above commands are listed in Table 4.16.

Table 4.16 Descriptions of commands to run genetic algorithm.

Command	Function	Parameter	Ouput
clc	Clear old commands	N/A	N/A
clear	Clear old workspaces	N/A	N/A
gaoptimset()	Get GA parameters	@ga	Default values of options
gaoptimset()	Change GA option, Plot GA's optimizing process	options, 'PlotFcns', {@gaplotbestf}, 'Display', 'iter'	New parameters of options
lb	Lower bound on x	Lower bound of PID	[0 0 0]
ub	Upper bound on x	Upper bound of PID	[100 100 100]
ga()	Perform GA	@Call_PID: Fitness function 3: size of chromosome Options: Plotting GA process	x: chromosome fval: value of the fitness function at x

Options of GA (Default)

```
options =
PopulationType: 'doubleVector'
PopInitRange: [2x1 double]
PopulationSize: 20
EliteCount: 2
CrossoverFraction: 0.8000
ParetoFraction: []
MigrationDirection: 'forward'
MigrationInterval: 20
MigrationFraction: 0.2000
Generations: 100
TimeLimit: Inf
FitnessLimit: -Inf
StallGenLimit: 50
StallTimeLimit: Inf
TolFun: 1.0000e-006
TolCon: 1.0000e-006
InitialPopulation: []
```

```
InitialScores: []
InitialPenalty: 10
PenaltyFactor: 100
PlotInterval: 1
CreationFcn: @gacreationuniform
FitnessScalingFcn: @fitscalingrank
SelectionFcn: @selectionstochunif
CrossoverFcn: @crossoverscattered
MutationFcn: {[1x1 function_handle] [1] [1]}
DistanceMeasureFcn: []
HybridFcn: []
Display: 'final'
PlotFcns: []
OutputFcns: []
Vectorized: 'off'
UseParallel: 'never'
```

Step 4 Optimizing the PID Parameters by Genetic Algorithm

In order to compare the results of GA optimized PID controller, two simulations are first performed with a random set x = [0.0310, 0.6467, 0.0563] and x = [5,5,5] and the simulation results are shown in Figure 4.34.

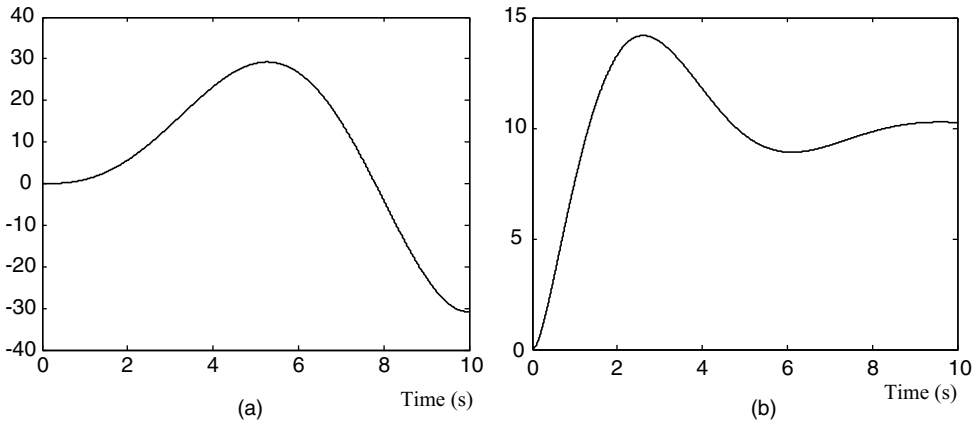


Figure 4.34 Two simulation results of PID control model. (a) With random PID parameter x = [0.0310, 0.6467, 0.0563]; (b) With x = [5,5, 5].

Run the GA command set described in **Step 3**.

```
clc;
clear;
options = gaoptimset(@ga);
options = gaoptimset(options,'PlotFcns',{@gaplotbestf},'Display',
'iter');
lb = [0 0 0];
ub = [100 100 100];
[x,fval] = ga(@Call_PID,3,[],[],[],[],lb, ub,[],options);
```

Because initial chromosomes in the genetic algorithm are randomly created, the program may need to be run several times in order to obtain satisfactory results. Figures 4.35 and 4.36 show that the best output is 1.997 with the final PID parameters x = [25.20, 0.045, 4.25].

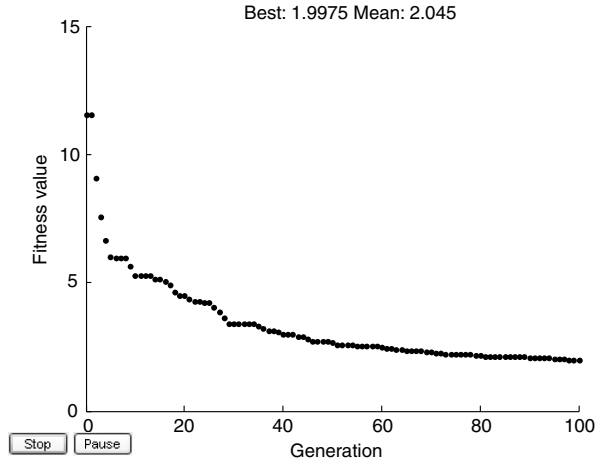


Figure 4.35 Best evaluated outputs of the fitness function at each generation.

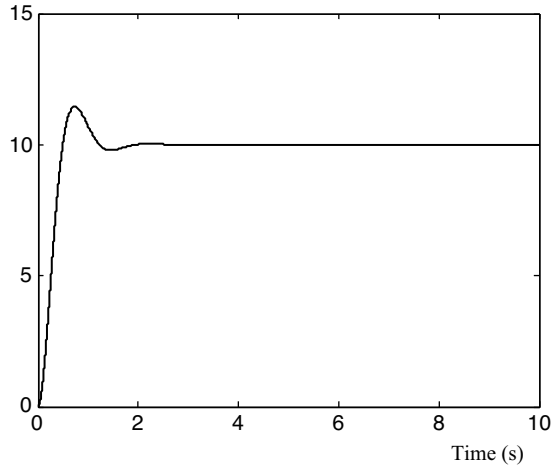


Figure 4.36 The final simulation output of Simulink model 'PID_controller.mdl' with the optimized PID parameters x = [25.20, 0.045, 4.25] and the mean square error 1.997.

Step 5 Improving the GA Performance Results

In order to improve the performance of the GA, several GA option parameters may be adjusted with the following commands.

1. Increase population size to 40 by command by entering

```
options = gaoptimset(options,'PopulationSize',40);
```

2. Increase better individuals in old population to 10 that are guaranteed to survive to the next generation by entering

```
options = gaoptimset(options, 'EliteCount',10);
```

3. Decrease crossover fraction to 0.6 by entering

```
options = gaoptimset(options, 'CrossoverFraction',0.6);
```

4. Increase generations to 140 by entering

```
options = gaoptimset(options, 'Generations',140);
```

Insert the above four commands into GA run commands described in **Step 3**. The GA run commands become:

```
clc
clear
options = gaoptimset(@ga);
options = gaoptimset(options,'PlotFcns',{@gaplotbestf},'Display',
'iter');
options = gaoptimset(options,'PopulationSize',40);
options = gaoptimset(options, 'EliteCount',10);
options = gaoptimset(options, 'CrossoverFraction',0.6);
options = gaoptimset(options, 'Generations',140);
```

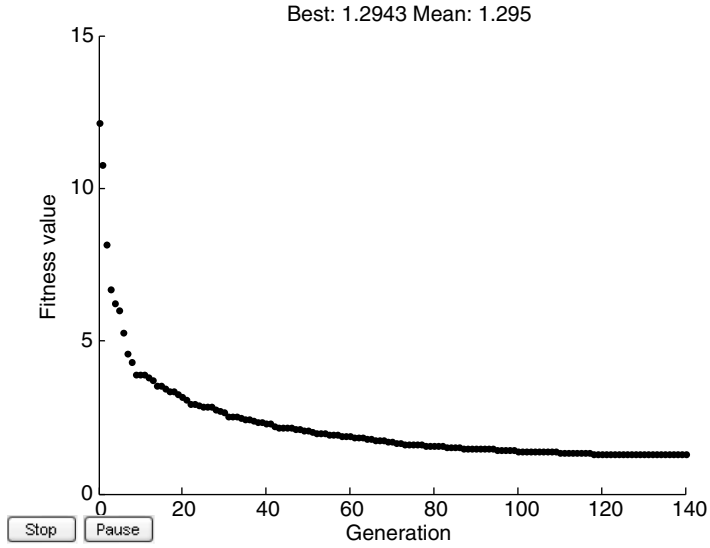


Figure 4.37 Best and mean evaluated outputs of the fitness function at each generation.

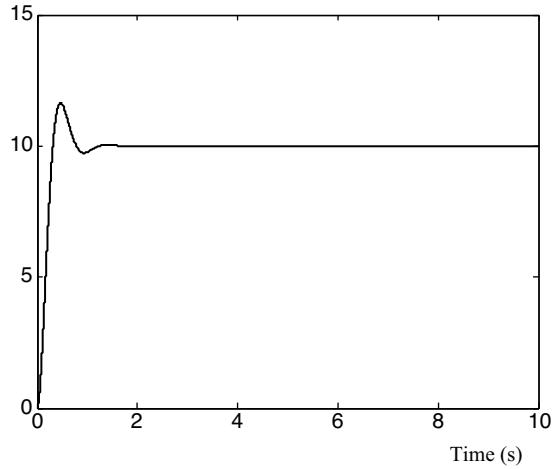


Figure 4.38 Simulation output of Simulink model 'PID_controller.mdl' with the final optimized PID parameters x = [59.99, 0.03, 6.73] and mean square error 1.294.

```
options = gaoptimset(options,'MutationFcn',@mutationadaptfeasible);
lb = [0;0;0];
ub = [60;10;10];
[x,fval] = ga(@Call_PID,3,[],[],[],[],lb, ub,[],options);
```

After running the above revised set of GA commands, better chromosomes (PID parameters) are obtained as shown in Figure 4.37 and the evaluated output of the Simulink model 'PID_controller.mdl' is smaller, as shown in Figure 4.38.

The best output is 1.294 and the final PID parameters are x = [59.99, 0.03, 6.73].

By comparing Figure 4.37 with Figures 4.35 and 4.38 with Figure 4.36, we conclude that the GA optimization result can be improved by adjusting the GA performance parameters. The mean square error is decreased from 1.997 to 1.294.

4.6 Summary

The example of fuzzy PI controller illustrates how to start using 'Fuzzy Logic Toolbox' of MATLAB®/Simulink. The example of neural network based Park's transformation demonstrates using a neural network of parallel calculation structure to implement multi-output nonlinear functions. In the Kalman filter example, linear measured signal is separated from random noises by the 'Kalman Filter' block of Simulink. It is also demonstrated that the performance of the Kalman filter is sensitive to the measurement noise covariance *R*. The example of genetic algorithm optimized PID controller illustrates how to get started using the 'ga' function delivered by MATLAB® software. After working through these four examples, the readers should have acquired the basic techniques of intelligent control simulation.

References

Kalman, R.E. (1960) A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, **82**, 34–45.

The MathWorks, Inc. (2008a) MATLAB® Getting Started Guide. The MathWorks, Inc.

The MathWorks, Inc. (2008b) Simulink Getting Started Guide. The MathWorks, Inc.

The MathWorks, Inc. (2008c) Fuzzy Logic Toolbox User's Guide. The MathWorks, Inc.

The MathWorks, Inc. (2008d) Neural Network Toolbox User's Guide. The MathWorks, Inc.

The MathWorks, Inc. (2008e) Signal Processing Blockset Reference. The MathWorks, Inc.

The MathWorks, Inc. (2008f) Genetic Algorithm and Direct Search Toolbox User's Guide. The MathWorks, Inc.

Sousa, G.C.D. and Bose, B.K. (1995) Fuzzy logic application to power electronics and drives – an overview. Proceedings of the 1995 IEEE IECON 21st International Conference on Industrial Electronics, Control, and Instrumentation, Orlando, FLA, pp. 57–62.

5

Expert-System-based Acceleration Control[1]

5.1 Introduction

Conventional vector control of the induction motor includes field-oriented control (FOC) and direct self control (DSC). In recent years, FOC and DSC have been applied to inverter-fed induction motor drives for improving the transient response (Bose, 1986; Novotny and Lipo, 1996). Although the implementation of both methods has largely been successful, they both suffer from sensitivity to parameter variations and error accumulation when evaluating the definite integrals (Shi *et al.*, 1999). In both methods, the control must be continuous and the calculation must begin from an initial state. If the control time is long, degradation in the steady-state and transient responses will result due to drift in parameter values and excessive error accumulation.

An expert system is a computer program that is designed to emulate a human's skills in a specific problem domain (George and William, 1989). As the forerunner among all the AI techniques, expert systems have been researched since the early 1960s and have now become the most important branch of artificial intelligence. Since then, expert systems have found wide applications in many areas. If an expert system is used as part of a feedback controller to emulate the expertise of a human in performing control of plant, it is called 'expert-system control' (Passino and Lunardhi, 1996a). Expert-system control was originally proposed by Åström and Anton (1984) (Åström and Anton, 1984). Due to the complexity existing in many real-world control problems, not all of them can be well represented, solved and implemented

[1] (a) Portions reprinted from K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A new acceleration control scheme for an inverter-fed induction motor," *Electric Power Components and Systems*, **27**(5), 527–554, © 1999, by permission of Taylor & Francis Ltd, http://www.tandf.co.uk/journals
(b) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.
(c) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," Proceedings of IEEE International Electric Machines and Drives Conference (IEMDC '99), pp. 613–615, Seattle, Washington, U.S.A. © 1999 IEEE.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

by the traditional mathematical methodologies and tools. Consequently, the applications of human-like expert system in real-time dynamic control have been an attractive research area. The core parts of expert-system control are the knowledge-base and the inference engine. The knowledge base consists of facts and rules that characterize strategies on how to control the plant. The inference engine is designed to emulate the control expert's decision-making process. In addition, an expert-system controller may have a user interface to interpret the control process and to modify the knowledge base by a user. In this chapter, the expert-system principle is employed to control the rotor acceleration of an induction motor, the main objective being to overcome the drawbacks of common vector control schemes. Based on the relationship between stator voltage vector and rotor acceleration, a control method with voltage vector comparison and voltage vector retaining is proposed to control the rotor acceleration. This method uses a trial-and-error strategy to determine the best of seven voltage vectors in every interval of the control process, which is then selected and retained. To decrease the number of voltage vectors to be compared, the production knowledge base is modified by tracking the angle of the stator current vector. After using the heuristic knowledge, the number of voltage vectors compared is decreased to two, and the influence of sub-optimal voltage vectors is reduced to a minimum. Fourteen rules represent the rotor acceleration control knowledge and the inference engine based on a production system (Buchanan and Shortliffe, 1984; Hayes-Roth, 1985) processes the rules in order to arrive at the desired control goals.

5.2 Relationship between the Stator Voltage Vector and Rotor Acceleration

The usual vector controllers rely on flux calculations. Since flux is calculated from an integral of input electrical energy, the controller cannot thoroughly eliminate the flux accumulation error. However, by using rotor acceleration control instead of electromagnetic torque control, flux calculations will no longer be needed. In practice, most of the inverters in use can produce only seven discrete space vector values of actuating variables. Usually none of these is exactly equal to the desired instantaneous value of the space vector. In the proposed rotor acceleration controller, one voltage vector is selected in every period so as to decrease the error between the actual rotor acceleration and the acceleration command. Because there is no direct relationship between the rotor acceleration and the stator voltage vector, the optimum voltage vector has to be selected by comparing the incremental acceleration produced by every voltage vector. The formulations of incremental acceleration and stator voltage vector are deduced as follows.

The dynamic equation of the mechanical system may be expressed as (Krause, Wasynczuk, and Sudhoff, 1995):

$$\gamma \frac{d\omega_o}{dt} = T - T_L. \tag{5.1}$$

where γ is the normalized mechanical time constant, T_L is the load torque, and T is the electromagnetic torque given by:

$$T = \lambda_\mu^s \times i_s^s. \tag{5.2}$$

From Equations (5.1) and (5.2), the rotor acceleration is:

$$\frac{d\omega_o}{dt} = \frac{1}{\gamma}\left(\lambda_\mu^s \times i_s^s - T_L\right). \tag{5.3}$$

The stator flux can be expressed as:

$$\lambda_\mu^s(t_0) = \int_0^{t_0} \left(V_s^{(k)} - R_s i_s^s\right) dt \tag{5.4}$$

or

$$d\lambda_\mu^s = \left(V_s^{(k)} - R_s i_s^s\right) dt. \tag{5.5}$$

When $t > t_0$, the stator voltage vector $V_s^{(k)}$ determines the increment of the stator flux, $d\lambda_\mu^s$, which is shown in Figure 5.1.

If the rotor acceleration is denoted by a, then Equation (5.1) becomes

$$a = \frac{d\omega_o}{dt} = \frac{1}{\gamma}(T - T_L) \tag{5.6}$$

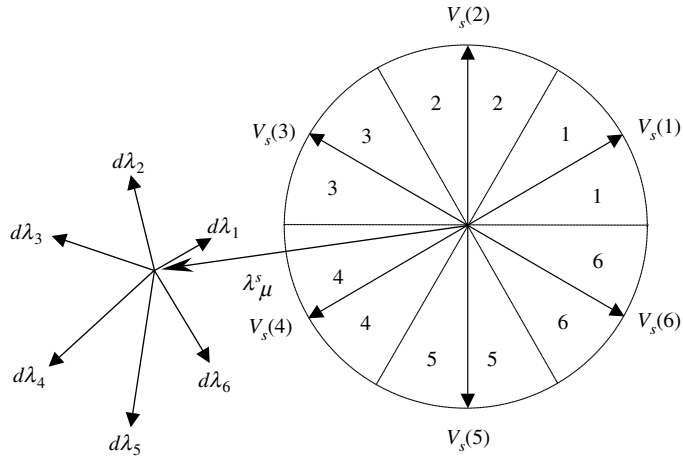


Figure 5.1 Flux increment of induction motor. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

The differential rotor acceleration is

$$da = \frac{1}{\gamma}[dT - dT_L] \tag{5.7}$$

From Equation (5.2),

$$dT = d\lambda_\mu^s \times i_s^s + \lambda_\mu^s \times di_s^s. \tag{5.8}$$

Substituting Equations (5.5), (5.8) into (5.7),

$$da(k) = \frac{1}{\gamma}\left[\left(V_s^{(k)} - R_s i_s^s\right)dt \times i_s^s + \lambda_\mu^s \times di_s^s - dT_L\right]. \tag{5.9}$$

Because $i_s^s \times i_s^s = 0$, Equation (5.9) becomes

$$da(k) = \frac{1}{\gamma}\left[\left(V_s^{(k)} \times i_s^s\right)dt + \lambda_\mu^s \times di_s^s - dT_L\right] \tag{5.10}$$

where k ($mod(k) = 7$) denotes one of the seven voltage vectors.

The incremental acceleration of the rotor is, from Equation (5.10),

$$\Delta a(k) = \frac{1}{\gamma}\left[\left(V_s^{(k)} \times i_s^s\right)\Delta t + \lambda_\mu^s \times \Delta i_s^s - \Delta T_L\right] \tag{5.11}$$

$$\Delta a(k) = \frac{1}{\gamma}\left[\Delta t\, R^o |V_s^{(k)}||i_s^s| \sin \vartheta_i + \lambda_\mu^s \times \Delta i_s^s - \Delta T_L\right] \tag{5.12}$$

where R^o is a unit vector.

Equation (5.12) explains that the stator voltage vector $V_s^{(k)}$ determines the incremental accelerations of the rotor. When $t > t_0$, the incremental accelerations that result from the six stator voltage vectors are shown in Figure 5.2.

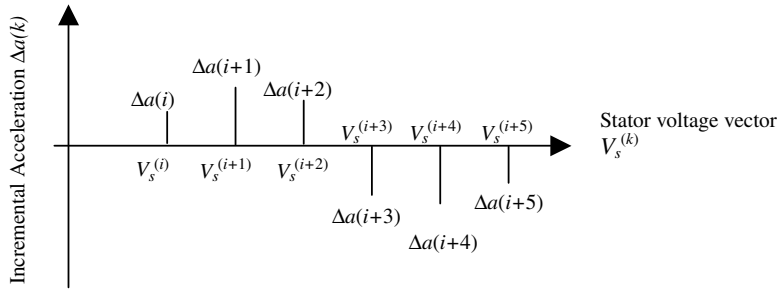


Figure 5.2 Rotor acceleration increments of six stator voltage vectors. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

Two important results are obtained from Equation (5.12).

1. Incremental acceleration of the rotor is determined by the stator voltage vector.
2. If $\Delta a(j) < 0$, there is at least one number m such that $\Delta a(j + m) > 0$. If $\Delta a(j) > 0$, there is at least one number n such that $\Delta a(j + n) < 0$.

According to these two results, a controller may be designed using a voltage vector comparison and retaining method.

5.3 Analysis of Motor Acceleration of the Rotor

An example is given to illustrate rotor acceleration controlled by stator voltage vectors. If two stator voltage vectors are supplied to the motor in succession for $t > t_0$, they will produce two different accelerations of rotor as shown in Figure 5.3.

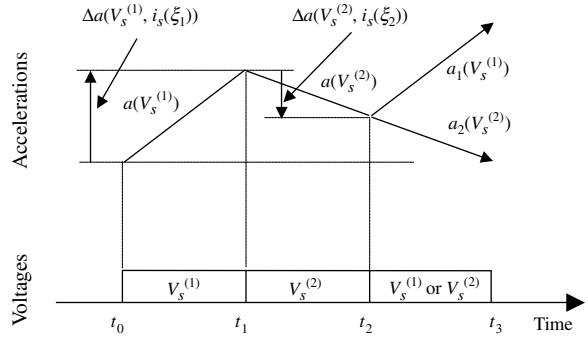


Figure 5.3 Effect of applied voltage vector on rotor acceleration.

During $t_2 < t < t_3$, selecting and retaining the voltage vector $V_s^{(1)}$ that results in a larger incremental acceleration of the rotor will produce a larger acceleration. This may be proved as follows.

$$\text{Let } \Delta t = t_1 - t_0 = t_2 - t_1 = t_3 - t_2.$$

Then from Equation (5.11), the incremental acceleration of the rotor may be written as,

$$\Delta a\left(V_s^{(k)}, i_s^s(\xi)\right) = \frac{1}{\gamma}\left[\Delta t\left(V_s^{(k)}\right) \times i_s^s(\xi) + \lambda_\mu^s \times \Delta i_s^s - \Delta T_L\right] \tag{5.13}$$

where $t_0 < \xi < t_0 + \Delta t$.

The rotor accelerations at instants t_1, t_2 and t_3 are as follows:

$$a(t_1) = a(t_0) + \Delta a\left(V_s^{(1)}, i_s^s(\xi_1)\right),\ t_0 < \xi_1 < t_1 \tag{5.14}$$

$$a(t_2) = a(t_1) + \Delta a\left(V_s^{(2)}, i_s^s(\xi_2)\right),\ t_1 < \xi_2 < t_2 \tag{5.15}$$

$$a_1(t_3) = a(t_2) + \Delta a\left(V_s^{(1)}, i_s^s(\xi_3)\right),\ t_2 < \xi_3 < t_3 \tag{5.16}$$

$$a_2(t_3) = a(t_2) + \Delta a\left(V_s^{(2)}, i_s^s(\xi_4)\right),\ t_2 < \xi_4 < t_3. \tag{5.17}$$

Suppose $i_s^s(\xi_3) \approx i_s^s(\xi_1)$ and $i_s^s(\xi_4) \approx i_s^s(\xi_2)$, then

$$a_1(t_3) = a(t_2) + \Delta a\left(V_s^{(1)}, i_s^s(\xi_3)\right) \approx a(t_2) + \Delta a\left(V_s^{(1)}, i_s^s(\xi_1)\right) \tag{5.18}$$

$$a_2(t_3) = a(t_2) + \Delta a\left(V_s^{(2)}, i_s^s(\xi_4)\right) \approx a(t_2) + \Delta a\left(V_s^{(2)}, i_s^s(\xi_2)\right). \tag{5.19}$$

If $\Delta a\left(V_s^{(1)}, i_s^s(\xi_1)\right) > \Delta a\left(V_s^{(2)}, i_s^s(\xi_2)\right)$, then $a_1(t_3) > a_2(t_3)$. In this case, if we wish to increase the acceleration of the rotor during the subsequent time interval, the voltage vector $V_s^{(1)}$ should be retained and the voltage vector $V_s^{(2)}$ should be discarded.

To find the optimum voltage vector at every period, a control strategy of voltage vector comparison and voltage vector retaining is proposed.

5.4 Control Strategy of Voltage Vector Comparison and Voltage Vector Retaining

In the proposed control method, the time is divided into many small intervals each consisting of a voltage vector comparison period and a voltage vector retaining period. In the comparison period, several voltage vectors are supplied to the induction motor in proper order, and the incremental acceleration of each voltage vector is recorded. At the end of the comparison period, the optimum voltage vector that produces a larger incremental acceleration is selected for the retaining stage. In the latter stage, the controller retains this optimum voltage vector to the motor. If the rotor acceleration is above or below a certain threshold during the voltage-retaining period, a zero voltage vector is supplied. A cycle of the control process for the induction motor is illustrated in Figure 5.4.

Three problems need to be solved before the proposed method can be applied directly to control an induction motor. The first problem is how to assign the comparison time and retaining time, as this will affect the results of control for the induction motor. The second problem is how to select heuristically the voltage vectors to be compared, because too many voltage vector comparisons will degrade the performance of the induction motor. The third problem is how to compare the rotor acceleration, because the voltage vector of maximum rotor acceleration may not be the optimum voltage vector. When the rotor accelerations produced by different voltage vectors are compared, the maintenance of the current amplitude should also be considered.

Assign the comparison time and retaining time. To determine the appropriate cycle time, a heuristic approach may be used. Because there are six nonzero voltage vectors supplied during a revolution and the stator voltage vector rotation is faster than the rotor by the asynchronous principle, the cycle time should be shorter than the time for the rotor to rotate through 1/6 of a revolution. The cycle time should thus be shorter when the speed of rotor is faster. By this heuristic, the cycle time may be fuzzily determined and adjusted by measuring the rotor speed.

Strategy of selecting the voltage vectors to be compared. In order to decrease the number of voltage vectors to be compared, a method of selecting these voltage vectors heuristically is used by tracking the angle of stator current vector. Equation (5.3) may be written as

$$\frac{d\omega_o}{dt} = \frac{1}{\gamma}\left(|\lambda_\mu^s||i_s^s|\sin\varphi_i - T_L\right). \tag{5.20}$$

$$\text{When } \frac{d\omega_o}{dt} = 0,\ \varphi_i\Big|_{\frac{d\omega}{dt}=0} = \arcsin\left\{\frac{T_L}{|\lambda_\mu^s||i_s^s|}\right\} = \varphi_0 \tag{5.21}$$

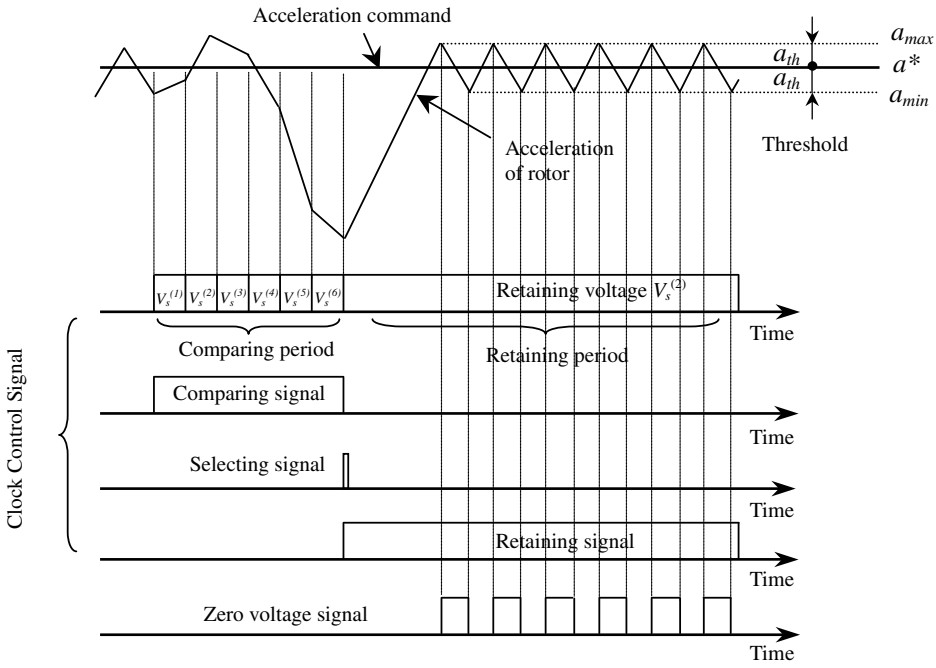


Figure 5.4 A cycle of voltage comparison and voltage retaining. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

$$\text{When } \frac{d\omega_o}{dt} > 0, \varphi_0 < \varphi_i < \pi - \varphi_0 \tag{5.22}$$

$$\text{When } \frac{d\omega_o}{dt} \leq 0, -\pi - \varphi_0 \leq \varphi_i \leq \varphi_0 \tag{5.23}$$

After the actual acceleration of the rotor has been known, the angle range between the stator current vector and the flux vector may be determined from Equations (5.22) and (5.23). Once the actual angle of the stator current vector is detected, the flux angle range may be estimated. Because the estimated angle range of the flux vector should not exceed π rad, the incremental acceleration of the rotor can be determined by comparing only two voltage vectors.

An example of voltage vector comparison based on the position of the stator current vector is shown in Figure 5.5. When the rotor acceleration is larger than zero and the stator current vector is in area 5, the flux λ_μ^s may be in area 2, 3, or 4 according to Equation (5.22). According to a predictive (optimal) principle of current control (Nabae, Ogasawara, and Akagi, 1986), the incremental current Δi_1, Δi_3, Δi_4, Δi_5 and Δi_6 are as shown in Figure 5.5.

From Equation (5.11) and Figure 5.5, when the rotor acceleration is larger than zero and the stator current vector is in area 5, six rules may be summarized as follows:

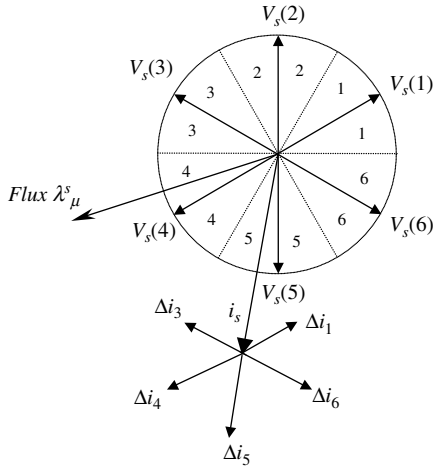


Figure 5.5 An example of selecting the voltages to be compared. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

1. If $V_s(5)$ is chosen, then the current amplitude and the rotor acceleration will be increased.
2. If $V_s(6)$ is chosen, then the current amplitude will be maintained and the rotor acceleration will be increased.
3. If $V_s(1)$ is chosen, then the current amplitude will be decreased and the rotor acceleration will be increased.
4. If $V_s(3)$ is chosen, then the current amplitude will be decreased and the rotor acceleration will be decreased.
5. If $V_s(4)$ is chosen, then the current amplitude will be maintained and the rotor acceleration will be decreased.
6. If $V_s(0)$ is chosen, then the current amplitude and the rotor acceleration will be decreased.

The selection pattern of compared voltage vectors can be derived from the above six rules. When the rotor acceleration command is larger than zero (a counterclockwise acceleration command), the supply current should be maintained at a larger value, because a positive torque is needed to accelerate the rotor. Hence, the compared voltage vectors should be $V_s(5)$ and $V_s(6)$ and the voltage vectors $V_s(1)$, $V_s(2)$, $V_s(3)$ and $V_s(4)$ are discarded. When the rotor acceleration command is less than zero (a clockwise deceleration command), magnitude of the supply current should be maintained at a larger value, because a negative torque is needed to decelerate the rotor. Hence, the compared voltage vectors should be $V_s(5)$ and $V_s(4)$ and the voltage vectors $V_s(1)$, $V_s(2)$, $V_s(3)$ and $V_s(6)$ are discarded. When the rotor acceleration command is equal to zero, the supply current should be maintained at a smaller value. Hence, the compared voltage vectors should be $V_s(5)$ and $V_s(1)$ when rotor speed command is larger than zero (a counterclockwise rotation command); or should be $V_s(5)$ and $V_s(3)$ when rotor speed command is less than zero (a clockwise rotation command); and the voltage vectors $V_s(6)$, $V_s(2)$, and $V_s(4)$ are discarded. These selection patterns are expressed in Table 5.1.

Table 5.1 Selection of the vectors to be compared according to rotor acceleration and speed commands.

a^*	Voltage vectors compared when stator current vector is in area $n=5$						Retained voltage vector
ω_o^*	$V_s^{(1)}$	$V_s^{(2)}$	$V_s^{(3)}$	$V_s^{(4)}$	$V_s^{(5)}$	$V_s^{(6)}$	$V_s^{(0)}$
$a^* \gg 0$					Yes	Yes	Yes
$a^* \ll 0$				Yes	Yes		Yes
$a^* \approx 0$ $\omega_o^* > 0$	Yes				Yes		Yes
$a^* \approx 0$ $\omega_o^* < 0$			Yes		Yes		Yes

(Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

When stator current vector is in any area n, the general patterns of selecting the voltage vectors to be compared can be summarized as follows:

1. If $a^* \gg 0$, and stator current vector is in area n, the compared voltage vectors are $V_s^{(n)}$ and $V_s^{(n+1)}$.
2. If $a^* \ll 0$, and stator current vector is in area n, the compared voltage vectors are $V_s^{(n)}$ and $V_s^{(n-1)}$.
3. If $a^* \approx 0, \omega_o^* > 0$, and stator current vector is in area n, the compared voltage vectors are $V_s^{(n)}$ and $V_s^{(n+2)}$.
4. If $a^* \approx 0, \omega_o^* < 0$, and stator current vector is in area n, the compared voltage vectors are $V_s^{(n)}$ and $V_s^{(n-2)}$.

Comparison strategy of rotor acceleration: According to the above rules, the amplitude increment of stator current produced by $V_s^{(n)}$ is larger than $V_s^{(n+1)}$, $V_s^{(n+2)}$ or $V_s^{(n-1)}$ when the stator current vector is in area n. In order to maintain the amplitude of stator current, once the rotor acceleration produced by $V_s^{(n+1)}$ or $V_s^{(n+2)}$ is larger than zero (for the case $a^* \gg 0$, or, $a^* \approx 0$ and $\omega_o^* > 0$), or, $V_s^{(n-1)}$ or $V_s^{(n-2)}$ is less than zero (for the case $a^* \ll 0$, or, $a^* \approx 0$ and $\omega_o^* < 0$), they should preferentially be selected as the retained voltage vector rather than $V_s^{(n)}$. The method of the voltage vectors comparison can be expressed as follows.

$$\Delta a\left(V_s^{(R)}\right) = \max\left\{\Delta a\left(V_s^{(n)}\right), \eta\Delta a\left(V_s^{(n+1)}\right)\right\} \quad \text{when } a^* \gg 0$$

$$\Delta a\left(V_s^{(R)}\right) = \min\left\{\Delta a\left(V_s^{(n)}\right), \eta\Delta a\left(V_s^{(n-1)}\right)\right\} \quad \text{when } a^* \ll 0$$

$$\Delta a\left(V_s^{(R)}\right) = \max\left\{\Delta a\left(V_s^{(n)}\right), \eta\Delta a\left(V_s^{(n+2)}\right)\right\} \quad \text{when } a^* \approx 0 \text{ and } \omega_o^* > 0$$

$$\Delta a\left(V_s^{(R)}\right) = \max\left\{\Delta a\left(V_s^{(n)}\right), \eta\Delta a\left(V_s^{(n-2)}\right)\right\} \quad \text{when } a^* \approx 0 \text{ and } \omega_o^* < 0$$

η is a preferential parameter that is larger than 1 to implement the preferential selection. A satisfactory value of η is 500 from the results of computer simulation. $V_s^{(R)}$ is the optimum voltage vector that will be supplied in the retaining stage of the control process.

5.5 Expert-System Control for Induction Motor

The functions of an expert-system controller are to interpret plant outputs and reference inputs, to reason about alternative control strategies, and to generate inputs in order to improve the performance of the closed-loop system. An expert-system controller functionally works as follows (Åström and Björn, 1995; Passino and Lunardhi, 1996a):

1. Through the input interface, the controller receives information that is numerical (quantitative) or linguistic (qualitative). Through the output interface, the controller sends electrical signals to the plant.
2. Based on the real-time information received, a knowledge base and an inference engine provide the decision making to control the system under an uncertain environment, such as changes of noise, parameter, load, or power supply. The decision making can be achieved just by tuning the parameters of the controller, changing the algorithm, or modifying the control structure.
3. Monitoring performance faults, consulting with the control expert, and modifying the knowledge base via a user interface. The knowledge-base modification based on consultation with the control expert can make the controller more flexible and adaptive than the prototype controller.

The control system proposed consists of the expert-system controller, inverter, and induction motor as shown in Figure 5.6.

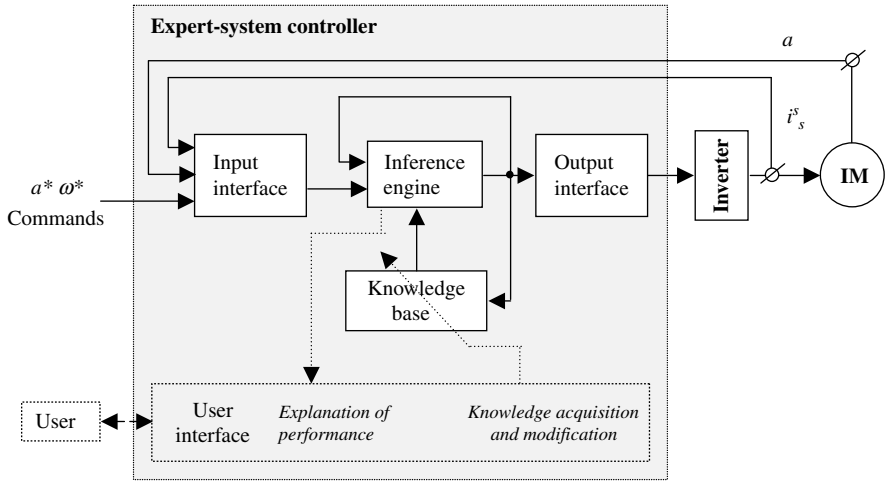


Figure 5.6 Induction motor control system with expert-system controller.

The expert-system controller consists of a knowledge base, an inference engine, an input interface, an output interface, and a user interface. The knowledge base includes a database and a rulebase. The data can be separated into facts and goals. Examples of facts are statements such as 'acceleration is zero,' and 'in retaining period.' An example of goals is 'rotor acceleration is between super-value and under-value.' New facts can also be created by the rules. The rulebase contains production rules of the type: '**if** premise **then** conclusion (action).' The premise is the fact or the goal of the database. The conclusion results in an action and may add a new fact to the database or modify an existing fact. In this way, the knowledge base can choose the most appropriate strategy to control the induction motor. In collecting command input (rotor acceleration command), induction motor output (rotor acceleration), and the inference engine output (working state of controller), the inference engine is designed to emulate the control expert's decision-making process to operate the rules to arrive at the conclusion or to satisfy the goals. The input interface implements the numerical and linguistic coding of electrical signals. The output interface implements the transformation from the numerical values and linguistic commands to electrical signals. In order to improve the performance of the prototype controller, the user interface may be designed to monitor the performance faults, to consult the control expert, and to modify the knowledge base of the controller.

The task of knowledge representation is to capture the essential features of a problem domain and make that information accessible to a problem-solving procedure. The rotor acceleration control knowledge can be represented as 14 rules by the logical language, '**if** premise **then** conclusion (action).' At the beginning of the comparison period, the space region n of stator current vector is detected. Then, depending on the rotor acceleration command, two voltage vectors are supplied to the induction motor in succession, and the corresponding rotor accelerations are obtained from the speed sensor. By comparing the two accelerations, the optimum voltage vector is obtained and is held constant in the next period (retaining period).

In the comparison period, the control knowledge is expressed as 10 rules as follows.

1. **If** in comparison period (denote as E) and comparison voltage vectors have not been input (denote as ¬H),
   then to detect the region of stator current space vector (denote as $n = i$).
2. **If** in comparison period (E), rotor acceleration command is much larger than zero (A_1), and comparison voltage vectors have not been input (¬H),
   then supply $V_s(n)$ and $V_s(n+1)$ to induction motor in succession (X_1).
3. **If** in comparison period (E), rotor acceleration command is near zero (A_2), and comparison voltage vectors have not been input (¬H),
   then supply $V_s(n)$ and $V_s(n+2)$ to induction motor in succession (X_2).
4. **If** in comparison period (E), rotor acceleration command is much less than zero(A_3), and comparison voltage vectors have not been input (¬H),
   then supply $V_s(n)$ and $V_s(n-1)$ to induction motor in succession (X_3).

5a. **If** in comparison period (E), rotor acceleration command is much larger than zero (A_1), comparison voltage vectors have been input (H), and $\Delta a(V_s(n)) > \eta \Delta a(V_s(n+1))$ (B_1),
then $V_s(n)$ is supplied to induction motor and is retained ($k = n$, $V_s(k)$, counter $t = 0$).

5b. **If** in comparison period (E), rotor acceleration command is near zero (A_2), comparison voltage vectors have been input (H), and $\Delta a(V_s(n)) > \eta \Delta a(V_s(n+2))$ (B_3),
then $V_s(n)$ is supplied to induction motor and is retained ($k = n$, $V_s(k)$, counter $t = 0$).

5c. **If** in comparison period (E), rotor acceleration command is much less than zero (A_3), comparison voltage vectors have been input (H), and $\Delta a(V_s(n)) < \eta\Delta a(V_s(n-1))$ (B_5), **then** $V_s(n)$ is supplied to induction motor and is retained ($k = n$, $V_s(k)$, counter $t = 0$).

5d. **If** in comparison period (E), rotor acceleration command is near zero (A_2), comparison voltage vectors have been input (H), and $\Delta a(V_s(n)) < \eta\Delta a(V_s(n-2))$ (B_7), **then** $V_s(n)$ is supplied to induction motor and is retained ($k = n$, $V_s(k)$, counter $t = 0$).

6. **If** in comparison period (E), rotor acceleration command is much larger than zero(A_1), comparison voltage vectors have been input (H), and $\Delta a(V_s(n)) \leq \eta\Delta a(V_s(n+1))$ (B_2), **then** $V_s(n+1)$ is supplied to induction motor and is retained ($k = n+1$, $V_s(k)$, counter $t = 0$).

7. **If** in comparison period (E), rotor acceleration command is near zero (A_2), $\omega_o^* > 0$ (F), comparison voltage vectors have been inputted (H), and $\Delta a(V_s(n)) \leq \eta\Delta a(V_s(n+2))$ (B_4), **then** $V_s(n)$ is supplied to induction motor and is retained ($k = n+2$, $V_s(k)$, counter $t = 0$).

8. **If** in comparison period (E), rotor acceleration command is near zero (A_2), $\omega_o^* < 0$ ($\neg$F), comparison voltage vectors have been inputted (H), and $\Delta a(V_s(n)) \geq \eta\Delta a(V_s(n-2))$ (B_8), **then** $V_s(n)$ is supplied to induction motor and is retained ($k = n-2$, $V_s(k)$, counter $t = 0$).

9. **If** in comparison period (E), rotor acceleration command is much less than zero (A_3), comparison voltage vectors have been input (H), and $\Delta a(V_s(n)) \geq \eta\Delta a(V_s(n-1))$ (B_6), **then** $V_s(n)$ is supplied to induction motor and is retained ($k = n-1$, $V_s(k)$, counter $t = 0$).

In the voltage vector retaining period, the voltage vector supplied is unchanged if the rotor acceleration is within the range (a_{min}, a_{max}), where $a_{min} = a^* - a_{th}$ and $a_{max} = a^* + a_{th}$, a^* is the rotor acceleration command and a_{th} is a specified threshold value (Figure 5.4). When the rotor acceleration is larger than a_{max} and if the rotor acceleration command is larger than zero, then a zero voltage vector is supplied; if the rotor acceleration command is less than zero, then optimum voltage vector is supplied. When the rotor acceleration is less than a_{min} and if the rotor acceleration command is less than zero, then zero voltage vector is supplied; if the rotor acceleration command is larger than zero, then the optimum voltage vector is supplied.

In the retaining period, the control knowledge is therefore expressed as five rules as follows.

10a. **If** in retaining period (denote as $\neg$E), rotor acceleration command is much larger than zero (denote as A_1), zero voltage vector is supplied (denote as C_1), and rotor acceleration is larger than a_{max} (Figure 5.4) (denote as D_1), **then** the zero voltage vector is kept ($V_s(k)$, $k = 0$).

10b. **If** in retaining period ($\neg$E), rotor acceleration command is much less than zero (A_3), zero voltage vector is supplied (C_1), and rotor acceleration is less than a_{min} (D_3), **then** zero voltage vector is supplied ($V_s(k)$, $k = 0$).

11a. **If** in retaining period ($\neg$E), rotor acceleration command is much larger than zero (A_1), nonzero optimum voltage vector is supplied (C_2), and rotor acceleration is larger than a_{max} (D_1), **then** the nonzero voltage vector is recorded in m ($m = k$, $k = 0$), and zero voltage vector is supplied ($V_s(k)$).

11b. **If** in retaining period ($\neg$E), rotor acceleration command is much less than zero (A_3), nonzero optimum voltage vector is supplied (C_2), and rotor acceleration is less than a_{min} (D_3),

then the nonzero voltage vector is recorded in m ($m = k, k = 0$), and zero voltage vector is supplied ($V_s(k)$).

12. **If** in retaining period (¬E), rotor acceleration is between a_{max} and a_{min} of reference value (D_2),
then voltage vector supplied (the optimum voltage vector or zero voltage vector) is unchanged ($V_s(k)$).

13a. **If** in retaining period (¬E), rotor acceleration command is much larger than zero(A_1), zero voltage vector is supplied (C_1), and rotor acceleration is less than a_{min} (D_3),
then the optimum voltage vector is restored and supplied ($k = m$, $V_s(k)$).

13b. **If** in retaining period (¬E), rotor acceleration command is much less than zero (A_3), zero voltage vector is supplied (C_1), and rotor acceleration is larger than a_{max} (D_1),
then the optimum voltage vector is restored and supplied ($k = m$, $V_s(k)$).

14a. **If** in retaining period (¬E), rotor acceleration command is much larger than zero (A_1), nonzero optimum voltage vector is supplied (C_2), and rotor acceleration is less than a_{min} (D_3),
then the optimum voltage vector supplied is unchanged ($V_s(k)$).

14b. **If** in retaining period (¬E), rotor acceleration command is much less than zero (A_3), nonzero optimum voltage vector is supplied (C_2), and rotor acceleration is larger than a_{max} (D_1),
then the optimum voltage vector supplied is unchanged ($V_s(k)$).

The detailed algorithm of the expert-system acceleration control for an induction motor is given in APPENDIX D.

A typical sequence of the expert-system acceleration control is illustrated in Table 5.2.

Table 5.2 A typical sequence of the expert-system acceleration control.

Iteration No.	Working memory	Conflict Set	Rule fired	Remarks
i	$A_1 \wedge \neg E \wedge D_1 \wedge C_2$	11	11	Modify database $m = k$, $k = 0$
i + 1	$\neg E \wedge D_2$	12	12	Retaining voltage vector
i + 2	$A_1 \wedge \neg E \wedge D_3 \wedge C_1$	13	13	Modify database $k = m$
i + 3	$E \wedge \neg H$	1,2,3,4	1	Modify database $n = i$
i + 4	$E \wedge A_1 \wedge \neg H$	2	2	Input comparing voltage vectors
i + 5	$E \wedge A_1 \wedge H \wedge B_1$	6	6	Modify database $k = n + 1$, $t = 0$
i + 6	$\neg E \wedge D_3 \wedge C_2$	14	14	Keep voltage vector
i + 7	$\neg E \wedge D_2$	12	12	Retaining voltage vector
i + 8	$\neg E \wedge D_1 \wedge C_2$	11	11	Modify database $m = k$, $k = 0$
⋮	⋮	⋮	⋮	⋮
i + k	S		Halt	Motor stops

(Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

5.6 Computer Simulation and Comparison

Five computer simulation examples are presented to prove the feasibility of expert-system based control. In the first example, the performance of an inverter-fed induction motor driving a constant torque load is investigated. The simulation results of DSC and the proposed expert-system controller are compared. The second example compares the robustness of the two controllers. The third example verifies that the expert-system controller has exchangeability, that is, the same controller can be used for different induction motors. The fourth example demonstrates that the expert-system controller can work in a very low speed range. The fifth example simulates the expert-system controller working with an encoder model and establishes that the encoder should have a minimum required precision. A computer simulation of the expert-system acceleration controller is implemented using MATLAB®/Simulink software as shown in Figure 5.7.

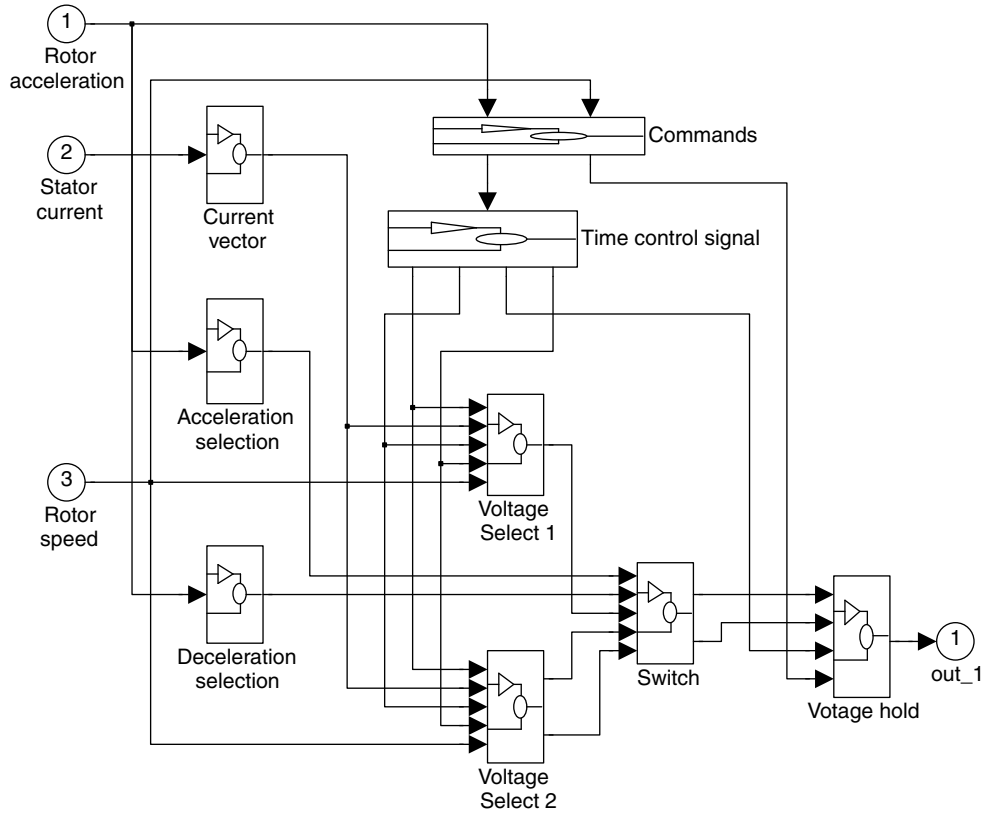


Figure 5.7 Simulink model of the expert-system acceleration controller.

The simulation model of the acceleration controller consists of a 'Current vector' block, a 'Commands' block, a 'Time control signal' block, an 'Acceleration selection' block, a 'Deceleration selection' block, a 'Voltage select 1' block, a 'Voltage select 2' block,

a 'Switch' block, and a 'Voltage hold' block. The inputs of the model are the rotor acceleration a (which may be obtained from the rotor speed by a differential calculation), stator current i_s, and rotor speed signal ω. The output is the stator voltage vector supplied to the induction motor.

The 'Current vector' block (which stores the expert-system rule 1) transfers the stator current into current vector. The 'Commands' block issues rotor speed command ω^*, rotor acceleration command a^*, and rotor acceleration error Δa. The 'Time control signal' block generates time control signals of the comparison period and retaining period. The 'Acceleration selection' block and 'Deceleration selection' block implement the rotor acceleration/deceleration comparisons and selections. The 'Voltage select 1' block (which stores the expert-system rules 2, 3, 5a, 5b, 6, 7, 8) issues comparison voltage vectors according to rotor acceleration command. The 'Voltage select 2' block (which stores rules 4, 5c, 5d, 9) issues comparison voltage vectors according to the deceleration command. The 'Switch' block is used to switch control signals of the rotor acceleration and deceleration. The 'Voltage hold' block (which stores rules 10–14) generates and retains stator voltage vector signals according to the results of the rotor acceleration/deceleration comparisons and selections.

5.6.1 *The First Simulation Example*

This example is a comparison of DSC and the expert-system controller in respect of rotor acceleration, torque, current, and flux. The induction motor parameters chosen for the simulation studies are listed in 'Motor 1' of Appendix B, and the load torque is 20 Nm.

Speed command:

$$\omega_o^* = 40(\text{rad/s}) \quad 0\,\text{s} \le t < 0.4\,\text{s}$$
$$\omega_o^* = 20(\text{rad/s}) \quad 0.4\,\text{s} \le t < 0.8\,\text{s}$$

Rotor acceleration command:

$$a^* = 300(\text{rad/s}^2) \quad \omega_o^* = 40 \wedge \omega_o < 40$$
$$a^* = 0(\text{rad/s}^2) \quad \omega_o^* = 40 \wedge \omega_o = 40$$
$$a^* = -300(\text{rad/s}^2) \quad t \ge 0.4\,\text{s} \wedge \omega_o > 20$$
$$a^* = 0(\text{rad/s}^2) \quad \omega_o^* = 20 \wedge \omega_o = 20$$

Figure 5.8a, Figure 5.9a, Figure 5.10a, and Figure 5.11a are the simulation results for the expert-system controller. Figure 5.8b, Figure 5.9b, Figure 5.10b, and Figure 5.11b are the simulation results for the DSC controller.

As shown in the above figures, the responses of speed, torque and rotor acceleration of the two controllers are almost the same. With the proposed controller, however, oscillations in rotor acceleration and torque are produced during the transition period. Although the flux is not directly controlled by the proposed scheme, the stator current and flux do not deviate significantly from the corresponding curves of DSC.

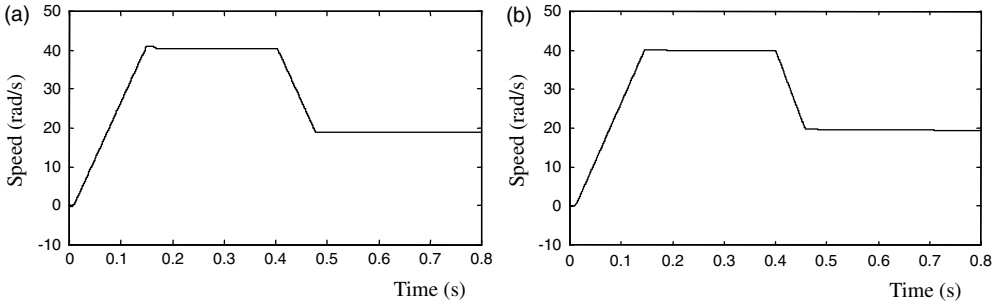


Figure 5.8 (a) Rotor speed response of expert-system controller; (b) Rotor speed response of DSC. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

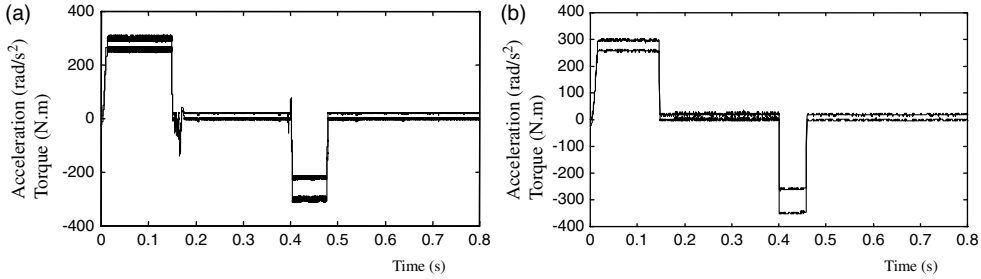


Figure 5.9 (a) Controlled acceleration and torque response of expert-system controller; (b) Acceleration response and controlled torque of DSC controller. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

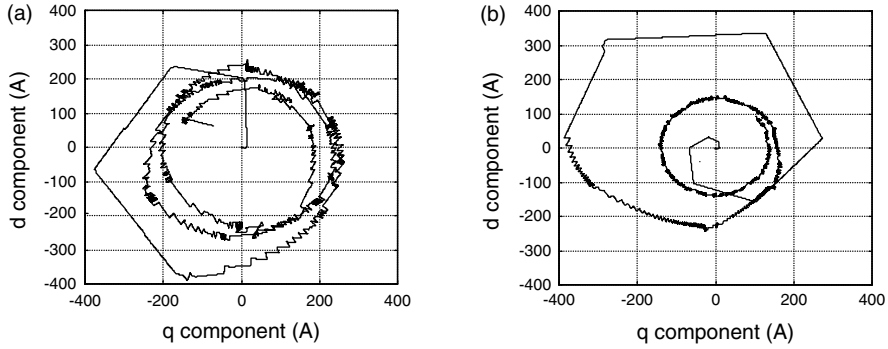


Figure 5.10 (a) Stator current vector of expert-system controller, time = 0 s ∼ 0.15 s; (b) Stator current vector of DSC, time = 0 s ∼ 0.13 s. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

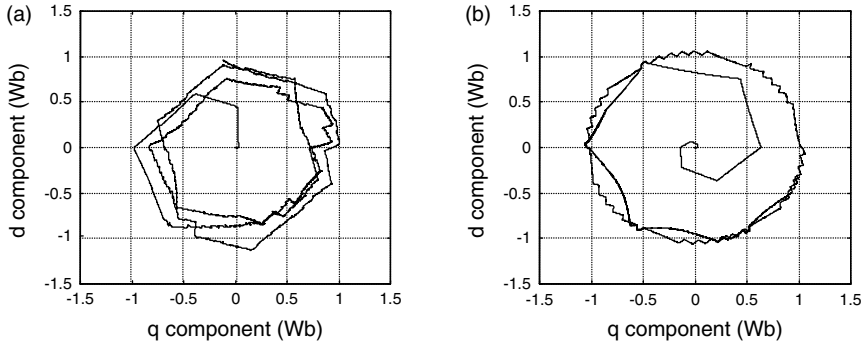


Figure 5.11 (a) Primary flux vector of expert-system controller, time = 0 s ~ 0.15 s; (b) Primary flux vector of DSC, time = 0 s ~ 0.13 s. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

5.6.2 *The Second Simulation Example*

Let $\tilde{i}_s^s = i_s^s + n$ ($\tilde{i}_s^s$: current with noise, i_s^s: current, *n:* sensor noise). Substituting $\tilde{i}_s^s$ to Equation (5.4) which is DSC flux calculation model (Takahashi and Noguchi, 1986), Equation (5.24) may be obtained and the flux error may be calculated by Equation (5.25).

$$\lambda_\mu^s(t_0) = \int_0^{t_0} (V_s^{(k)} - R_s \tilde{i}_s^s)dt = \int_0^{t_0} (V_s^{(k)} - R_s i_s^s)dt - \int_0^{t_0} R_s n dt \tag{5.24}$$

$$\Delta\lambda_\mu^s(t_0) = \int_0^{t_0} R_s n dt \tag{5.25}$$

When the control time is long, it is difficult to ensure that there is no noise source of nonzero mean value in a practical control system (Engberg and Larsen, 1995). Accumulation of flux error in DSC may be produced by a current sensor noise with nonzero mean value.

In order to verify the robustness of the new controller to load changes and noise, an oscillating load is applied to the motor and a drift noise (nonzero mean value) is added to the current.

Figure 5.12a shows that the expert-system controller has good noise immunity and effective control is obtained over a long period of time. On the other hand, DSC is sensitive to the noise and the load. At $t = 2$ s, the motor speed drops to zero and the controller fails (Figure 5.12b).

It has been said that DSC can achieve an instantaneous torque response. However, this is true only when the integral calculation of flux does not have excessive error accumulation over a long period. When an excessive error accumulation (such as that created by noise) occurs, the instantaneous torque response is not assured and the error would not be corrected, as shown in this simulation example. Since the new expert-system controller does not

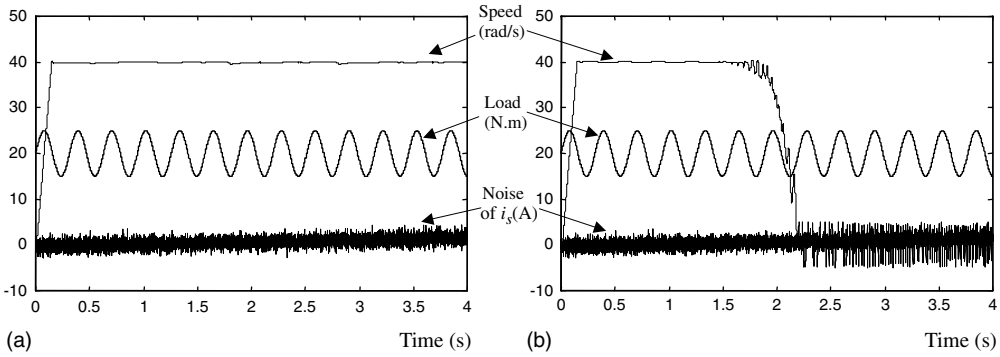


Figure 5.12 (a) Expert-system controller with drift noise and oscillating load; (b) DSC controller with drift noise and oscillating load. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A rule-based acceleration control scheme for an induction motor," *IEEE Transactions on Energy Conversion*, **17**(2), 2002: 254–259. © 2002 IEEE.)

require the calculation of integrals, the control is more effective and accurate over a longer period than DSC.

5.6.3 The Third Simulation Example

Because the expert-system controller is independent of motor parameters, the third example will verify that the new controller has exchangeability when it is used for different induction motors.

When the expert-system controller is used to control a 0.75 kW induction motor, only the inverter needs to be changed. The controller is the same as that used for the 7.5 kW induction motor in the first simulation example. The 0.75 kW induction motor parameters chosen for the simulation studies are listed in 'Motor 2' of Appendix B with load 2 Nm. The speed command and the rotor acceleration command are the same as the first simulation example.

The simulation results shown in Figure 5.13a and b are almost the same as those obtained from the 7.5 kW induction motor in the first simulation example.

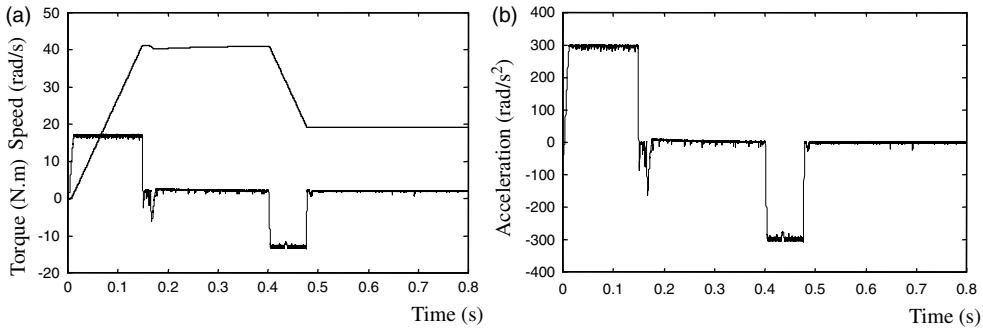


Figure 5.13 (a) Speed and torque of 0.75 kW induction motor with expert-system controller; (b) Controlled rotor acceleration of 0.75 kW induction motor with expert-system controller.

When the expert-system controller is applied to control a 0.147 kW induction motor (model 295, Bodine Electric Company), only the inverter needs to be changed. The controller is the same as that used for the 7.5 kW induction motor in the first simulation example. The 0.147 kW induction motor parameters chosen for the simulation studies are listed in 'Motor 3' of APPENDIX B with load 1 Nm. The speed command and the rotor acceleration command are the same as the first simulation example. Figure 5.14 shows stator voltage and current of the 0.147 kW induction motor. Figure 5.15 shows controlled rotor acceleration and torque responses of the induction motor, while Figure 5.16 shows rotor speed response of the induction motor.

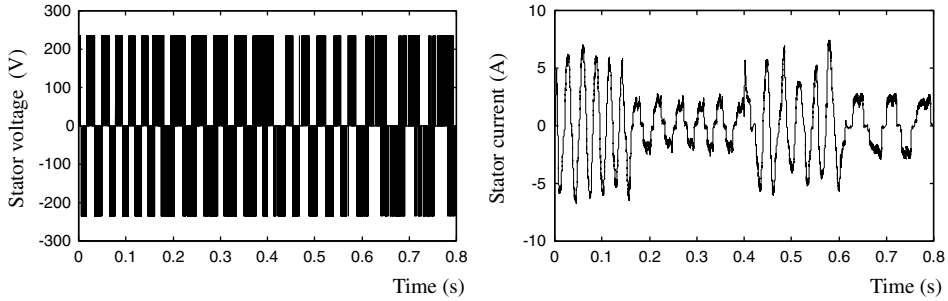


Figure 5.14 Stator voltage and current of 0.147 kW induction motor.

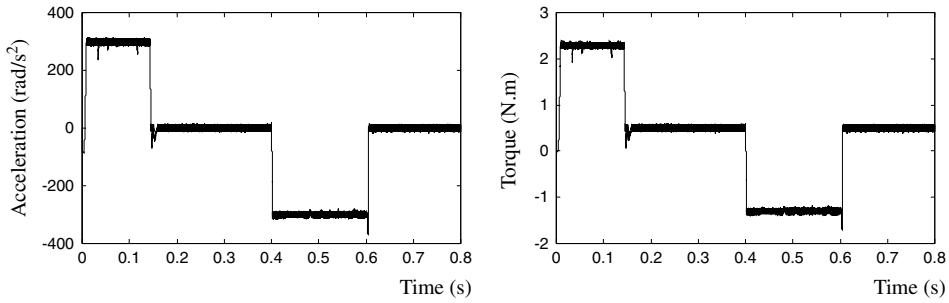


Figure 5.15 Controlled rotor acceleration and torque response of 0.147 kW induction motor.

The controlled rotor acceleration, torque response, and speed response shown in Figures 5.15 and 5.16 are almost the same as those obtained from the 7.5 kW induction motor in Figures 5.8a and 5.9a of the first simulation example and from the 0.75 kW induction motor in Figure 5.13a and b.

5.6.4 The Fourth Simulation Example

The fourth simulation example will demonstrate that the expert-system controller can work very well in very low speed range. The same 0.147 kW induction motor (Bodine Electric Company model 295) is used and all the simulation parameters and conditions are the same

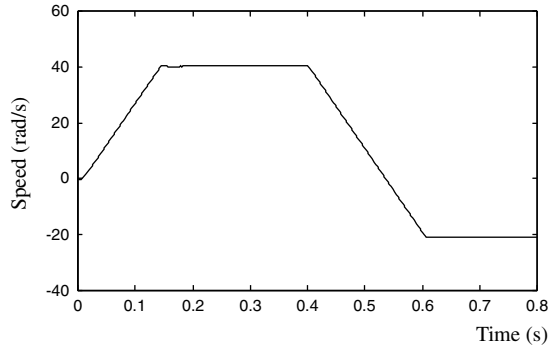


Figure 5.16 Speed response of 0.147 kW induction motor.

those in third simulation example, except that the speed command is reset. Figure 5.17 shows the controlled acceleration and torque response of the 0.147 kW induction motor with the expert-system controller when the speed command is set as 6 rad/s and −6 rad/s, while Figure 5.18 shows the rotor speed response of the induction motor.

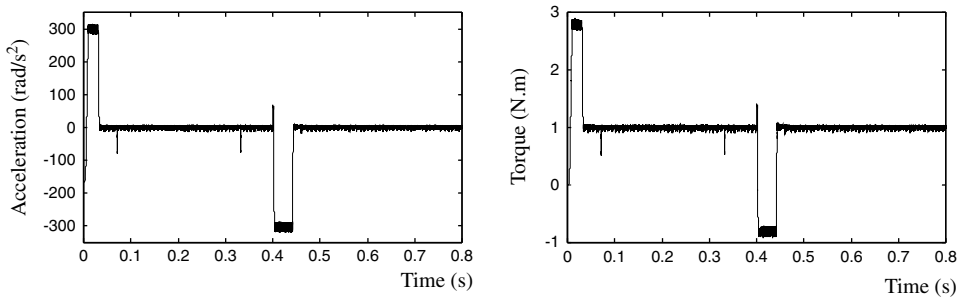


Figure 5.17 Controlled rotor acceleration and torque response of 0.147 kW induction motor when speed command is 6 rad/s and −6 rad/s.

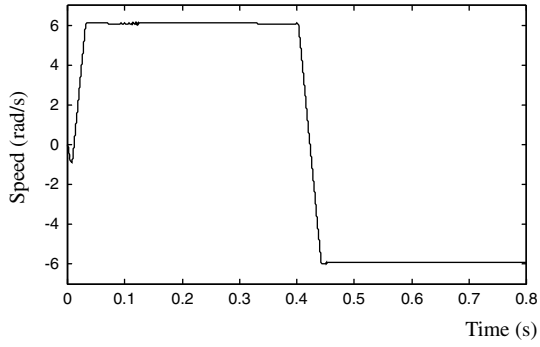


Figure 5.18 Speed response of 0.147 kW induction motor when speed command is 6 rad/s and −6 rad/s.

Figure 5.19 shows controlled acceleration and torque response of the 0.147 kW induction motor with the expert-system controller when the speed command is set as 1 rad/s and −1 rad/s, while Figure 5.20 shows the rotor speed response of the induction motor.

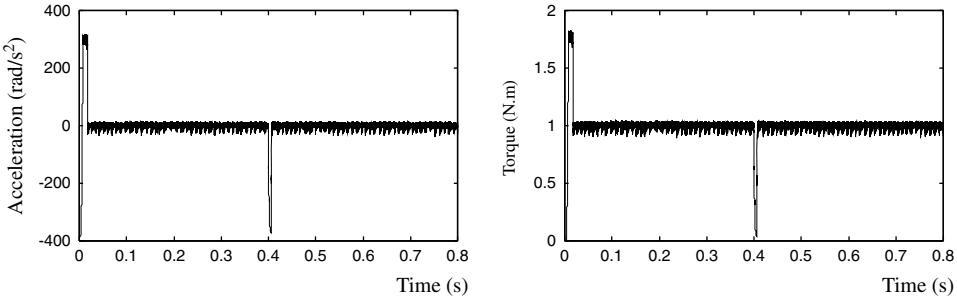


Figure 5.19 Controlled rotor acceleration and torque response of 0.147 kW induction motor when speed command is 1 rad/s and −1 rad/s.

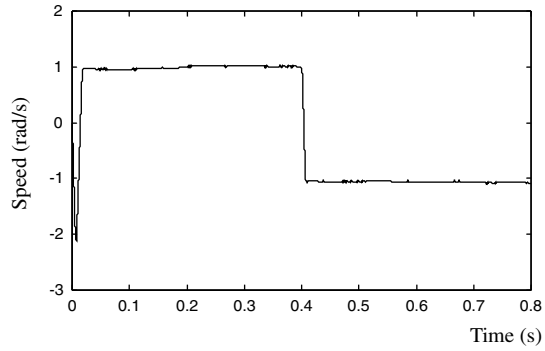


Figure 5.20 Speed response of 0.147 kW induction motor when speed command is 1 rad/s and −1 rad/s.

5.6.5 *The Fifth Simulation Example*

The fifth simulation example demonstrates rotor speed responses of the expert-system acceleration controller working with speed encoders of various encoder precisions. The simulation program consists of the encoder and decoder models built in Chapter 3, the 0.147 kW voltage-input model of induction motor built in Chapter 3, and the expert-system controller model in Figure 5.7. The motor parameters are listed in 'Motor 3' of Appendix B and the speed command is from 0 rad/s to 40 rad/s. The encoder model receives rotor speed of the induction motor model and outputs speed code strings to the decoder model, while output of the decoder model is sent to the expert-system controller as rotor speed signal. Figure 5.21 shows the rotor speed responses of the induction motor with and without the encoder and decoder models, the encoder precision being set as 2000 pulses/revolution.

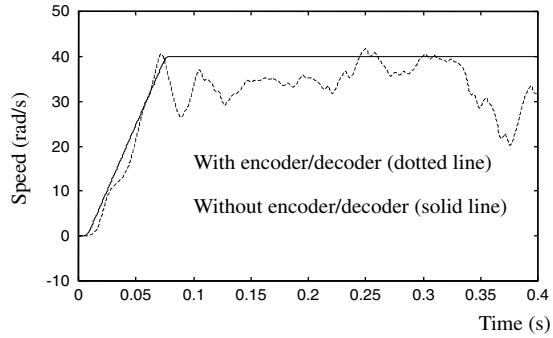


Figure 5.21 Speed responses of 0.147 kW induction motor with and without an encoder of 2000 pulses/revolution.

The rotor speed response with encoder and decoder models is different from the speed response without encoder and decoder models, because an encoder of lower precision can decrease control precision of the expert-system controller.

Figure 5.22 show the rotor speed responses of the induction motor with and without the encoder and decoder models, the encoder having a precision of 20 000 pulses/revolution.

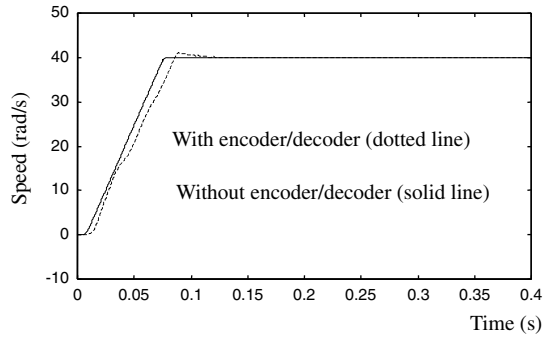


Figure 5.22 Speed responses of 0.147 kW induction motor with and without an encoder of 20 000 pulses/revolution.

Figure 5.23 shows the rotor speed responses of the induction motor with and without the encoder and decoder models, the encoder now having a resolution of 200 000 pulses/revolution.

Figures 5.21–5.23 show that the encoder's precision is extremely important for the expert-system acceleration controller. When the encoder precision is up to 200 000 pulses/revolution, the expert-system controller can have a good performance. Therefore, a high precision encoder, such as an expensive Gurley Model 8435H hollow-shaft optical encoder

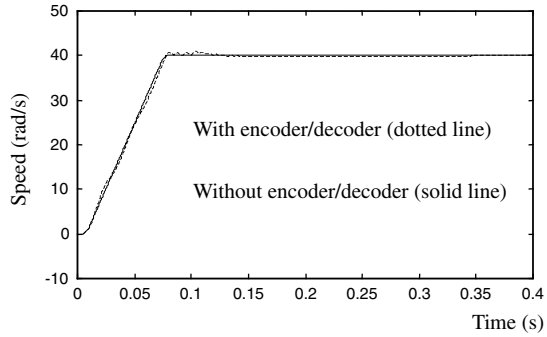


Figure 5.23 Speed responses of 0.147 kW induction motor with and without an encoder of 200 000 pulses/revolution.

with 900 000 counts/revolution is needed for hardware implementation of the expert-system acceleration controller.

5.7 Summary

The expert-system based control scheme of induction motor is quite different from the usual vector control schemes that depend on flux and torque calculations. Due to the use of inference instead of algebraic calculations, the expert-system controller has a small control error but no cumulative error. Another valuable property is that the controller is independent of the parameters of the induction motor, so the same controller can be used for different induction motors. Lastly, the control may be performed at any time, whereas conventional vector control must be performed from an initial state. Since the expert-system algorithm consists mainly of logic operations, the execution time of the control algorithm should be shorter than that of the conventional vector-control algorithm. But, since the rotor acceleration values are obtained by a differential operation on the angular speed, a high precision encoder is needed and control error may be produced by the noise present in the signal from the speed encoder.

Expert system is an effective method for induction motor control. It is envisaged that more and more advanced induction motor drives will be controlled using AI principles and algorithms (Bose, 1993; Passino, 1996b). The possible developments of the expert-system based acceleration controller are: (1) to systematize further the principle of rotor acceleration control, (2) to optimize further the control rules and algorithms, (3) to tackle the problem of noise in the measured rotor acceleration, and (4) DSP based hardware implementation with a high-precision speed sensor.

References

Åström, K.J. and Anton, J.J. (1984) Expert control. *Proceedings, 9th IFAC (International Federation of Automatic Control)* World Congress, Budapest, Hungary.

Åström, K.J. and Björn, W. (1995) *Adaptive Control*, Addison Wesley Publishing Company, Reading, MA.

Bose, B.K. (1986) *Power Electronics and AC Drives*, Prentice Hall, Englewood Cliffs, NJ.

Bose, B.K. (1993) Power electronics and motion control-technology status and recent trends. *IEEE Transactions on Industry Applications*, **29**, 902–909.

Buchanan, B. and Shortliffe, E.H. (1984) *Rule-Based Expert Systems*, MYCIN, Addison-Wesley, Reading, MA.

Engberg, J., and Larsen, T., (1995) *Noise Theory of Linear and Nonlinear Circuits*, John Wiley & Sons Ltd, Chichester.

George, F.L. and William, A.S. (1989) *Artificial Intelligence and the Design of Expert System*, Benjamin-Cummings Publishing Company, Inc., San Francisco, CA.

Hayes-Roth, F. (1985) *Building Expert Systems*, Addison-Wesley, Reading, MA.

Krause, P.C., Wasynczuk, O., and Sudhoff, S.D. (1995) *Analysis of Electric Machinery*, IEEE Press, New Jersey.

Nabae, A., Ogasawara, S., and Akagi, H. (1986) A novel control scheme for current-controlled PWM Inverters. *IEEE Transactions on Industry Applications*, **22**(4), 697–701.

Novotny, D.W. and Lipo, T.A. (1996) *Vector Control and Dynamics of AC Drives*, Oxford University Press, Oxford.

Passino, K.M. and Lunardhi, A.D. (1996a) Qualitative analysis of expert control systems, in *Intelligent Control Systems: Theory and Applications* (eds M.M. Gupta and N.K. Sinha) IEEE Press, New Jersey.

Passino, K.M. (1996b) Towards bridging the perceived gap between conventional and intelligent control, in *Intelligent Control Systems: Theory and Applications* (eds M.M. Gupta and N.K. Sinha) IEEE Press, New Jersey.

Shi, K.L., Chan, T.F., Wong, Y.K., and Ho, S.L. (1999) A new acceleration control scheme for an inverter-fed induction motor. *Electric Machines and Power Systems, USA*, **27**(5), 527–541.

Takahashi, I. and Noguchi, T. (1986) A new quick-response and high-efficiency control strategy of an induction motor. *IEEE Transactions on Industry Applications*, **22**(5), 820–827.

6

Hybrid Fuzzy/PI Two-Stage Control[1]

6.1 Introduction

Field orientation principle is one of the most promising methods to achieve high performance in adjustable speed induction motor drives (Bose, 1993). But, due to the dependence of performance on the motor parameters and the complicated calculations involved, accurate vector control is difficult to implement in practice. Two features of field-oriented control, however, deserve attention. Firstly, although the field-oriented controller does not control the frequency directly, its supply frequency does change and its slip frequency is constant during the acceleration/deceleration period. Secondly, when the torque command is constant, the supply current magnitude will remain constant. The first feature may be proved by noting the relationship between the supply frequency ω, the slip frequency ω_r, and the rotor angular speed ω_o:

$$\omega = \omega_r + \frac{P}{2}\omega_o. \tag{6.1}$$

[1] (a) Portions reprinted by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "A novel two-stage speed controller for an induction motor," The 1997 IEEE Biennial International Electrical Machines and Drives Conference, Paper MD2-4, May 18–21, 1997, Milwaukee, Wisconsin, U.S.A. © 1997 IEEE

(b) Portions reprinted by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.

(c) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "An improved two-stage control scheme for an induction motor," Proceedings of the IEEE 1999 International Conference on Power Electronics and Drive Systems, pp. 405–410, July 27–29, 1999, Hong Kong. © 1999 IEEE.

(d) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A novel hybrid fuzzy/PI two-stage controller for an induction motor drive," IEEE International Electric Machines and Drives Conference (IEMDC 2001), pp. 415–421, June 17–20, 2001, Cambridge, MA, U.S.A. © 2001 IEEE.

(e) Portions reprinted from K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Modeling and simulation of a novel two-stage controller for an induction motor," *International Association of Science and Technology for Development (IASTED) Journal on Power and Energy Systems*, **19**(3), 257–264, © 1999, with permission from ACTA Press.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

The slip frequency ω_r is given by Equation (2.18):

$$\omega_r = \frac{3R_r T^*}{P\lambda_{dr}^{e\,*2}} \tag{6.2}$$

where R_r is the rotor resistance, P is the number of poles, T^* denotes the torque command, and $\lambda_{dr}^{e\,*}$ denotes rotor flux command.

If T^* is maintained constant during acceleration, ω_r is also constant. As ω_o changes during acceleration and deceleration, ω has to be varied so that Equation (6.1) is satisfied.

The second feature may be proved using the following field orientation conditions (Trzynadlowski, 1994):

$$\begin{aligned} \lambda_{qr}^e &= 0 \\ \lambda_{dr}^e &= const. \end{aligned} \tag{6.3}$$

Substituting Equation (6.3) into Equation (2.15),

$$i_{ds}^e = \frac{\lambda_{dr}^e}{L_M} = const. \tag{6.4}$$

Equation (2.17) may be rewritten as:

$$i_{qs}^e = \frac{T^*}{k_q \lambda_{dr}^e} \tag{6.5}$$

where $k_q = \dfrac{PL_M}{3L_r}$.

Stator phase current magnitude $|\hat{I}_s|$ can be expressed by:

$$|\hat{I}_s| = \frac{2}{3}\sqrt{(i_{ds}^e)^2 + (i_{qs}^e)^2} \tag{6.6}$$

Substituting Equations (6.4) and (6.5) into Equation (6.6),

$$|\hat{I}_s| = \frac{2}{3}\sqrt{\left(\frac{\lambda_{dr}^e}{L_M}\right)^2 + \left(\frac{T^*}{k_q \lambda_{dr}^e}\right)^2}. \tag{6.7}$$

When the torque command T^* is constant, Equation (6.7) becomes:

$$|\hat{I}_s| = \frac{2}{3}\sqrt{i_{ds}^{e2} + i_{qs}^{e2}} = const.$$

In this chapter, the above two features of field-oriented control are employed in the design of a novel two-stage control scheme for an induction motor. A simulation study on the proposed hybrid fuzzy/PI two-stage controller is carried out using the software MATLAB®/Simulink and the results are compared with that obtained from an indirect field-oriented controller.

6.2 Two-Stage Control Strategy for an Induction Motor

The current-input induction motor model shown in Figure 6.1 has three inputs, namely the stator current magnitude I_s, supply frequency ω, and load torque T_L. It has an output, namely the rotor speed ω_o. The relationship between the output and inputs may be expressed as:

$$\omega_o = IM(I_s, \omega, T_L). \tag{6.8}$$

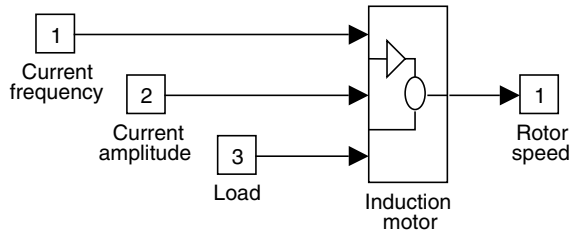


Figure 6.1 An induction motor model with current input. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

The speed response of the motor may be divided into two stages, an initial acceleration/deceleration stage, and a final steady-state stage, as shown in Figure 6.2.

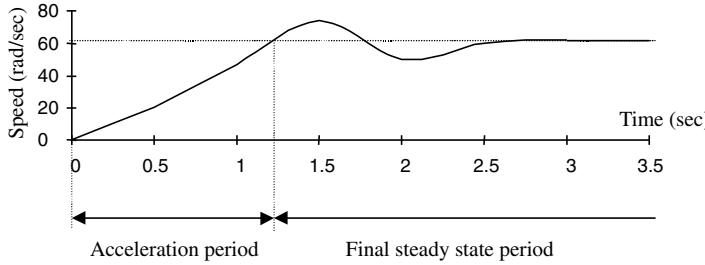


Figure 6.2 Typical speed response of an induction motor. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

The basic principle of the two-stage controller may be described as follows (Shi, Chan and Wong, 1997; Shi, Chan and Wong, 1999).

1. During the acceleration/deceleration stage, the stator current magnitude $|I_s|$ is maintained constant and the rotor accelerates or decelerates depending on the input frequency ω.

2. During the final steady-state stage, the input frequency ω is held constant and the speed of rotor ω_o is maintained constant by controlling the stator current magnitude $|I_s|$.

In the two-stage speed control scheme, the relationship between inputs and outputs is described by Table 6.1.

Table 6.1 Two-stage speed control scheme.

Stages	Inputs		Output	Control
	ω	$\|I_s\|$	ω_o	objective
Acceleration or deceleration	change	constant	change	change speed
Steady-state	constant	change	constant	eliminate oscillations

(Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

A problem arises as how to vary the supply frequency from zero to the final frequency of the command speed. If the supply frequency rises too fast, the torque will produce oscillations so that the acceleration/deceleration period is longer, but if the supply frequency rises too slowly, the torque will be so small that the acceleration/deceleration period is again prolonged. To tackle this problem, a fuzzy controller is designed using the equations of field-oriented control.

6.3 Fuzzy Frequency Control

Fuzzy-logic control is an important intelligent control method which uses fuzzy rule sets and linguistic representation of a human's knowledge to control a plant. In the proposed two-stage control scheme, fuzzy logic frequency control based on the frequency feature of the field orientation principle is developed. During the acceleration stage, the torque command has a larger value, whereas during the steady-state stage, the torque command has a smaller value. These commands can be determined from the difference between rotor speed ω_o and speed command $\omega_o{}^*$:

$$T^* = \begin{cases} T_{\text{acceleration}} & \text{when } \Delta\omega_o \neq 0 \\ T_{\text{steady}} & \text{when } \Delta\omega_o = 0 \end{cases} \tag{6.9}$$

where speed error $\Delta\omega_o = \omega_o{}^* - \omega_o$.

Substituting Equations (6.2)–(6.9), the slip frequency of field-oriented control is

$$\omega_r = \begin{cases} \dfrac{3R_r T_{\text{acceleration}}}{P\lambda_{dr}^{e\,*2}} & \text{when } \Delta\omega_o \neq 0 \\ \dfrac{3R_r T_{\text{steady}}}{P\lambda_{dr}^{e\,*2}} & \text{when } \Delta\omega_o = 0 \end{cases} \tag{6.10}$$

At steady state, the torque command T_{steady} is equal to the load torque T_L. If T_L is a function of the motor speed, that is, $T_L = T_L(\omega_o)$, then

$$T_{\text{steady}} = T_L(\omega_o). \tag{6.11}$$

For the present investigation, the following load characteristic is assumed (Wade, Dunnigan and Williams, 1997):

$$T_L = \mu\omega_o \tag{6.12}$$

where coefficient $\mu = 0.18$ Nm/(rad/s).

When $\Delta\omega = 0$, replacing ω_o with ω_o^*, and substituting Equation (6.12) into Equation (6.10), the steady-state slip frequency may be written as:

$$\omega_r = \begin{cases} \dfrac{3R_r}{P\lambda_{dr}^{e\,*2}} \cdot T_{\text{acceleration}} & \text{when } \Delta\omega_o \neq 0 \\ \dfrac{3R_r}{P\lambda_{dr}^{e\,*2}} \cdot \mu\omega_o^{\,*} & \text{when } \Delta\omega_o = 0 \end{cases}. \tag{6.13}$$

Because the slip frequency ω_r is a function of variables $\Delta\omega$ and ω_o^*, it can be written as:

$$\omega_r = f(\omega_o^*, \Delta\omega_o) \tag{6.14}$$

According to (6.14), the speed error $\Delta\omega$ and speed command ω_o^* can be used as the inputs of the fuzzy frequency control which consists of fuzzification, fuzzy logic inference, rulebase, database and defuzzification. Figure 6.3 shows a fuzzy frequency control system.

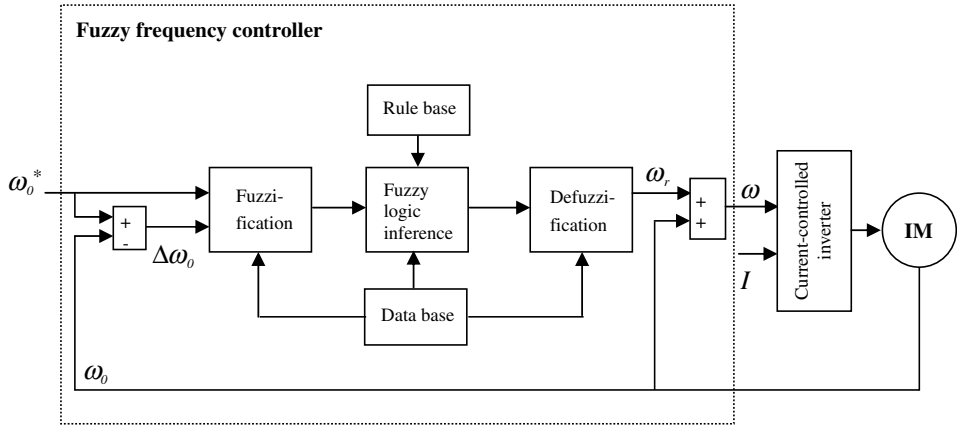


Figure 6.3 Fuzzy frequency control system. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

The fuzzification operation is the process which converts the crisp input values ($\Delta\omega$, ω_o^*) to fuzzy sets. A fuzzy set consists of elements each having a degree of membership and associated with linguistic values. The defuzzification operation is the process which determines the best numerical value ω_s to represent a given fuzzy set. The database stores the memberships of fuzzy variables, while the rulebase provides the necessary linguistic control rules for the fuzzy inference (Sousa and Bose, 1994).

6.3.1 *Fuzzy Database*

The fuzzy slip frequency control uses two fuzzy state variables (speed command and speed error) and one control variable (slip frequency). Consequently, the fuzzy database consists of membership functions of speed command, speed error, and slip frequency.

6.3.1.1 Membership Functions of Speed Command

Let the normal range of speed (ω_o) be from -120 rad/s to 120 rad/s. The universe of discourse of the speed command fuzzy variable is divided into seven overlapping fuzzy sets: $F_{wo} = \{\mathrm{NB_{wo}}, \mathrm{NM_{wo}}, \mathrm{NS_{wo}}, \mathrm{Z_{wo}}, \mathrm{PS_{wo}}, \mathrm{PM_{wo}}, \mathrm{PB_{wo}}\}$, as shown in Figure 6.4, where $\mu_{(wo)}$ denotes the degree of membership of speed command.

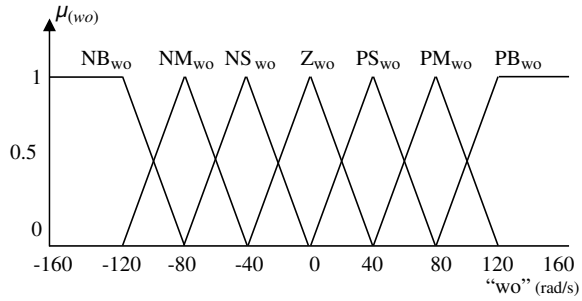


Figure 6.4 Membership function of speed command. NB: negative big; NM: negative medium; NS: negative small; Z: zero; PB: positive big; PM: positive medium; PS: positive small. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

6.3.1.2 Membership Functions of Speed Error

When the range of speed command (ω_o^*) is from -120 rad/s to 120 rad/s, the range of the speed error ($\Delta\omega_o$) is from -240 rad/s to 240 rad/s. The universe of discourse of the speed error fuzzy variable is divided into three overlapping fuzzy sets: $F_{Dwo} = \{\mathrm{N_{Dwo}}, \mathrm{Z_{Dwo}}, \mathrm{P_{Dwo}}\}$ as shown in Figure 6.5, where $\mu_{(Dwo)}$ denotes the degree of membership of speed error.

6.3.1.3 Membership Functions of Slip Frequency

For the induction motor to be studied, $R_r = 0.151\,\Omega$, $P = 6$, and $T_{(\text{acceleration})} = 100$ Nm. If $\lambda_{dr}^{e\,*} = 0.67$ Wb during the acceleration stage, then $\omega_r = 16.8$ rad/s by Equation (6.13). For the

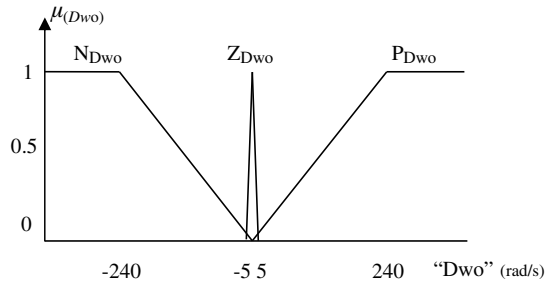


Figure 6.5 Membership function of speed error. N: negative; Z: zero; P: positive. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

steady-state stage, the values of steady-state slip frequency can be calculated from (6.13). The speed command, slip frequency, and fuzzy linguistic values are shown in Table 6.2.

Table 6.2 Speed, slip frequency, and fuzzy linguistic values.

Stages	$\omega_o{}^*$	ω_r	F_{wr}
Deceleration	—	−16.8	NB_{wr}
Steady state	−120	−3.64	NM_{wr}
	−80	−2.42	N_{wr}
	−40	−1.21	NS_{wr}
	0	0	Z_{wr}
	40	1.21	PS_{wr}
	80	2.42	P_{wr}
	120	3.64	PM_{wr}
Acceleration	—	16.8	PB_{wr}

(Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

For the acceleration/deceleration stage, the slip frequencies are larger, so that their universe of discourse is defined as $F'_{wr} = \{PB_{wr}, NB_{wr}\}$. For the steady-state stage, the slip frequencies are smaller so that their universe of discourse is defined as $F''_{wr} = \{PM_{wr}, PS_{wr}, P_{wr}, Z_{wr}, N_{wr}, NS_{wr}, NM_{wr}\}$. The universe of discourse of the slip frequency is $F_{wr} = \{F'_{wr}, F''_{wr}\}$, and $\mu_{(wr)}$ denotes the degree of membership of slip frequency. Using the slip frequency values given in Table 6.2, the slip fuzzy membership functions are defined using the triangular distribution as shown in Figure 6.6.

6.3.2 *Fuzzy Rulebase*

For induction motors from 3 to 50 hp, the torque computed using the steady-state equivalent circuit is approximately equal to the average of the transient torque given by

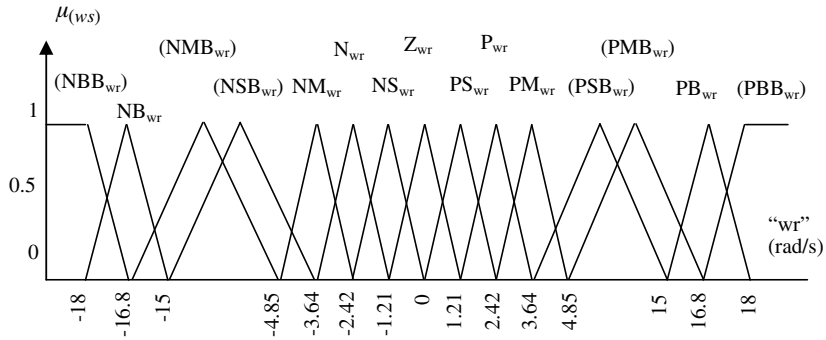


Figure 6.6 Membership function of slip frequency.

the free-acceleration torque-speed characteristic (Krause, Wasynczuk and Sudhoff, 1995). Accordingly the steady-state characteristics are used in the control formulations.

The torque-speed curves in Figure 6.7 illustrate the four operating states of an induction motor. When the induction motor is operated in regions I or IV, it works in the normal motoring mode. When the induction motor is operated in regions II or III, it works in the generating mode. The slip speed ω_r of the induction motor may be expressed as:

$$\omega_r = \omega - \frac{P}{2}\omega_o. \tag{6.15}$$

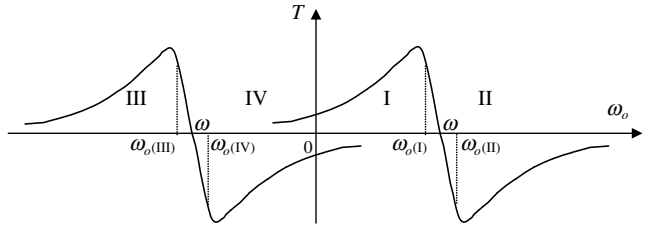


Figure 6.7 Torque-speed curves of an induction motor in four operating states. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

From Figure 6.7, it is seen that $T(\omega_o(\mathrm{I})) > 0$, $T(\omega_o(\mathrm{II})) \leq 0$, $T(\omega_o(\mathrm{IV})) \leq 0$, and $T(\omega_o(\mathrm{III})) > 0$. In order to obtain larger acceleration/deceleration torque, when $\omega_o^* > 0$ and $\omega_o^* - \omega_o > 0$ (operating state I), or when $\omega_o^* \leq 0$ and $\omega_o^* - \omega_o > 0$ (operating state III), the instantaneous slip frequency command ω_r should have a positive value and its fuzzy linguistic value is PB$_{wr}$. When $\omega_o^* > 0$ and $\omega_o^* - \omega_o < 0$ (operating state II), or when $\omega_o^* \leq 0$ and $\omega_o^* - \omega_o < 0$ (operating state IV), the instantaneous slip frequency command ω_r should have a negative value and its fuzzy linguistic value is NB$_{wr}$. The slip frequency control rules for the acceleration/deceleration stage may be summarized in Table 6.3.

Table 6.3 Slip speed control rules for the acceleration/deceleration stage.

Control Conditions	$\omega_o^* > 0$ $(\omega_o^* - \omega_o) > 0$	$\omega_o^* > 0$ $(\omega_o^* - \omega_o) < 0$	$\omega_o^* \leq 0$ $(\omega_o^* - \omega_o) > 0$	$\omega_o^* \leq 0$ $(\omega_o^* - \omega_o) < 0$
Operation States	I	II	III	IV
Action Rules	ω_r is PB_{wr}	ω_r is NB_{wr}	ω_r is PB_{wr}	ω_r is NB_{wr}

(Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

Let 'wo' denote the speed command ω_o^*, 'Dwo' denote the speed error $(\omega_o^* - \omega_o)$, and 'wr' denote the slip frequency ω_r. With reference to the control rules of the acceleration/deceleration stage in Table 6.3 and the steady-state control rules in Table 6.2, 21 slip frequency control rules are formulated and input to the fuzzy rulebase as shown in Table 6.4.

Table 6.4 Fuzzy rulebase of slip frequency control.

Dwo \ wo	NB_{wo}	NM_{wo}	NS_{wo}	Z_{wo}	PS_{wo}	PM_{wo}	PB_{wo}
N_{Dwo}	NB_{wr}	NB_{wr}	NB_{wr}	NB_{wr}	NB_{wr}	NB_{wr}	NB_{wr}
Z_{Dwo}	NM_{wr}	N_{wr}	NS_{wr}	Z_{wr}	PS_{wr}	P_{wr}	PM_{wr}
P_{Dwo}	PB_{wr}	PB_{wr}	PB_{wr}	PB_{wr}	PB_{wr}	PB_{wr}	PB_{wr}

(Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

6.3.3 Fuzzy Inference

The fuzzy inference includes a linguistic inference and a degree of membership calculation which gives the degree of membership of the output variable according to the rules. The inputs of the linguistic inference consist of two fuzzy linguistic values $F_{(wo)i}$ and $F_{(Dwo)j}$, and the output is $u(k)$ (crisp fuzzy linguistic value of slip frequency). The inputs of the degree of membership calculation consists of two degrees of membership $\mu_{(wo)s}$ and $\mu_{(Dwo)t}$, and the output is $\mu_{(wr)}(y_k)$ (degree of slip frequency membership).

6.3.3.1 Linguistic Inference

Using the 21 rules of fuzzy rulebase in Table 6.4, the linguistic inference of the fuzzy linguistic values can be expressed as:

$$\text{IF} F_{(wo)i} \text{ AND } F_{(Dwo)j}, \text{ THEN } F_{(wr)k} \tag{6.16}$$

where $F_{(wo)i} \in \{NB_{wo}, NM_{wo}, NS_{wo}, Z_{wo}, PS_{wo}, PM_{wo}, PB_{wo}\}$, $F_{(Dwo)j} \in \{N_{Dwo}Z_{Dwo}P_{Dwo}\}$,

and $F_{(wr)k} \in \{NB_{wr}, NM_{wr}, N_{wr}, NS_{wr}, Z_{wr}, PS_{wr}, P_{wr}, PM_{wr}, PB_{wr}\}$. (6.17)

The crisp value of fuzzy linguistic $F_{(wr)k}$ is:

$$u(k) = u(F_{(wr)k}) \tag{6.18}$$

where $u(k) \in \{-16.8\ -3.64, -2.42, -1.21, 0, 1.21, 2.42, 3.64, 16.8\}$.

6.3.3.2 Degree of Membership Calculation

The MAX-MIN principle (Bose, 1997) is adopted in the degree of membership calculation. If the input signals are x_1 (speed command) and x_2 (speed error), then the degree of membership of speed command is $\mu_{(wo)}(x_1)$, and the degree of speed error is the $\mu_{(Dwo)}(x_2)$. If the output signal is y, then the degree of membership of slip frequency is $\mu_{(wr)}(y)$.

Because the operation between '$wo =$' and 'Dwo' is 'AND' according to (6.16), the 'MIN' (Intersection) operation is used. For any inference rule, the degree of membership of output 'wr' is $\mu_{(wr)}$ and the degree of membership calculation of the slip frequency can be expressed as:

$$\mu_{(wr)q} = \text{MIN}[\mu_{(wo)s}(x_1),\ \mu_{(Dwo)t}(x_2)] \tag{6.19}$$

where $s \in \{\text{NB}_{\text{wo}}, \text{NM}_{\text{wo}}, \text{NS}_{\text{wo}}, \text{Z}_{\text{wo}}, \text{PS}_{\text{wo}}, \text{PM}_{\text{wo}}, \text{PB}_{\text{wo}}\}$, $t \in \{\text{N}_{\text{Dwo}}\ \text{Z}_{\text{Dwo}}\ \text{P}_{\text{Dwo}}\}$, and $q \in \{\text{NB}_{\text{wr}}, \text{NM}_{\text{wr}}, \text{N}_{\text{wr}}, \text{NS}_{\text{wr}}, \text{Z}_{\text{wr}}, \text{PS}_{\text{wr}}, \text{P}_{\text{wr}}, \text{PM}_{\text{wr}}, \text{PB}_{\text{wr}}\}$.

Because the logic relationships between the 21 fuzzy rules are 'OR' in Table 6.4, the degree of membership of output y is calculated by 'MAX' (Union) operation.

$$\mu_{(wr)}(y_k) = \text{MAX}(\mu_{(wr)p|p=1,2,\ldots 21}) \tag{6.20}$$

6.3.4 *Defuzzification*

After the fuzzy inference, the fuzzy output needs to be converted into a crisp value required by the inverter control. The inputs of defuzzification are $u(k)$ and $\mu_{(wr)}(y_k)$, which are the outputs of fuzzy inference. The output is the numerical value of slip frequency, y. In this design, the centroid method is used to implement the defuzzification operation (Kosko, 1997).

$$y = \frac{\sum_{k=1}^{n} u(k)\mu_{(ws)}(y_k)}{\sum_{k=1}^{n} \mu_{(ws)}(y_k)} \tag{6.21}$$

6.3.5 *Fuzzy Frequency Controller*

Simulation of the frequency controller may be implemented using the software MATLAB®/Simulink with Fuzzy Logic Toolbox as shown in Figure 6.8. The inputs of the controller are the speed command and the rotor speed. The output is the frequency command to the inverter.

The controller has three characteristics: (1) its supply frequency is almost the same as field-oriented control, hence it can maintain a large acceleration/deceleration torque in the four

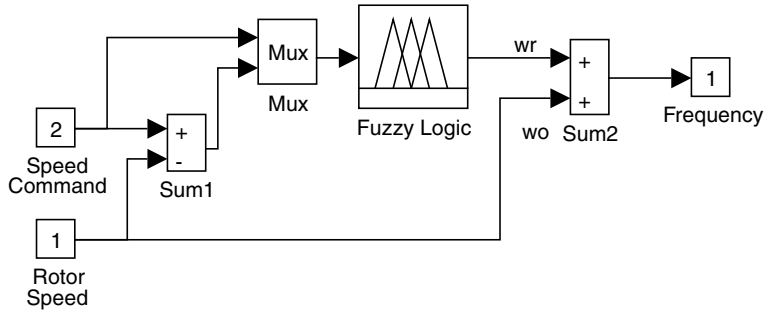


Figure 6.8 Simulink blocks of frequency controller. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

operating states; (2) it is insensitive to motor parameter changes; (3) the speed response over the whole speed range, even subjected to large noise and load disturbances, is very satisfactory.

6.4 Current Magnitude PI Control

During the acceleration/deceleration stage, the stator current magnitude is regulated as the maximum permissible value of the inverter drive system. During the final steady-state period, the supply frequency is maintained constant, while the stator current magnitude is adjusted to control the rotor speed. When the supply frequency is fixed, the torque-current relationship may be expressed as Equation (6.22) (Krause, Wasynczuk and Sudhoff, 1995) and is illustrated in Figure 6.9.

$$T = \frac{3P}{2} \cdot \frac{R_r \left(\omega - \frac{P}{2}\omega_o\right) L_m^2 I_s^2}{{R_r}^2 + \left(\omega - \frac{P}{2}\omega_o\right)^2 (L_m + L_{lr})^2} \text{ (N·m)} \tag{6.22}$$

Figure 6.9 shows the process of controlling the speed ω_o by stator current magnitude according to the speed error. The decaying oscillations of speed about the final operating point have been eliminated.

The following proportional-and-integral control with output saturation is used in the nonlinear control.

$$I_s = K_p(\omega_o{}^* - \omega_o) + K_I \int (\omega_o{}^* - \omega_o)\, dt \tag{6.23}$$

$$|I_s| = \begin{cases} |I_s| & |I_s| < 50 \text{ A} \quad \text{steady-state stage} \\ 50 & |I_s| \geq 50 \text{ A} \quad \text{acceleration stage} \end{cases} \tag{6.24}$$

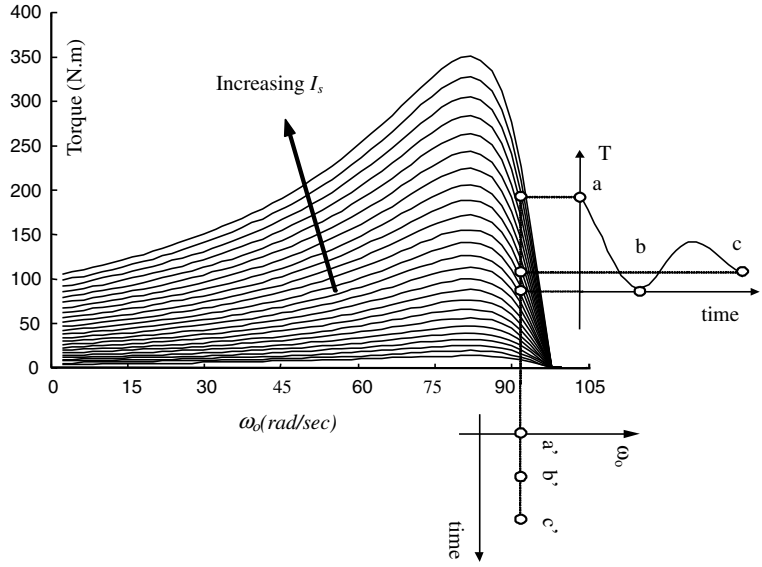


Figure 6.9 Speed controlled by stator current magnitude. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

When a PI controller is used to control the nonlinear system, it must be tuned very conservatively in order to provide stable behavior over the entire range of operating conditions (Henson and Seborg, 1997). In this study, coordination of the current magnitude control for the two stages is achieved using the following strategy:

1. When $|\omega_o^* - \omega_o| \geq 3$, the current magnitude control changes from a steady-state stage to an acceleration stage.
2. When $2.5 < |\omega_o^* - \omega_o| < 3$, the control pattern is unchanged.
3. When $|\omega_o^* - \omega_o| \leq 2.5$, the current magnitude control changes from an acceleration stage to a steady-state stage.

During the steady-state stage, stator current I_s has a small value. When $|\omega_o^* - \omega_o| \geq 3$, the current magnitude control changes from the steady-state stage to an acceleration stage. If we let $K_p(\omega_o^* - \omega_o) > 50$, that is, $K_p > 16.6$, then I_s of Equation (6.23) is larger than 50 A, and hence the proportional coefficient K_p may be chosen as 17.

The maximum value of the integral part of Equation (5.63) can be estimated from an acceleration process with $\omega_o^* = 120$, $\omega_o(t = 0\,\text{s}) = 0$, assuming that the rotor speed rises at uniform acceleration and $\omega_o(t = 0.2\,\text{s}) \approx 120$, that is,

$$\int_{\tau=0}^{0.2} (\omega_o^* - \omega_o) d\tau \approx 12 \tag{6.25}$$

When the current magnitude control changes from an acceleration stage to a steady-state stage, substituting $(\omega_o^* - \omega_o) = 2.5$, $K_p = 17$, and Equation (6.25) into Equation (6.23), and let $I_s < 50$, then $K_I < 0.625$. Therefore the integral coefficient K_I is chosen as 0.6.

A current magnitude controller can be designed as shown in Figure 6.10.

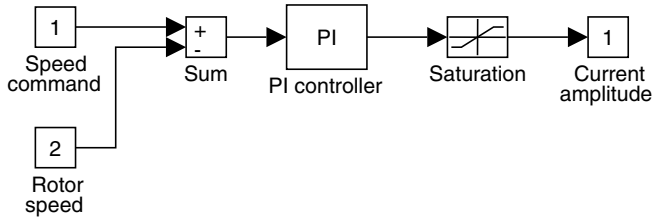


Figure 6.10 Simulink blocks of current magnitude controller. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

The current magnitude controller consists of a sum block to calculate speed error, a PI block to implement Equation (6.23), and a saturation block to implement Equation (6.24). During acceleration or deceleration stage, there is a larger difference between the speed command and the rotor speed. Consequently, the PI controller has a larger output. Due to the function of the saturation block, the supply current amplitude is maintained constant during acceleration or deceleration stage. During the steady-state stage, there is a smaller difference between speed command and rotor speed. Consequently, the output of the PI controller is smaller than the limit of the saturation block and the rotor speed is controlled by the current amplitude.

6.5 Hybrid Fuzzy/PI Two-Stage Controller for an Induction Motor

By combining the fuzzy frequency controller and the current magnitude PI controller, a hybrid fuzzy/PI two-stage controller can be formed. During the acceleration/deceleration period, the current magnitude controller outputs the maximum permissible current. During the final steady-state period, the fuzzy frequency controller outputs the frequency that corresponds to the speed command. The model of two-stage speed controller for the induction motor is constructed using MATLAB®/Simulink as shown in Figure 6.11. The current-input model of induction motor in Chapter 3 is employed.

The simulation model of the two-stage controller consists of the induction motor model, the frequency control sub-model (Figure 6.8), the current magnitude control sub-model (Figure 6.10), a load block, a command block, and a scope sink for display of rotor speed. In addition, three scope sinks are configured inside the sub-models for observing the current, slip frequency, and torque. The load block implements the load function of Equation (6.12).

6.6 Simulation Study on a 7.5 kW Induction Motor

Computer simulations were performed on the fuzzy/PI two-stage controller shown in Figure 6.11 for a 7.5 kW induction motor. Five investigations were undertaken: (1) comparison

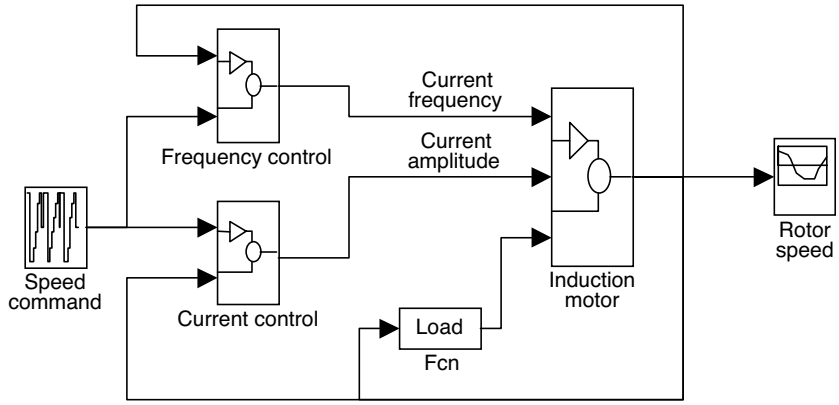


Figure 6.11 Simulink blocks of the hybrid fuzzy/PI control system. (Reproduced by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Hybrid fuzzy two-stage controller for an induction motor," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1898–1903, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE.)

with field-oriented control, (2) effects of parameter variation, (3) effects of noise in the measured speed and input current, (4) effects of magnetic saturation, and (5) effects of load torque variation.

The parameters of the 7.5 kW induction motor chosen for the simulation studies are listed in 'Motor 1' of Appendix B. It is assumed that the induction motor is taken through the following control cycle:

Speed Command	Period
$\omega_o^* = 120$ rad/s	$0\,\text{s} \le t < 4\text{s}$
$\omega_o^* = -120$ rad/s	$4\,\text{s} \le t < 8\text{s}$
$\omega_o^* = 120$ rad/s	$8\,\text{s} \le t < 12\text{s}$
$\omega_o^* = 0$ rad/s	$12\,\text{s} \le t < 15\text{s}$

Permissible magnitude of stator current of the induction motor is 50 A. The moment of inertia, J_L, of the load equals that of the motor.

6.6.1 *Comparison with Field-Oriented Control*

Figures 6.12, 6.14, 6.16 and 6.18 show the simulation results of the hybrid fuzzy/PI controller. Very fast speed response is obtained with the two-stage control method. Due to the current control in the final steady-state stage, the oscillations of speed about the final operating point are completely eliminated.

In order to compare the new controller with a field-oriented controller, an indirect rotor flux field-oriented controller was investigated. In the computer simulation, the acceleration torque $T^* = 100$ Nm, the rotor flux command $\lambda_{dr}^{e\,*} = 0.67$ Wb, $R_r = 0.151\,\Omega$, $L_M = 0.042$ H/ph and

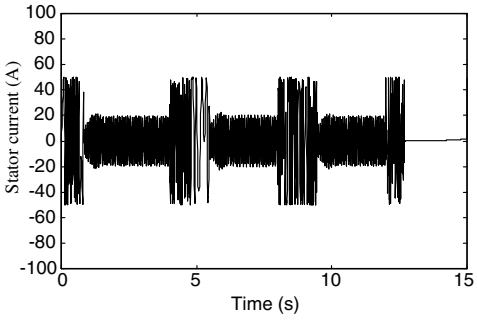


Figure 6.12 Stator current of fuzzy controller.

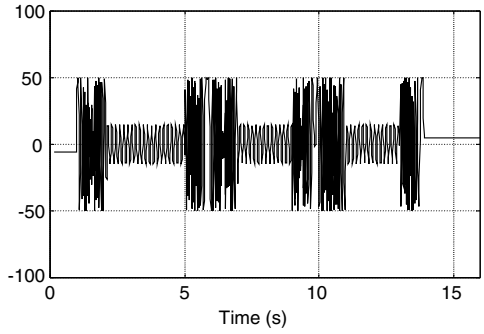


Figure 6.13 Stator current of FOC controller.

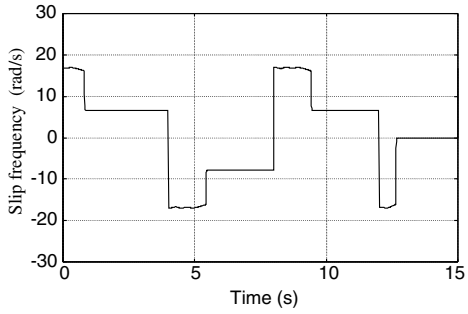


Figure 6.14 Slip frequency of fuzzy controller.

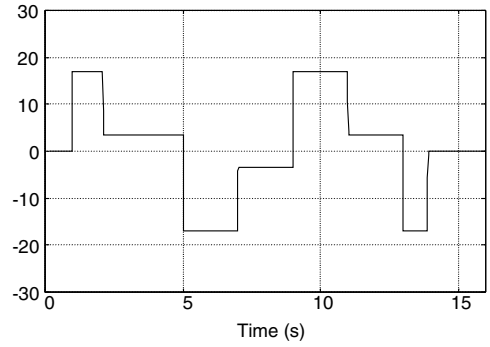


Figure 6.15 Slip frequency of FOC controller.

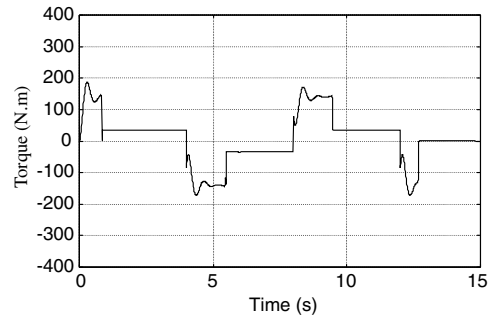


Figure 6.16 Torque response of fuzzy controller.

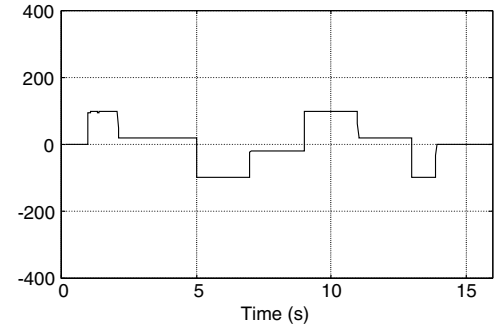


Figure 6.17 Torque response of FOC controller.

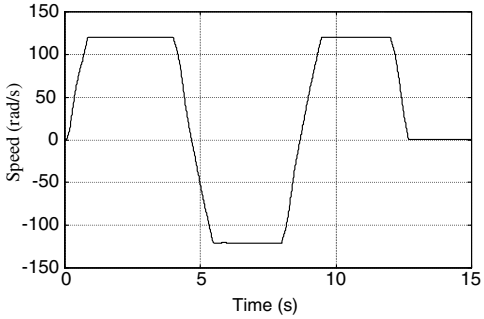


Figure 6.18 Speed response of fuzzy controller.

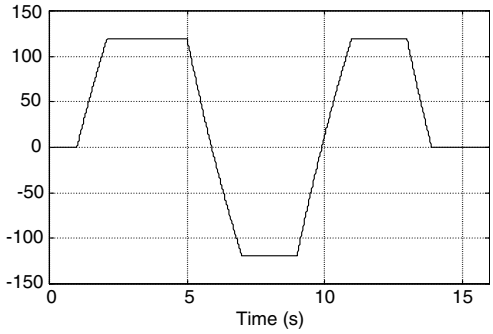


Figure 6.19 Speed response of FOC controller.

$k_q = 2$, hence the magnitude of phase current is about 50 A from Equation (6.7) (same as that for the hybrid fuzzy/PI controller), while the slip frequency is 16.8 rad/s from (6.13). Figures 6.13, 6.15, 6.17, and 6.19 show the simulation results of the indirect FOC controller.

Since the hybrid fuzzy/PI two-stage controller has almost the same current and slip frequency responses as FOC, it has approximately the same torque response, although in the former case, the torque has not been directly controlled. Consequently, the speed response of the new controller is almost the same as the field-oriented controller (Figures 6.18 and 6.19). But the torque oscillations and the transient torque peaks (Figure 6.16) may increase the mechanical stress of the motor shaft.

6.6.2 *Effects of Parameter Variation*

In order to illustrate the insensitivity of the hybrid fuzzy/PI two-stage controller to the variation of motor parameters, the rotor resistance and the mutual inductance are assumed to be changed to $2R_r$ and $0.7L_M$ respectively. Figure 6.20 shows the stator current of the induction motor drive, while Figures 6.21 and 6.22 show that the torque and speed responses of the fuzzy controller are insensitive to parameter variations.

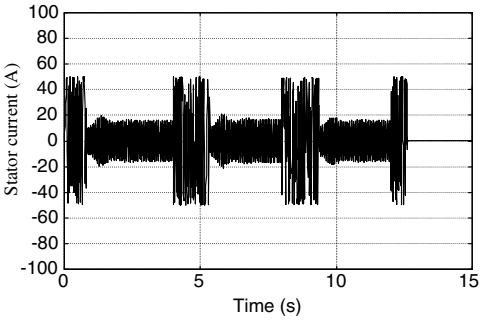


Figure 6.20 Stator current of the induction motor drive with parameter variations.

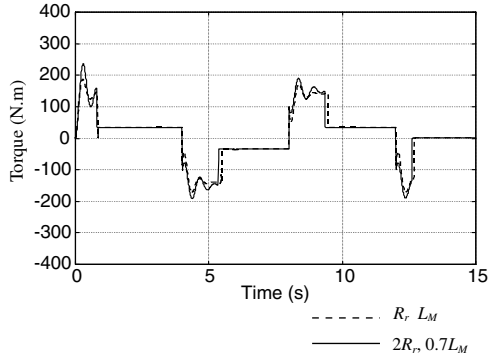


Figure 6.21 Torque response of fuzzy controller with parameter variations.

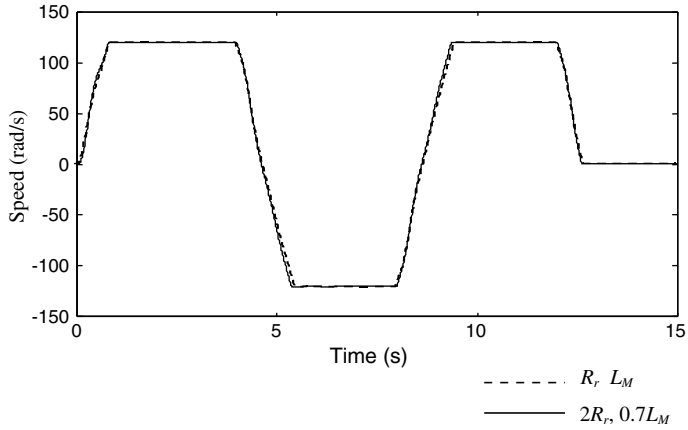


Figure 6.22 Speed response of fuzzy controller with parameter variations.

6.6.3 *Effects of Noise in the Measured Speed and Input Current*

In order to evaluate the effects of the noise of speed sensor and the noise of the input current, distributed random noises are added into the feedback speed and input current. The simulation is achieved using the random number blocks of Simulink, which generate a pseudo-random, normally distributed (Gaussian) number (Simulink, 1994). The speed response with the measured speed noise (mean of zero and variance of 3) and in the current noise (mean of zero and variance of 10) shows that the hybrid fuzzy/PI two-stage controller has good disturbance rejection (Figure 6.23).

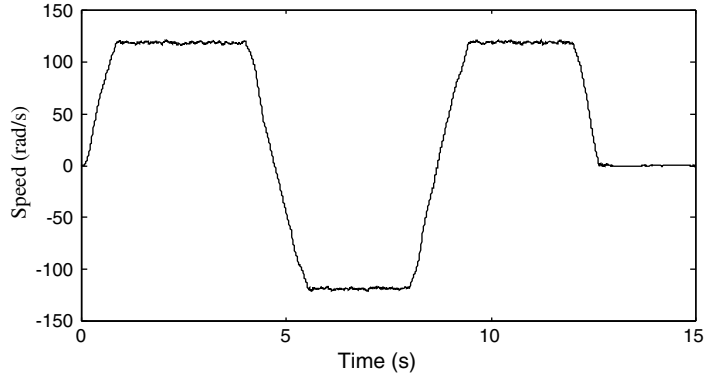


Figure 6.23 Rotor speed response with current noise and measured speed noise.

6.6.4 *Effects of Magnetic Saturation*

In order to study the effect of magnetic saturation of the induction motor on the controller performance, two saturation blocks are included in the induction motor model to simulate the

nonlinear relationship between the current and flux, where λ_{dr}^{e} and λ_{qr}^{e} are assumed to be less than 0.8 Wb due to saturation. Figures 6.24–6.26 show the simulation results of the rotor speed, stator phase current and torque responses. As a result of magnetic saturation, flux increase is limited so that the torque oscillations are reduced significantly, but excessive magnetic saturation during the acceleration/deceleration stage will produce larger losses and a higher temperature rise.

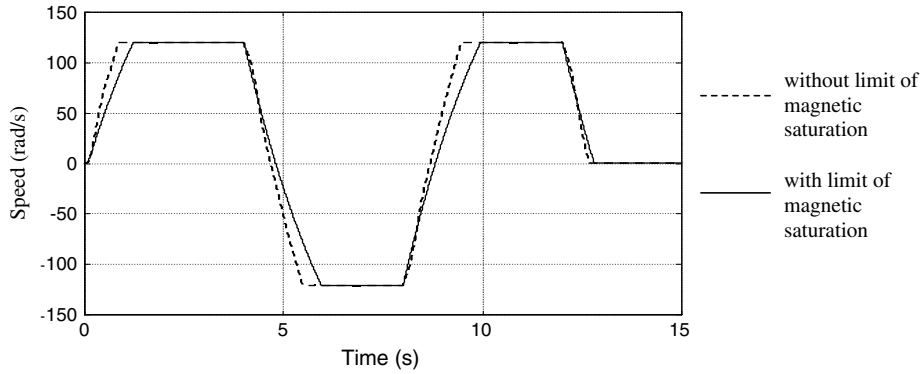


Figure 6.24 Magnetic saturation effect on speed response of fuzzy controller.

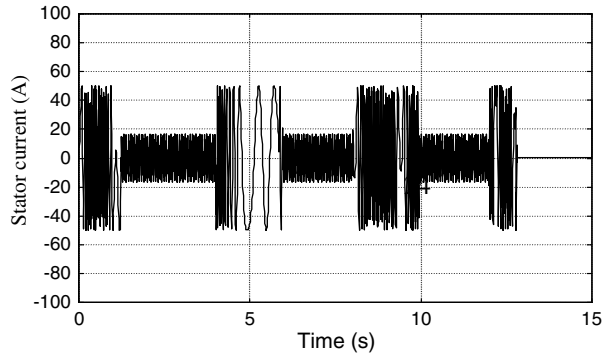


Figure 6.25 Magnetic saturation effect on stator current of fuzzy controller.

6.6.5 *Effects of Load Torque Variation*

The simulation will investigate effects of load torque variation on the hybrid fuzzy/PI control system. The load torque variation may be implemented by changing parameters of the load function block in the simulation program (see Figure 6.11). In the simulation, the control system experiences sudden changes in the load torque: at $t = 2.5$ s, the load increases from 100% to 200% of the rated torque, T_L, at $t = 7$ s, the load decreases to 100% of T_L, and at $t = 10$ s, the load increases to 200% of T_L again. Figure 6.27 shows the rotor speed of the fuzzy

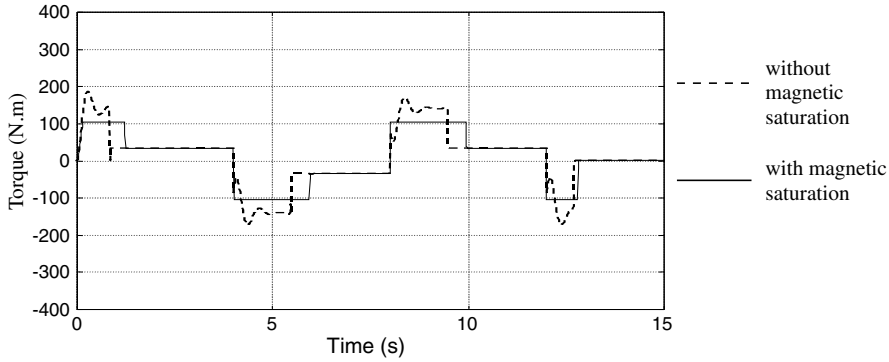


Figure 6.26 Magnetic saturation effect on torque response of fuzzy controller.

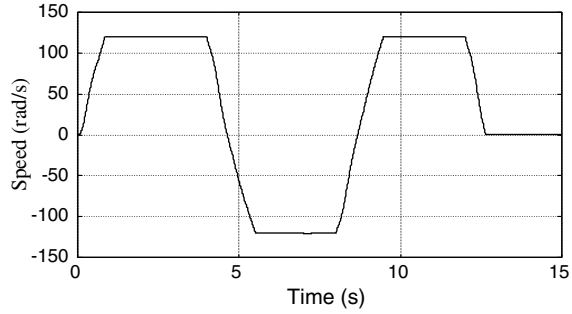


Figure 6.27 Rotor speed of the fuzzy control system with the load torque variation.

control system with load torque variation. At about 2.5 s, following the load changes, the speed, initially at 120.08 rad/s, drops to 119.94 rad/s, but is restored to 120.08 rad/s in 0.2 s.

Figures 6.28–6.30 show the stator current, slip frequency, and torque response with load torque variation.

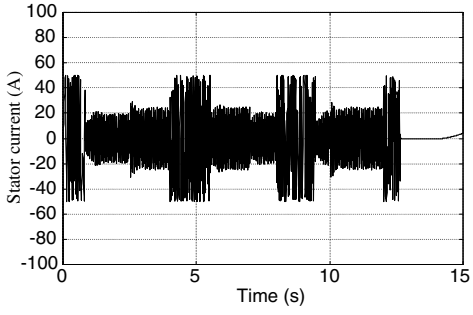


Figure 6.28 Stator current with load torque variation.

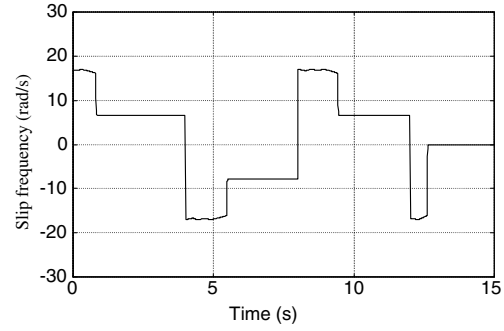


Figure 6.29 Slip with load torque variation.

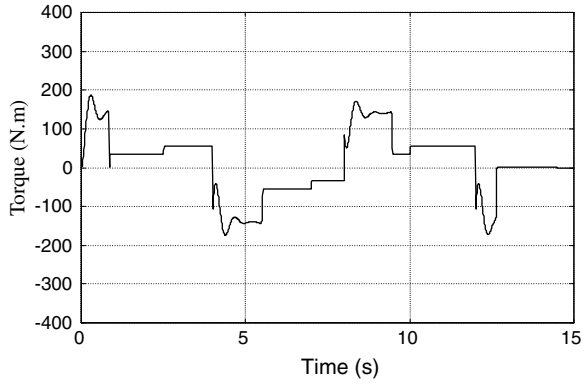


Figure 6.30 Torque response with load torque variation.

The above simulation results show that the fuzzy/PI controller can accommodate larger changes in load torque.

6.7 Simulation Study on a 0.147 kW Induction Motor

The fuzzy/PI controller has been simulated using the current-input model of the induction motor. In this section, the voltage-input model of an induction motor built in Chapter 3 is used in simulation studies on a 0.147 kW induction motor (Bodine Electric Company model 295) for the fuzzy/PI two-stage controller. Parameters of the 0.147 kW induction motor for the simulation studies are listed in 'Motor 3' of Appendix B. The simulation program of fuzzy/PI two-stage control system is shown in Figure 6.31. The current frequency and magnitude are transformed to stator current commands (i_a^*, i_b^*, i_c^*) by the '3-phase current' block which is described in Figure 3.2 in Chapter 3. The stator voltages (V_a, V_b, V_c) are produced by three 'PI' blocks of stator-current control with difference of the stator current commands (i_a^*, i_b^*, i_c^*) and the actual stator currents (i_a, i_b, i_c) as input.

The load characteristic of Equation (6.12) is assumed with the coefficient $\mu = 0.00\,532$ Nm/(rad/s). When the rotor speed is 188 rad/s, the load torque equals 1 Nm.

Because the pole number of the induction motor is 4, the normal range of speed (ω_o) is from -188 rad/s to 188 rad/s. The membership function of speed command has to be designed again and is shown in Figure 6.32.

When the range of speed command (ω_o^*) is from -188 rad/s to 188 rad/s, the range of the speed error ($\Delta\omega_o$) is from -376 rad/s to 376 rad/s. The membership function of speed error is designed as shown in Figure 6.33.

For the 0.147 kW induction motor, $R_r = 12.76\,\Omega$, $P = 4$, and $T_{(\text{acceleration})} = 1.3$ Nm. If $\lambda_{dr}^{e\,*} = 0.8$ Wb during the acceleration stage, then $\omega_r = 19.53$ rad/s from Equation (6.3). For the steady-state stage, the value of steady-state slip frequency can be calculated from $s = 0.03$.

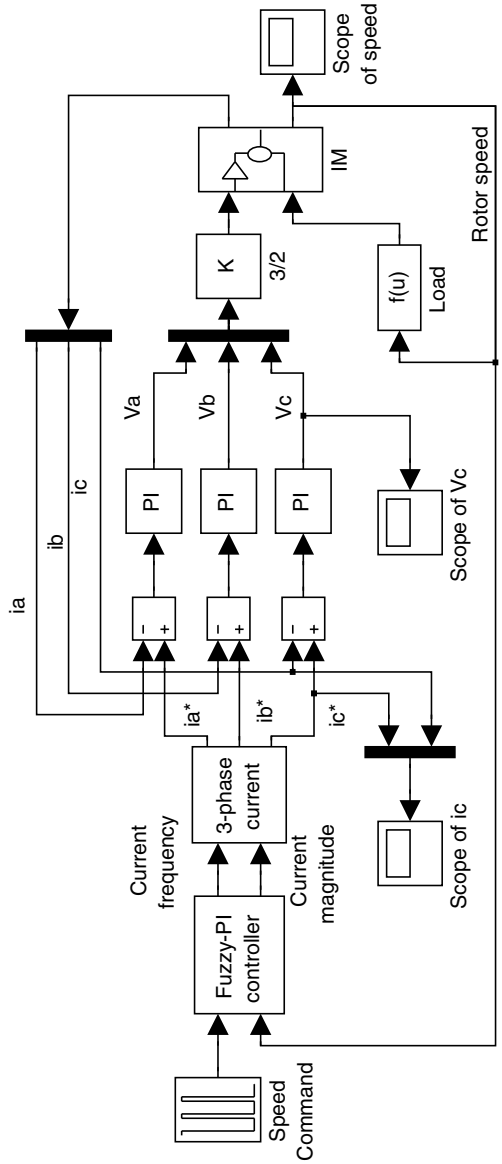


Figure 6.31 Simulink program of fuzzy/PI control system with voltage-input model of induction motor. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A novel hybrid fuzzy/PI two-stage controller for an induction motor drive," IEEE International Electric Machines and Drives Conference (IEMDC 2001), pp. 415–421, June 17–20, 2001, Cambridge, MA, U.S.A. © 2001 IEEE.)

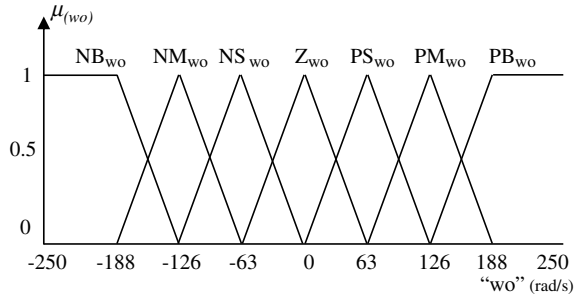


Figure 6.32 Membership function of speed command. NB: negative big; NM: negative medium; NS: negative small; Z: zero; PB: positive big; PM: positive medium; PS: positive small.

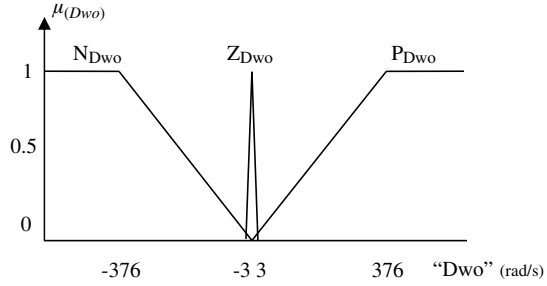


Figure 6.33 Membership function of speed error. N: negative; Z: zero; P: positive.

The speed command, slip frequency, and fuzzy linguistic values of the membership functions of slip frequency are shown in Table 6.5.

When permissible magnitude of stator current of the induction motor is assumed as 3 A, the proportional and integral parameters in the 'fuzzy-PI controller' block in Figure 6.31 may be

Table 6.5 Speed, slip frequency, and fuzzy linguistic values.

Stages	ω_o^*	ω_r	F_{wr}
Deceleration	—	−19.53	NB_{wr}
Steady state	−188	−5.64	NM_{wr}
	−80	−2.4	N_{wr}
	−40	−1.2	NS_{wr}
	0	0	Z_{wr}
	40	1.2	PS_{wr}
	80	2.4	P_{wr}
	188	5.64	PM_{wr}
Acceleration	—	19.53	PB_{wr}

designed according to Equations (6.23), (6.24), and (6.25) as:

$$K_p = 1$$

$$K_I = 0.03$$

The three PI controllers of stator currents in the 'PI' blocks in Figure 6.31 would force the error between the stator current commands and actual stator currents to zero. The inner current loops were tuned with constant stator current command. (i.e. with the fuzzy-PI controller disabled). Once satisfactory results were obtained, the fuzzy-PI controller was tuned.

The transfer function of the PI controller of stator currents is represented by

$$G_{PI}(s) = K_p + \frac{K_I}{s}. \tag{6.26}$$

The controller has a zero at:

$$s = -\frac{K_I}{K_p} \tag{6.27}$$

and a pole at the origin.

$$\text{Let} \quad \sigma = 1 - \frac{L_m^2}{L_s L_r} \tag{6.28}$$

Substituting Equations (6.28) and (3.10) into Equation (3.11) gives:

$$V_e^s = R_s i_s^e + \sigma L_s \frac{d}{dt} i_s^e + \frac{L_m}{L_r} \frac{d}{dt} \lambda_r^e - j\omega \lambda_s^e. \tag{6.29}$$

In this equation, the stator voltage is represented by four terms. The last two can be considered as disturbance terms and can be neglected in the PI controller design. The first two terms represent the plant to be controlled.

The transfer function of the plant is represented by:

$$G_{\text{motor}} = \frac{1}{\sigma L_s s + R_s} \tag{6.30}$$

which has a pole at:

$$s = -\frac{R_s}{\sigma L_s}. \tag{6.31}$$

To give a critically damped response, the ratio of K_p and K_I can be chosen to place the controller zero at the plant pole:

$$\frac{K_I}{K_p} = \frac{R_s}{\sigma L_s}. \tag{6.32}$$

The open-loop transfer function of the PI controller of stator currents and the motor can be expressed as:

$$G_{\text{openloop}} = G_{\text{PI}} \times G_{\text{motor}} = K_p \frac{1}{\sigma L_s s}. \tag{6.33}$$

Making the following substitution:

$$K = K_p \frac{1}{\sigma L_s} \tag{6.34}$$

gives the following equation:

$$G_{\text{openloop}} = \frac{K}{s}.$$

Closing the feedback loop with unity gain results in the following closed-loop transfer function:

$$G_{\text{closedloop}} = \frac{K}{s+K}. \tag{6.35}$$

This is recognized to be a single-pole low pass filter with 3-dB corner frequency at:

$$F_{3dB} = \frac{K}{2\pi}. \tag{6.36}$$

Choosing the simulation controller bandwidth of 100 Hz results in:

$$K = 2\pi \times 100. \tag{6.37}$$

K_p can be obtained by substituting K into Equation (6.34):

$$K_p = 200\pi \times \sigma L_s. \tag{6.38}$$

From Appendix H and Equation (6.28), $\sigma = 0.208$ and $L_s = 0.3185$ H. Substituting these values into Equation (6.38),

$$\begin{aligned} K_p &= 200\pi \times 0.208 \times 0.3185 \\ K_p &= 42 \end{aligned}. \tag{6.39}$$

K_I can be obtained from Equation (6.32) with $\sigma = 0.208$, $L_s = 0.3185$ H, and $R_s = 14.6\,\Omega$:

$$\begin{aligned} K_I &= \frac{42 \times 14.6}{0.208 \times 0.3185}. \\ K_I &= 9256 \end{aligned} \tag{6.40}$$

It is assumed that the induction motor is taken through the following control cycle:

Speed Command	Period
$\omega_o^* = 180$ rad/s	$0\,\text{s} \leq t < 0.5\,\text{s}$
$\omega_o^* = 20$ rad/s	$0.5\,\text{s} \leq t \leq 1.5\,\text{s}$

Figure 6.34 shows that, when permissible magnitude of stator current of the induction motor is 3 A, the actual stator current i_c (dotted line) of the voltage-input model of the induction motor is able to track the stator current command i_c^* (solid line).

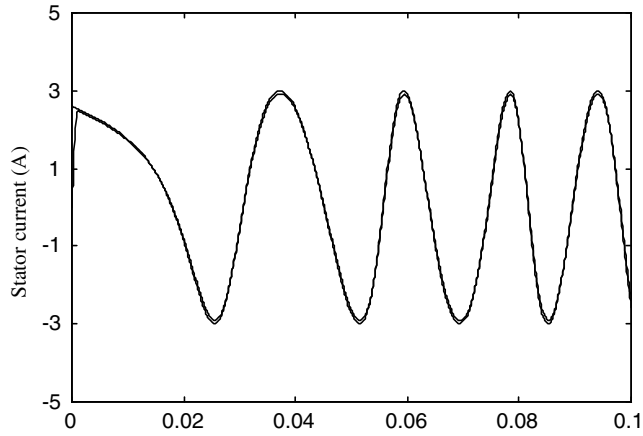


Figure 6.34 Actual stator current i_c (dotted line) and stator current command i_c^* (solid line).

Figure 6.35 shows controlled slip frequency and controlled stator current of the fuzzy/PI control system.

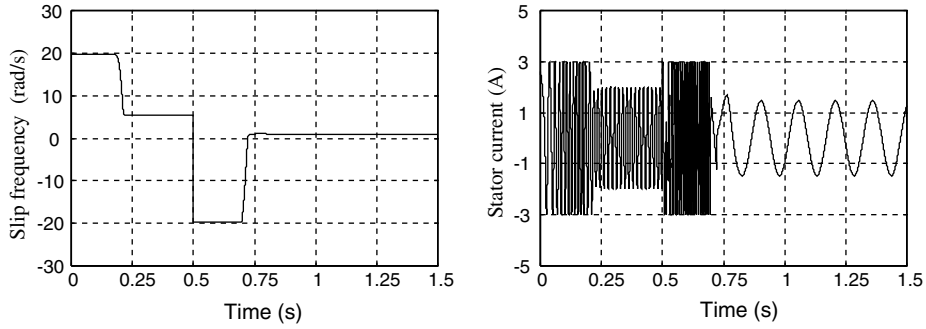


Figure 6.35 Controlled slip frequency and controlled stator current.

Figure 6.36 shows the phase voltage and rotor speed response of the voltage-input model of the 0.147 kW induction motor.

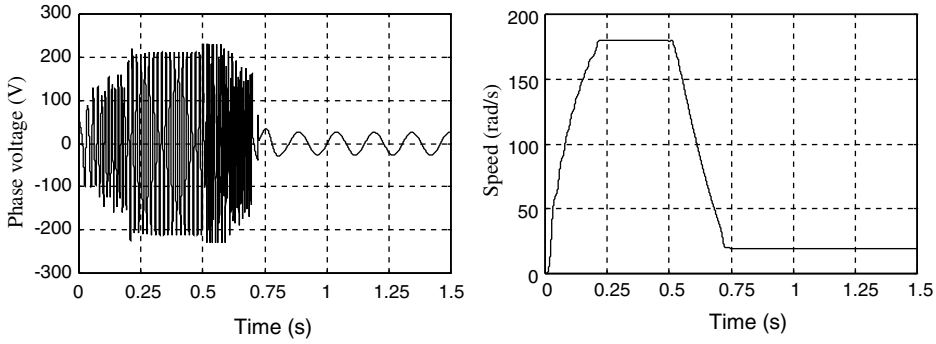


Figure 6.36 Phase voltage and rotor speed response of the voltage-input model of the 0.147 kW induction motor.

6.8 MATLAB®/Simulink Programming Examples

Two examples are given to illustrate programming of a fuzzy/PI two-stage controller using MATLAB®/Simulink. The voltage-input model of an induction motor is first developed, and it is then used in a fuzzy/PI two-stage controller for an induction motor.

6.8.1 Programming Example 1: Voltage-Input Model of an Induction Motor

This example demonstrates programming of the voltage-input model of a 0.147 kW induction motor, which is the 'IM' block in Figure 6.31. The electrical model of the induction motor is described by Equation (3.16) and the mechanical model is described by Equation (3.17).

Step 1 Implementing the Induction Motor Model

The induction motor model is implemented by using Simulink blocks as shown in Figure 6.37.

The inputs of the induction motor model are *dq*-axis stator voltages (V_{ds}, V_{qs}) and the load. The outputs of the model are the three-phase stator currents ($i_{as}i_{bs}i_{cs}$), rotor speed, and torque. In the induction motor model, the 'Matrix A' block is implemented by a 'MATLAB Function' block and the 'Matrix B' block is implemented by a 'Matrix Gain' block. The two matrix blocks in Equation (3.16) are used to simulate the electrical model of induction motor. The '2/3' block is implemented by a 'Matrix Gain' block, which simulates *dq*-axis to three-phase transformation described in Equation (3.5).

The mechanical model of the induction motor is simulated by the 'Mechanical model' block in Figure 6.37 and its details are shown in Figure 6.38.

The inputs of the mechanical model are stator currents, rotor currents, and torque. The outputs of the mechanical model are torque and rotor speed. The stator currents and rotor currents come from the electrical model in Figure 6.37. Calculation of the torque in the

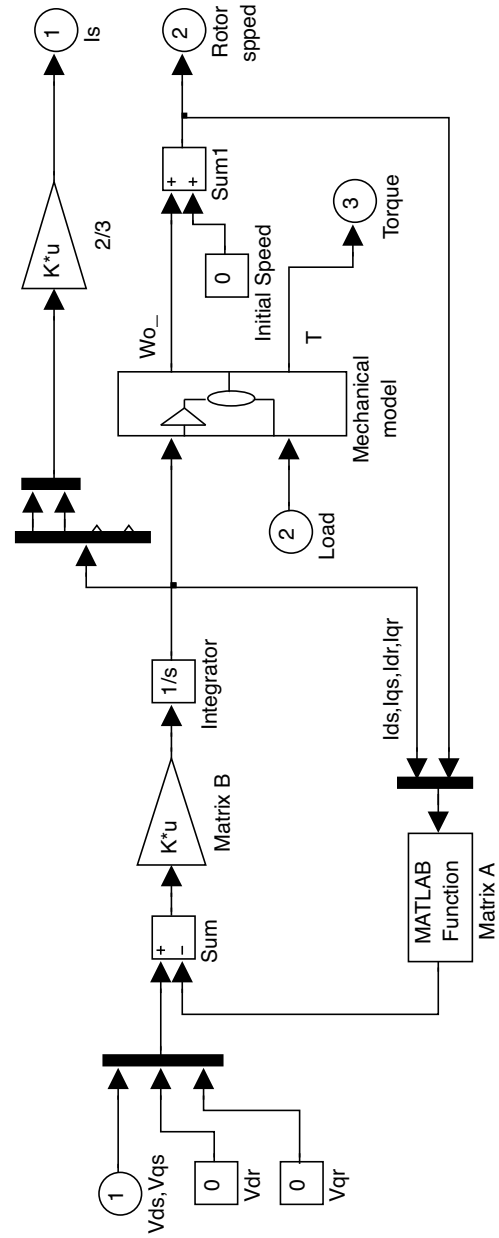


Figure 6.37 Voltage-input model of 0.147 kW induction motor.

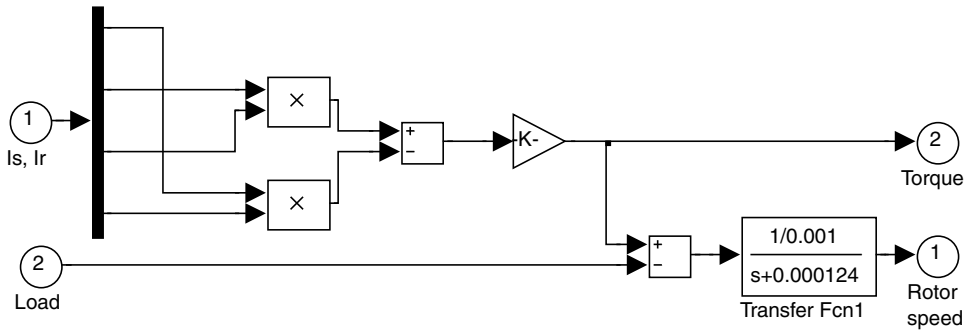


Figure 6.38 Mechanical model of induction motor.

mechanical model is based on Equation (3.17) and calculation of the rotor speed is based on Equation (3.12). The motor parameters ('Motor 3' in Appendix B) are given as follows.

0.147 kW induction motor (Bodine Electric Company model 295)

Stator resistance: $R_s = 14.6\,\Omega$/ph
Rotor resistance $R_r = 12.77\,\Omega$/ph
Stator inductance, $L_s = 0.3185$ H/ph
Mutual inductance, $L_M = 0.2963$ H/ph
Rotor inductance $L_r = 0.3482$ H/ph
Moment of inertia of the rotor $J_M = 0.001\,\text{kg m}^2$
Coefficient of friction $C_f = 0.000\,124$
Number of poles $P = 4$

Step 2 Configuring the Parameters of the Simulink Blocks

With the above electrical parameters of the induction motor, matrix A is as shown in Table 6.6, while matrix B is obtained by a matrix inverse according to Equation (3.16), as follows:

$$\text{B} = \text{inv}([0.3185, 0, 0.2963, 0; 0, 0.3185, 0, 0.2963; 0.2963, 0, 0.3482, 0; 0, 0.2963, 0, 0.3482])$$

Upon entering the above MATLAB® command, the matrix B is obtained:

$$B = \begin{bmatrix} 15.0684 & 0 & -12.8224 & 0 \\ 0 & 15.0684 & 0 & -12.8224 \\ -12.8224 & 0 & 13.7831 & 0 \\ 0 & -12.8224 & 0 & 13.7831 \end{bmatrix}$$

The functions and parameters of the main blocks of the induction motor are listed in Table 6.6.

Table 6.6 Function and parameters of the blocks of the 0.147 kW induction motor.

Block Name	Function	Parameters
Matrix A 'MATLAB Function' block	Implement matrix A in Equation (3.16)	[u(1)*14.6; u(2)*14.6; u(2)*u(5)* 2*0.2963 + u(3)*12.77 + u(4)*u(5)*2*0.3482; (−0.2963)*u(1)*u(5)*2 +(−0.3482)*u(3)*u(5)*2 + u(4)*12.77]
Matrix B 'Matrix Gain' block	Implement matrix B in Equation (3.16)	[15.0684 0 −12.8224 0; 0 15.0684 0 −12.8224; −12.8224 0 13.7831 0; 0 −12.8224 0 13.7831]
2/3 'Matrix Gain' block	Implement *dq*-axis to 3-phase transform in Equation (3.5)	[1 0; −1/2 sqrt(3)/2; −1/2 −sqrt(3)/2]
Transfer Fcn1 'Transfer Fcn' block	Calculate rotor speed in Equation (3.12)	Numerator coefficient: [1/0.001] Denominator coefficient: [1 0.000 124]
V_{dr}, V_{qr} 'Constant' blocks	Simulate rotor voltages	$V_{dr} = 0$; $V_{qr} = 0$.

Step 3 Adding a Voltage Source and the Load to the Induction Motor Model
For testing of the motor model, a 'dq-axis Voltage source' block and a 'Load' block are added to the motor model, as shown in Figure 6.39.

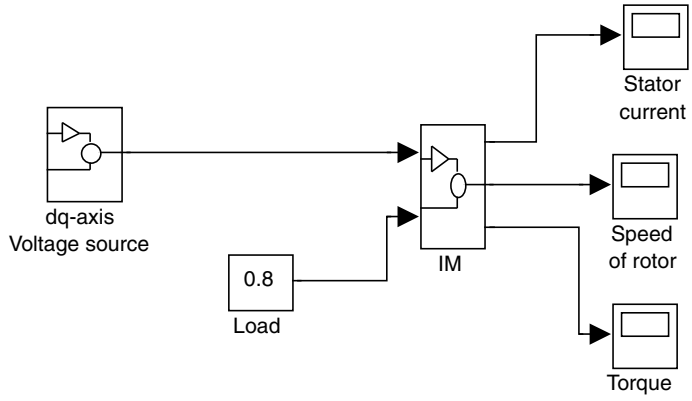


Figure 6.39 Test model of the 0.147 kW induction motor.

The 'Load' block with a constant value 0.8 simulates a load torque of 0.8 Nm. The 'dq-axis Voltage source' block yields the *dq*-axis voltages V_{ds} and V_{qs}, and its details are shown in Figure 6.40.

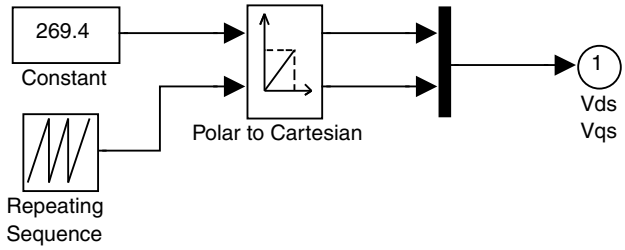


Figure 6.40 A *dq*-axis voltage source with output V_{ds}, V_{qs}.

The 'dq-axis Voltage source' block consists of a 'Constant' block (which contains the amplitude of the voltage source), a 'Repeating Sequence' block (which yields the phase angle of the voltage source), and a 'Polar to Cartesian' block (which transforms the amplitude and phase angle into *dq*-axis voltages V_{ds} and V_{qs}).

To simulate a voltage source with an amplitude of 269.4 V and a frequency of 60 Hz, the value 269.4 is input into the 'Constant' block and the 'Repeating Sequence' block is filled with time values of [0,0.001667] and an output value of [0,2*pi].

Step 4 Running the Simulink Model

The Simulink test model as shown in Figure 6.39 is run with following parameters.

Simulation type: variable-step
Max step = 0.0001
Min step = auto
Initial step = 0.0001
Simulation time = 0.5 s

The simulation results for rotor speed, torque, and three-phase stator currents are shown in Figures 6.41–6.43, respectively.

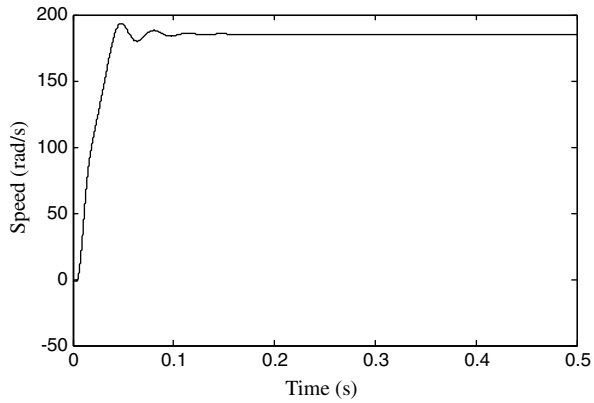


Figure 6.41 Rotor speed of the 0.147 kW induction motor.

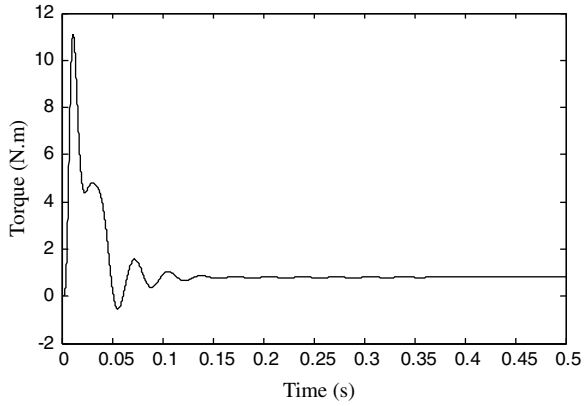


Figure 6.42 Torque of the 0.147 kW induction motor.

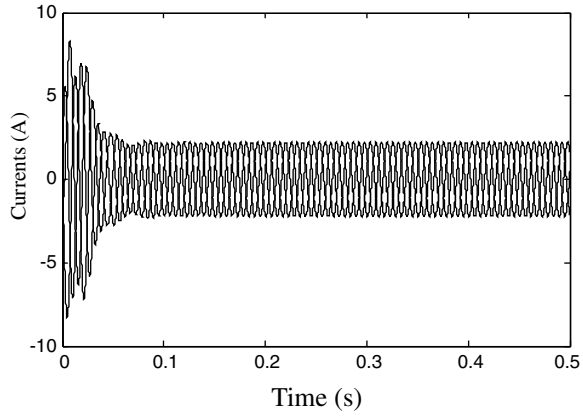


Figure 6.43 Three-phase stator currents of the 0.147 kW induction motor.

6.8.2 Programming Example 2: Fuzzy/PI Two-Stage Controller

This example illustrates programming of the fuzzy/PI two-stage controller for the 0.147 kW induction motor using MATLAB®/Simulink. The controller is the 'fuzzy-PI controller' block in Figure 6.31.

Step 1 Implementing a Simulink Model

A model of the fuzzy/PI two-stage controller is implemented as shown in Figure 6.44.

The fuzzy/PI two-stage controller consists of a 'Frequency Control' block and a 'PI Controller' block. Details of the two blocks are shown in Figures 6.8 and 6.10, respectively. A 'Saturation2' block in Simulink library is employed to limit the amplitude of the stator current.

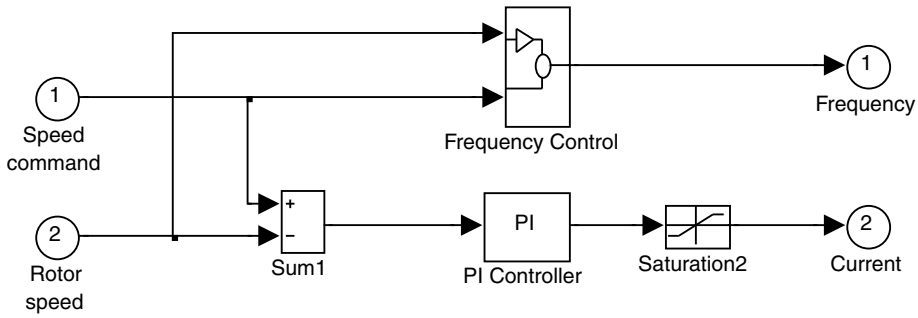


Figure 6.44 Fuzzy/PI two-stage controller.

Step 2 Choosing Parameters for the Simulink Blocks
The 'PI Controller' employs a 'PID Controller' in Simulink library with a derivative parameter of 0. Assuming the permissible magnitude of the stator current of the induction motor to be 3 A, the proportional and integral parameters in the 'PID Controller' block is designed as described in Section 6.7 and the following parameters are input into the block:

Proportional = 1
Integral = 0.03
Derivative = 0

The Simulink block 'Fuzzy Logic Controller' shown in Figure 6.8 contains the parameter 'ACCELE' which is a file name of FIS (Fuzzy Inference System). The file 'ACCELE' may be edited by the FIS editor using the command 'Fuzzy' in the MATLAB® window. The FIS edit operation steps have been described in Section 4.2.2.

The 'Fuzzy Logic Controller' block has two inputs and one output. They are the rotor speed command, rotor speed error, and reference slip frequency, respectively. The fuzzy membership functions are input and edited by 'Membership Function Editor' on FIS editor platform. The fuzzy membership function of the rotor speed command is shown in Figure 6.32, the fuzzy membership function of speed error is shown in Figure 6.33, and the fuzzy linguistic values of reference slip frequency are listed in Table 6.4. The fuzzy rules of the fuzzy controller are given in Table 6.5 and the fuzzy rulebase is input and edited by 'Rule editor' on the FIS editor platform.

Step 3 Implementing an Induction Motor Drive with Fuzzy/PI Two-Stage Controller
To study the performance of the induction motor drive with the fuzzy/PI two-stage controller, the latter model is connected to the induction motor model built earlier with a '3-phase current' block (which is described in Figure 3.2), three 'PI' blocks of stator-current control designed from Equations (6.26)–(6.40), a load which is described by Equation (6.12), a 'Speed Command' block which employs a 'Repeating Sequence' block in Simulink library, and a 3/2 transformation block which is described by Equation (3.5). These models are connected together to simulate the fuzzy/PI control system with a voltage-input model of the induction motor, as shown in Figure 6.31.

Step 4 Running the Simulink Model of Fuzzy/PI Control System
The model shown in Figure 6.31 is run with following parameters.

Simulation type: variable-step
Max step = 0.0001
Min step = auto
Initial step = 0.0001
Simulation time = 1.5 s

The rotor speed commands are as follows,

$$\omega_o{}^* = 180\,\text{rad/s} \quad 0\,\text{s} \le t < 0.5\,\text{s}$$
$$\omega_o{}^* = 20\,\text{rad/s} \quad 0.5\,\text{s} \le t \le 1.5\,\text{s}$$

The following parameters are input into the 'Speed Command' block.

Time values = [0 0.5 0.5 1.5];
Output values = [180 180 20 20].

The 'load' block is a 'Fcn' block in Simulink library, which contains the parameter 'u(1)*0.00 532' to simulate the load described by Equation (6.12) with coefficient $\mu = 0.00\ 532$ Nm/(rad/s), and the input u(1) is the rotor speed.

The simulation results of the fuzzy/PI control system are shown in Figures 6.34–6.36.

6.9 Summary

Based on the two-stage strategy and the heuristics deduced from the field-oriented principle, a hybrid fuzzy/PI controller is proposed. The controller has almost the same frequency and current characteristics as the field-oriented controller. During the acceleration/deceleration stage, the stator current magnitude is maintained at the maximum permissible value to give a large torque, and during the steady-state stage, the stator current magnitude is adjusted to control the rotor speed. Because the two features of field-oriented control are exploited, the performance of the two-stage controller is superior to a scalar controller (Garcia, Stephan and Watanabe, 1994). Besides, the hybrid fuzzy/PI controller has the advantages of simplicity and insensitivity to motor parameter changes, input current noise, noise in the measured speed, magnetic saturation, and load torque variation. Very encouraging results are obtained from a computer simulation using MATLAB®/Simulink. Due to the excellent speed response over the whole speed range, the method should find applications in practical industrial drive systems. The possible developments of the hybrid fuzzy/PI two-stage control are (1) to develop a design method of the fuzzy controller in the presence of disturbances and parameter variations, (2) optimizing the fuzzy rules and membership functions of the controller with self-learning methods such as neuro-fuzzy and genetic algorithm.

References

Bose, B.K. (1993) Power electronics and motion control-technology status and recent trends. *IEEE Transactions on Industry Applications*, **29**, 902–909.

Bose, B.K. (1997) Expert system, fuzzy logic, and neural networks in power electronics and drives, in *Power Electronics and Variable Frequency Drives: Technology and Applications* (ed. B.K. Bose) IEEE Press, New Jersey.

Garcia, G.O., Stephan, R.M., and Watanabe, E.H. (1994) Comparing the indirect field-oriented control with a scalar method. *IEEE Transactions on Industrial Electronics*, **41**(2), 201–207.

Henson, M.A. and Seborg, D.E. (1997) *Nonlinear Process Control*, Prentice-Hall, Upper Saddle River, NJ.

Kosko, B. (1997) *Fuzzy Engineering*, Prentice-Hall, Inc., New Jersey.

Krause, P.C., Wasynczuk, O., and Sudhoff, S.D. (1995) *Analysis of Electric Machinery*, IEEE Press, New Jersey.

The MathWorks, Inc. (1994) Simulink User's Guide, The MathWorks, Inc.

Shi, K.L., Chan, T.F., and Wong, Y.K. (May 1997) A novel two-stage speed controller for an induction motor. The 1997 IEEE International Electric Machines and Drives Conference, USA.

Shi, K.L., Chan, T.F., and Wong, Y.K. (1999) Modeling and simulation of a novel two-stage controller for an induction motor. *International Association of Science and Technology for Development (IASTED) Journal on Power and Energy Systems, USA*, **19**(3), 257–264.

Sousa, G.C.D. and Bose, B.K. (1994) A fuzzy set theory based control of a phase-controlled converter DC machine drive. *IEEE Transactions on Industry Applications*, **30**(1), 34–44.

Trzynadlowski, A.M. (1994) *The Field Orientation Principle in Control of Induction Motors*, Kluwer Academic Publishers, Boston.

Wade, S., Dunnigan, M.W., and Williams, B.W. (1997) Modeling and simulation of induction machine vector control with rotor resistance identification. *IEEE Transactions on Power Electronics*, **12**(3), 495–505.

7

Neural-Network-based Direct Self Control[1]

7.1 Introduction

Nonlinear dynamical control research using the neural network has been proposed for almost two decades (Narendra and Parthasarathy, 1990). Recently, it has been proposed that neural networks can be applied to parameter identification and state estimation of induction motor control systems (Simoes and Bose, 1995). However, complete ANN vector control of induction motors is seldom reported, one of the reasons being the complexity of the controller.

Direct self controller (DSC) is a dynamic, recurrent, and nonlinear signal processing method which theoretically can give an inverter-fed three-phase induction motor an excellent performance (Kazmierkowski and Kasprowicz, 1995). Because complicated calculations are involved, it is difficult to implement DSC using common integrated circuit (IC) circuit hardware. The DSC algorithms are usually studied by serial calculations on a digital signal processor (DSP) board. However, as a predictive control scheme, DSC has a steady-state control error produced by the time delay of the lengthy computations, which depends largely on the control algorithm and hardware performance. Typical DSP (TMS32010) execution time of the DSC algorithm is about 250 μs (Habetler *et al.*, 1992), hence the maximum switching frequency of the inverter has to be limited to 4 kHz. Consequently, DSC is usually suitable for motor drives with low switching frequencies. With simple architecture and the inherent parallel computation capability, a neural-network controller is superior to a DSP board in execution time and hardware structure. The execution times of neural devices are less than 0.5 μs (analog)

[1] (a) Portions reprinted by permission of K.L. Shi, T.F. Chan and Y.K. Wong, "Direct self control of induction motor using artificial neural network," 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1696–1701, October 11–14, 1998, San Diego, U.S.A. © 1998 IEEE
(b) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," IEEE Industry Applications Society (IEEE-IAS) 2000 Meeting, vol. 3, pp. 1380–1387, October 8–12, 2000, Rome, Italy. © 2000 IEEE.
(c) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

or 0.8 μs (digital) per neuron (Zaghloul, Meador and Newcomb, 1994). Two neural devices have been suggested (Simoes and Bose, 1995) for induction motor control. Micro Devices MD-1220, a digital VLSI device, takes 0.8 μs to process any synaptic input (or bias) when the clock rate is 20 MHz. Intel 80170NX (Electrically Trainable Analog Neural Network), on the other hand, is an analog device which takes only 3 μs to process through each layer. This chapter presents an ANN algorithm with 7 layers and 58 neurons to replace the DSP serial calculations of the classical DSC system. Because the 58 neurons take only 46.4 μs (using the digital MD-1220) or 21 μs (using the analog 80170NX), the control precision of DSC can be significantly improved using the neural-network algorithm.

7.2 Neural Networks

In general, a neural model is mathematically represented by a basis function (net function) and an activation function (neuron function). The selection of these functions often depends on the applications of the neural network. In other words, application-driven neural models are only loosely tied to the biological realities. Linear basis function $u(w_i,x)$ is a hyperplane-type function, where w_i stands for the weight matrix, x for the input vector, and θ_i for the bias or threshold. Mathematically (Kung, 1993),

$$u_i(w_i, x) = \sum_{j=1}^{n} w_{ij}x_j + \theta_i \tag{7.1}$$

where j is the dimension of input.

The net value as expressed by the basis function, $u_i(w_i,x)$, will be immediately transformed by an activation function of the neuron. Thus,

$$y_i = f(u_i) \tag{7.2}$$

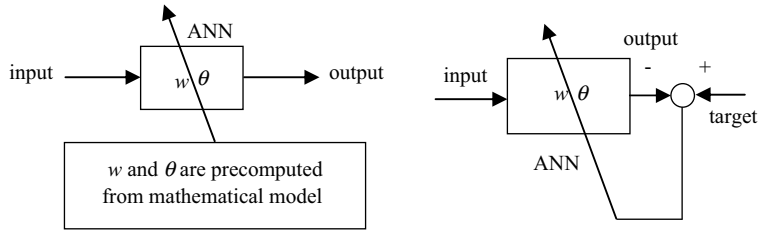
where y_i is net output and $f(\cdot)$ is the activation function. The activation functions used in this design include linear, square, log-sigmoid, tan-sigmoid, and hard limit functions (Appendix E).

The memory of a neural network lies in the weights and biases. The neural networks can be classified, in terms of how the weights and biases are obtained, into three categories (Kung, 1993; Fausett, 1994). They are fixed-weight, unsupervised and supervised networks. In this design, the fixed-weight networks and the supervised networks are used. The constructions of the two networks are shown in Figure 7.1. The training data consist of pairs of input and target produced by the DSC mathematical model.

The characteristic of the fixed-weight network is that the weights and biases are pre-computed and pre-stored from training data. The fixed-weight network, which is also called the direct design method, can be used to implement an exact mathematical model. In same cases, its implementation is easier than the supervised network.

In the supervised network, the weights and biases are adaptively trained by a learning mechanism, which has been the mainstream of neural model development. The back-propagation learning rule (Kung, 1993) is used to design the supervised networks for the DSC, details of which are presented as follows.

It is convenient to regard the threshold θ just as an extra weight, that is $\theta = w_{n+1}$ (Kung, 1993). The net value given by Equation (7.1) can be rewritten as:



(a) Fixed-weight network (b) Supervised network

Figure 7.1 Neural networks using different training algorithms. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

$$u_i(w_i, x) = \sum_{j=1}^{n+1} w_{ij} z_j = W_i Z \tag{7.3}$$

where $W_i = [w_{i1} w_{i2} \ldots w_{in}\ \theta_i]$ and $Z = [x_1 x_2 \ldots x_n\ 1]^{\mathrm{T}}$

The sum squared error E (cost function) for the set of M patterns of input is given by Equation (7.16) (Simoes and Bose, 1995)

$$E = \frac{1}{2} \sum_{m=1}^{M} E_m = \frac{1}{2} \sum_{m=1}^{M} \sum_{i=1}^{I} (t_i^m - y_i^m)^2 \tag{7.4}$$

where E_m is the squared output error of the output layer, I is the dimension of the output vector, y^m is the actual output vector, and t^m is the corresponding desired output vector. The weights are changed to reduce the cost function E to a minimum value by the gradient descent method.

The best initial weights and biases for back-propagation networks are created at random utilizing the minimum and maximum value of each input. The jth weight-update equation of the ith neuron is given as:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \left(\frac{\partial E_m}{\partial w_{ij}(t)} \right) \tag{7.5}$$

where η is the learning rate, $w_{ij}(t+1)$ is the new weight, and $w_{ij}(t)$ is the old weight.

The training strategies may be divided as mutual (whole) and individual (local) (Kung, 1993). In mutual training, the training of all the weights is influenced by all the input/output values. In individual training, the training of an individual subnet will not be influenced by the inputs and outputs of other subnets. Pure mutual training is almost impossible for DSC, due to three reasons. Firstly, the direct self controller is a dynamic (there are integrators), recurrent (there are hysteresis comparators), and nonlinear system. Secondly, the eight input variables (V_a, V_b, V_c, i_a, i_b, i_c, T^*, $|\lambda_\mu^s|^*$) constitute a huge training set. Thirdly, it may take substantially more iterations to reach a mutual agreement between all the nodes. For simpler and faster design, the individual training strategy is adopted.

7.3 Neural-Network Controller of DSC

A DSC scheme consists typically of 3/2 transformations of current and voltage, flux estimation, torque calculation, flux angle encoder, flux magnitude computation, hysteresis comparator, and optimum switching table. Figure 7.2 shows a DSC system in the MATLAB®/Simulink window, which consists of a DSC controller, an inverter, and an induction motor (Shi, Chan and Wong, 1998).

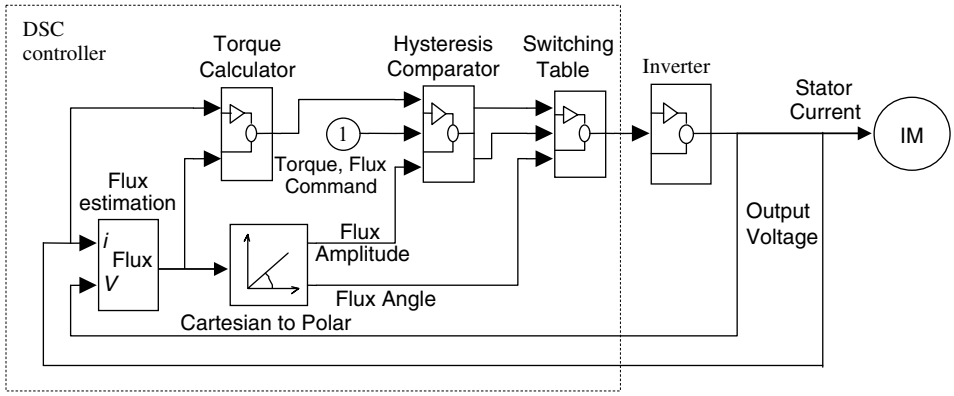


Figure 7.2 A construction of the direct self control system in MATLAB®/Simulink window. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

Based on DSC principle, the neural-network controller is divided into five sub-nets, which are individually trained: (1) flux estimation sub-net (fixed-weight) with dynamic neurons, (2) torque calculation sub-net (fixed-weight) with square neurons, (3) flux angle encoder and magnitude calculation sub-net (supervised) with log-sigmoid neurons and tan-sigmoid neurons, (4) hysteresis comparator sub-net (fixed-weight) with recurrent neurons, and (5) optimum switching table sub-net (fixed-weight or supervised) with hard limit neurons.

7.3.1 Flux Estimation Sub-Net

Based on Equation (2.6) and 3/2 transformation, the flux estimation is mathematically expressed as:

$$\frac{d\lambda^s_{d\mu}}{dt} = (V_a - R_s i_a) - \frac{1}{2}(V_b - R_s i_b) - \frac{1}{2}(V_c - R_s i_c) \tag{7.6}$$

$$\frac{d\lambda^s_{q\mu}}{dt} = \frac{\sqrt{3}}{2}(V_b - R_s i_b) - \frac{\sqrt{3}}{2}(V_c - R_s i_c) \tag{7.7}$$

where V_a, V_b, V_c are the phase voltages and i_a, i_b, i_c are phase currents, which come from the voltage and current sensors.

Neuron models can be divided in two basic types, namely static and dynamic. A dynamic neuron is one whose output is described by a differential equation (Delgado, Kambhampati and Warwick, 1995). Hence, the flux estimation sub-net should be constructed using two dynamic neurons which consist of linear neurons and integrators, as shown in Figure 7.3.

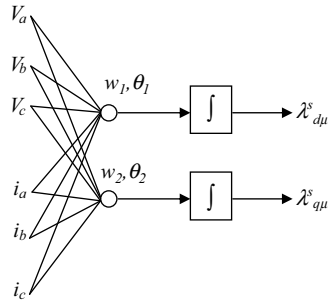


Figure 7.3 A dynamic net of the flux estimation. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

A supervised method, viz. the back-propagation learning rule, is used to train the linear neurons until they can approximate Equations (7.6) and (7.7).

Figure 7.3 shows the flux estimation network. Using a random generator function of MATLAB® 'randnc', 10 random inputs of the vector [V_a, V_b, V_c, i_a, i_b, i_c] are produced. The target outputs can be obtained from Equations (7.6) and (7.7). Since the network is linear, convergence can be obtained in relatively few training epochs. For the induction motor being studied, the weights and biases have been obtained as follows:

$$w = \begin{bmatrix} 0.5725 & -0.2910 & 0.7790 & -0.3740 & -0.7496 & -0.0172 \\ -0.3899 & 1.3073 & -0.2469 & 0.4546 & -0.7029 & 0.4575 \end{bmatrix}$$

$$\theta = \begin{bmatrix} -0.2029 \\ -0.2836 \end{bmatrix}.$$

7.3.2 *Torque Calculation Sub-Net*

Based on Equation (2.6), the torque equation for a DSC system is given by:

$$T = \frac{P}{2}\frac{2}{3}(\lambda^s_{d\mu} i^s_{qs} - \lambda^s_{q\mu} i^s_{ds}) \tag{7.8}$$

where P is the number of motor poles.

Since there are four inputs, $\lambda^s_{d\mu}$, $\lambda^s_{q\mu}$, i^s_{ds}, and i^s_{qs}, the data of all training patterns will be huge if high precision is required. To avoid training difficulties, the fixed-weight method is adopted. Equation (7.8) may be rewritten as a sum of square functions:

$$T = \frac{P}{6}\left[(\lambda^s_{d\mu} + i^s_{qs})^2 - (\lambda^s_{q\mu} + i^s_{ds})^2 - \lambda^{s2}_{d\mu} + \lambda^{s2}_{q\mu} + i^{s2}_{ds} - i^{s2}_{qs}\right] \tag{7.9}$$

where

$$\begin{bmatrix} i^s_{ds} \\ i^s_{qs} \end{bmatrix} = \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix}$$

A two-layer, fixed-weight neural network is used to implement Equation (7.9) directly as shown in Figure 7.4. The first layer is a square activation function with the weight and bias w_1 and θ_1, while the second layer is a linear active function with the weight and bias w_2 and θ_2.

$$w_1 = \begin{bmatrix} 1 & 0 & 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 0 & 1 & 1 & -1/2 & -1/2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1/2 & -1/2 \\ 0 & 0 & 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \quad \theta_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$w_2 = [1\ -1\ -1\ 1\ 1\ -1] \quad \theta_2 = [0]\,.$$

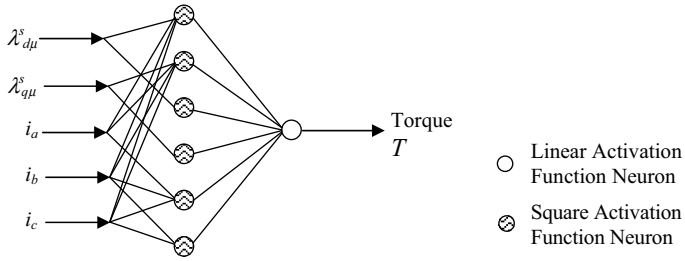


Figure 7.4 Neural network for torque calculation. (Reproduced by permission of K.L. Shi, T.F. Chan, Y. K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

7.3.3 Flux Angle Encoder and Flux Magnitude Calculation Sub-Net

The flux angle α and flux magnitude f can be calculated from the flux space vectors $\lambda^s_{d\mu}$ and $\lambda^s_{q\mu}$. Then the flux angle is encoded as $B_1B_2B_3$.

$$f = \sqrt{\lambda^{s2}_{d\mu} + \lambda^{s2}_{q\mu}} \tag{7.10}$$

$$\alpha = \tan^{-1}(\lambda^s_{d\mu}/\lambda^s_{q\mu}) \tag{7.11}$$

$$B_1B_2B_3 = \text{encoder}(\alpha) \tag{7.12}$$

In order to obtain accurate results and to simplify the design, Equations (7.10) and (7.11) are rewritten as:

$$\upsilon = \lambda^{s2}_{d\mu} + \lambda^{s2}_{q\mu} \tag{7.13}$$

$$f = \sqrt{\upsilon \times \begin{bmatrix} 1 \\ 1 \end{bmatrix}} \tag{7.14}$$

$$\xi = 1/\lambda^s_{q\mu} \tag{7.15}$$

$$\zeta = \lambda^s_{d\mu} \times \xi = [(\lambda^s_{d\mu} + \xi)^2 - \lambda^s_{d\mu}{}^2 - \xi^2]/2 \tag{7.16}$$

$$B_1B_2B_3 = \text{encoder}(\zeta). \tag{7.17}$$

The network of flux angle encoder and flux magnitude calculation consists of five nets as shown in Figure 7.5. Net1 with two square neurons implements Equation (7.13). Net2 with four tansig neurons implements Equation (7.14). Net3 with four logsig neurons, net4 with three square neurons, and net5 with ten hard limit neurons implement Equations (7.15)–(7.17), respectively. Net1, net4, and net5 are designed using the fixed weight method. Net2 and net3 are designed using the supervised method.

The output layers of the net1, net2, net3, and net4 will be merged with the input layers of their next sub-nets according to the rule shown in Figure 7.14. Hence, weights and biases of the output layers of these nets will not be listed hereinafter.

Design of net1 is similar to that used for the torque calculation sub-net described in Section 7.2. Weight w and bias θ of net1 with two square neurons can be directly designed according to Equation (7.13) as follows:

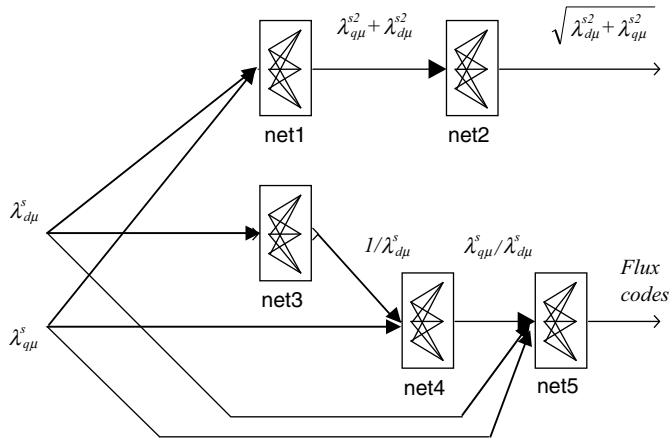


Figure 7.5 Individual training scheme for flux angle and magnitude calculations. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

$$w(\text{net } 1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \theta(\text{net } 1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The back-propagation learning rule is used to train net 2. 500 input/output pairs for training net 2 are produced by the square root function. After 50 000 training epochs, the sum-squared error E is less than 0.01. Weight w and bias θ of net2 with tansig neurons is as follows.

$$w(\text{net } 2) = \begin{bmatrix} -0.1781 & -0.1781 \\ 2.5008 & 2.5008 \\ -2.9325 & -2.9325 \\ 0.5233 & 0.5233 \end{bmatrix} \quad \theta(\text{net } 2) = \begin{bmatrix} 1.5851 \\ 6.0920 \\ -6.2367 \\ 0.3963 \end{bmatrix}$$

Figure 7.6 shows the implementation of the flux magnitude calculation using net 1 and net2.

The back-propagation learning rule is used to train net 3 until they can approximate the reciprocal function. 1000 input/output pairs are produced by the reciprocal function to train net3. After 100 000 training epochs, the sum-squared error E is less than 0.02.

$$w(\text{net } 3) = \begin{bmatrix} -31.6560 \\ 2.8829 \\ 14.8182 \\ 47.8563 \end{bmatrix} \quad \theta(\text{net } 3) = \begin{bmatrix} -0.7218 \\ -0.0022 \\ 0.0207 \\ 0.7360 \end{bmatrix}$$

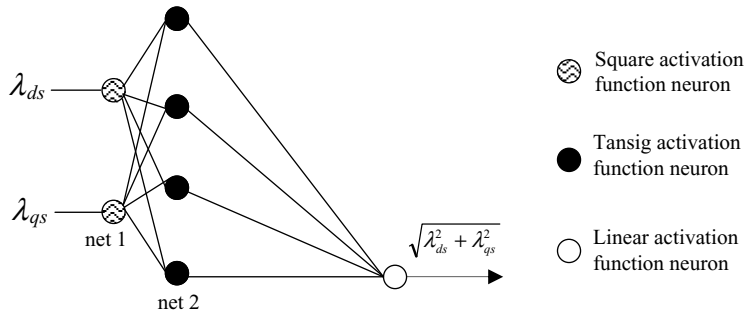


Figure 7.6 Implementation of the flux magnitude calculation. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

With three square neurons, net 4 implements Equation (6.13) using the same technique described in Section 7.2.

$$w(\text{net } 4) = \begin{bmatrix} 1/2 & 1/2 \\ -1/2 & 0 \\ 0 & -1/2 \end{bmatrix} \quad \theta(\text{net } 4) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Net5 implements the flux angle encoding directly from $\lambda_{d\mu}^s/\lambda_{q\mu}^s$ (output of net4), $\lambda_{d\mu}^s$, and $\lambda_{q\mu}^s$. To improve the algorithm, the trigonometric function computations of flux angle, which are necessary in previous DSC schemes, are replaced by logic operations. With reference to Figure 7.7, the flux angle code ($B_1\,B_2\,B_3$) can be directly derived from the following equations:

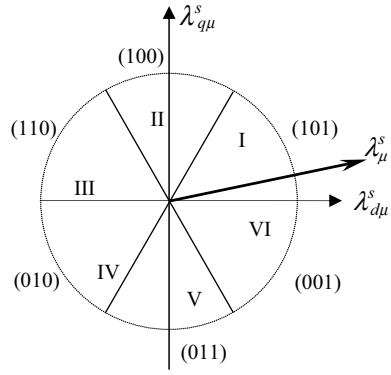


Figure 7.7 Space flux encoder ($B_1\,B_2\,B_3$). (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

$$B_1 = \begin{cases} 1 & \text{if } \lambda^s_{q\mu} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.18}$$

$$B_2 = \begin{cases} 1 & \text{if } (\lambda^s_{q\mu}/\lambda^s_{d\mu} \geq -\tan\ \pi/3 \text{ and } \lambda^s_{d\mu} < 0) \\ & \text{or if } (\lambda^s_{q\mu}/\lambda^s_{d\mu} < -\tan\ \pi/3 \text{ and } \lambda^s_{q\mu} < 0) \\ 0 & \text{otherwise} \end{cases} \tag{7.19}$$

$$B_3 = \begin{cases} 1 & \text{if } (\lambda^s_{q\mu}/\lambda^s_{d\mu} < \tan\ \pi/3 \text{ and } \lambda^s_{d\mu} \geq 0) \\ & \text{or if } (\lambda^s_{q\mu}/\lambda^s_{d\mu} \geq \tan\ \pi/3 \text{ and } \lambda^s_{q\mu} < 0) \\ 0 & \text{otherwise} \end{cases} \tag{7.20}$$

If we define:

$$M = \lambda^s_{q\mu}/\lambda^s_{d\mu} \geq \tan \pi/3 \quad \bar{M} = \lambda^s_{q\mu}/\lambda^s_{d\mu} < \tan \pi/3 \tag{7.21}$$

$$N = \lambda^s_{q\mu}/\lambda^s_{d\mu} \geq -\tan \pi/3 \quad \bar{N} = \lambda^s_{q\mu}/\lambda^s_{d\mu} < -\tan \pi/3 \tag{7.22}$$

$$K = \lambda^s_{d\mu} \geq 0 \quad \bar{K} = \lambda^s_{d\mu} < 0 \tag{7.23}$$

$$L = \lambda^s_{q\mu} \geq 0 \quad \bar{L} = \lambda^s_{q\mu} < 0 \tag{7.24}$$

then the encoder Equations (7.18)–(7.20) can be written as: (positive logic expression)

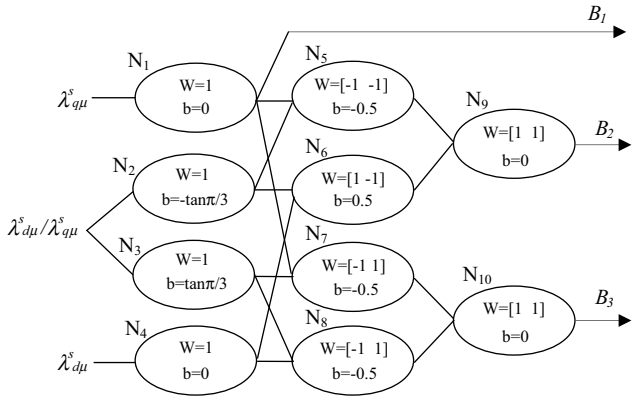
$$B_1 = L \tag{7.25}$$

$$B_2 = (N \cap \bar{K}) \cup (\bar{N} \cap \bar{L}) \tag{7.26}$$

$$B_3 = (\bar{M} \cap K) \cup (M \cap \bar{L}). \tag{7.27}$$

Equations (7.21)–(7.27) show that trigonometric function calculations are not needed for the flux angle encoding, hence the complexity of implementation is decreased. The flux angle encoder can be accomplished by the hard limit neurons as shown in Figure 7.8 (Fausett, 1994).

The five nets are linked together to form the network of flux angle encoder and flux magnitude computation, which has 24 different neurons as shown in Figure 7.9.



Equation (7.24) is implemented by the neuron N_1.

Equation (7.22) is implemented by the neuron N_2.

Equation (7.21) is implemented by the neuron N_3.

Equation (7.23) is implemented by the neuron N_4.

Equation (7.25) is implemented by the output of neuron N_1.

Equation (7.26) is implemented by the neurons N_5, N_6, and N_9.

Equation (7.27) is implemented by the neurons N_7, N_8, and N_{10}.

Figure 7.8 Flux angle encoder (net5) with hard limit neurons. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

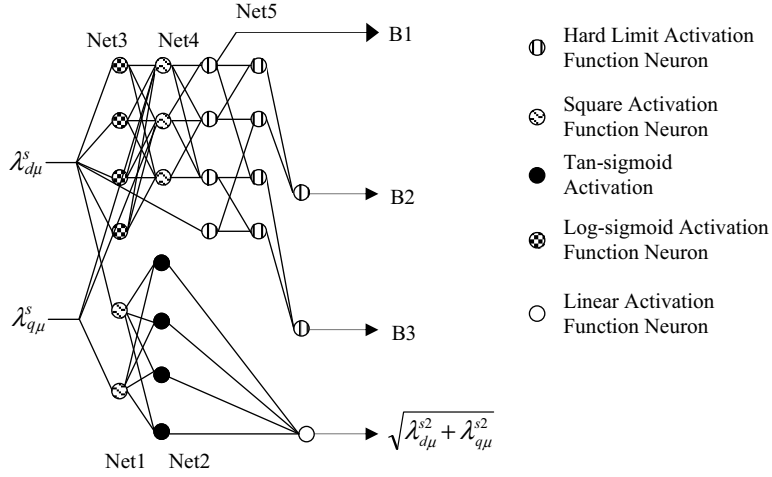


Figure 7.9 Network of flux angle encoder and flux magnitude computation. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

7.3.4 *Hysteresis Comparator Sub-Net*

Using the hysteresis comparator as shown in Figure 7.10, the flux error between stator flux $|\lambda_\mu^s|$ and its command $|\lambda_\mu^s|^*$ can be limited within $\pm\Delta|\lambda_\mu^s|$, and the flux error code B_6 produced by the hysteresis comparator will be used to select the space voltage vector (Takahashi and Noguchi, 1986).

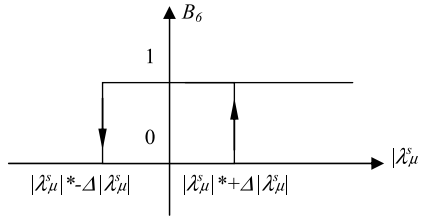


Figure 7.10 Flux magnitude hysteresis comparator. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

The flux error code B_6 can be expressed as:

$$B_6 = \begin{cases} 0 & (\text{if } |\lambda_\mu^s| < |\lambda_\mu^s|^* + \Delta|\lambda_\mu^s| \text{ and } B_6 = 0) \\ & \text{or } (\text{if } |\lambda_\mu^s| < |\lambda_\mu^s|^* - \Delta|\lambda_\mu^s| \text{ and } B_6 = 1) \\ 1 & (\text{if } |\lambda_\mu^s| \geq |\lambda_\mu^s|^* + \Delta|\lambda_\mu^s| \text{ and } B_6 = 0) \\ & \text{or } (\text{if } |\lambda_\mu^s| \geq |\lambda_\mu^s|^* - \Delta|\lambda_\mu^s| \text{ and } B_6 = 1) \end{cases} . \tag{7.28}$$

Equation (7.28) represents a recurrent calculation: *To obtain* B_6, *we have to do a calculation using* B_6, *and in order to do the calculation, we have to obtain* B_6. In order to derive the weight and bias of network, Equation (7.28) is rewritten as:

$$B_6 = \begin{cases} 0 & \text{if } (w_1 B_6 + |\lambda_\mu^s| - |\lambda_\mu^s{}^*| - \Delta|\lambda_\mu^s| < 0) \\ 1 & \text{if } (w_1 B_6 + |\lambda_\mu^s| - |\lambda_\mu^s{}^*| - \Delta|\lambda_\mu^s| \geq 0) \end{cases} \tag{7.29}$$

where $w_1 = 2\Delta|\lambda_\mu^s|$.

The output B_6 is connected as an input, which forms a recurrent network (The MathWorks, Inc., 1994). Using the basis function given by Equation (7.1) for a neural network, the weight and bias can be pre-computed. The input of network $x = [B_6, |\lambda|, |\lambda^*|]$. Its output is B_6, its weight $w = [2\Delta|\lambda| \ \ 1 \ \ -1]$, and its bias $\theta = [-\Delta|\lambda|]$. The flux hysteresis comparator is implemented by a recurrent network with hard limit function as shown in Figure 7.11.

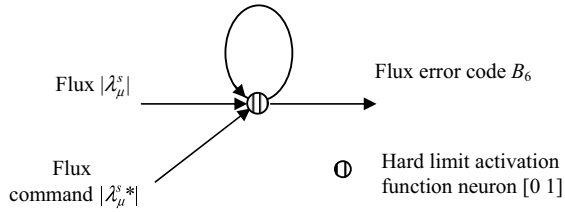


Figure 7.11 Flux hysteresis comparator of neural network. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

The difference between the motor torque T and the torque command T^* is compared with ΔT and the error flag is used to produce a torque error code for selecting the voltage space vector (Takahashi and Noguchi, 1986), that is.

$$T^* - \Delta T \leq T \leq T^* \tag{7.30}$$

(when λ_μ^s rotates in the clockwise direction)

$$T^* \leq T \leq T^* + \Delta T \tag{7.31}$$

(when λ_μ^s rotates in the counterclockwise direction)

The torque hysteresis comparator expressed by Equations (7.30) and (7.31) can be designed in a similar manner as the flux hysteresis comparator, which is shown in Figure 7.12. The torque error code consists of two bits B_4 and B_5.

The weight of u_1 is $[\Delta T\, 1\, -1]$ and the bias of u_1 is [0], while the weight of u_2 is $[\Delta T\, 1\, -1]$ and the bias of u_2 is $[\Delta T]$.

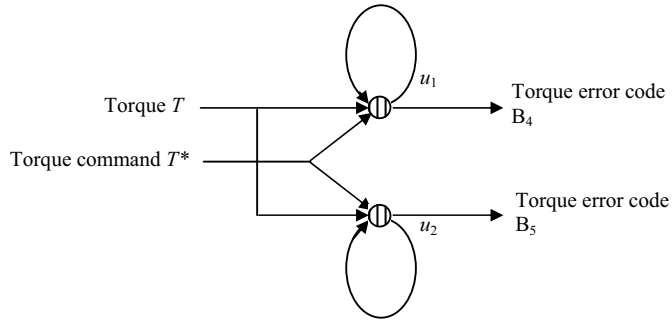


Figure 7.12 Torque hysteresis comparator of neural network. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

7.3.5 *Optimum Switching Table Sub-Net*

The DSC optimum switching table is expressed in Table 7.1. The flux angle code $B_1B_2B_3$, the torque error code B_4, B_5, and the flux magnitude error code B_6 determine the output voltage codes S_a, S_b, S_c. The output voltage codes of the optimum switch table represent the on/off status of the inverter switches (Takahashi and Noguchi, 1986).A two-layer network with a total of 26 hard limit neurons is employed to implement the optimum switching table. The first layer has 23 neurons and the second layer has 3 neurons. Utilizing the 36 pairs of input and output patterns shown in Table 7.1, the network is trained by a supervised method with perceptron training rule (The MathWorks, Inc., 1994). After 321 training epochs, the sum squared error E arrives at zero.

Table 7.1 DSC optimum switching table.

(S_a,S_b,S_c)	$B_1B_2B_3 =$ 001	$B_1B_2B_3 =$ 010	$B_1B_2B_3 =$ 011	$B_1B_2B_3 =$ 100	$B_1B_2B_3 =$ 101	$B_1B_2B_3 =$ 110
$B_4B_5B_6 = 010$	(0,1,1)	(1,1,0)	(0,1,0)	(1,0,1)	(0,0,1)	(1,0,0)
$B_4B_5B_6 = 000$	(1,1,1)	(1,1,1)	(0,0,0)	(1,1,1)	(0,0,0)	(0,0,0)
$B_4B_5B_6 = 100$	(1,0,1)	(0,1,1)	(0,0,1)	(1,1,0)	(1,0,0)	(0,1,0)
$B_4B_5B_6 = 011$	(0,1,0)	(1,0,0)	(1,1,0)	(0,0,1)	(0,1,1)	(1,0,1)
$B_4B_5B_6 = 001$	(0,0,0)	(0,0,0)	(1,1,1)	(0,0,0)	(1,1,1)	(1,1,1)
$B_4B_5B_6 = 101$	(1,0,0)	(0,0,1)	(1,0,1)	(0,1,0)	(1,1,0)	(0,1,1)

(Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

An optimum switching table implemented by a hard limit neural network is shown in Figure 7.13.

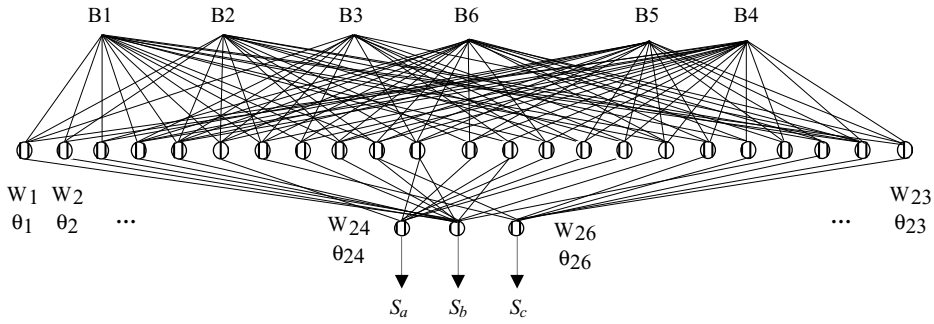


Figure 7.13 Optimum switching table implemented by neural network. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

Weights and biases of the trained network of the optimum switching table are as follows:

$$w_1 = \begin{bmatrix}
-0.3550 & 0.0369 & 0.2029 & 0.0241 & 0.4239 & -0.2690 \\
-0.2885 & -0.2401 & 0.6203 & 0.4311 & -0.1012 & 0.0932 \\
-0.7253 & 0.8712 & 0.8907 & 0.0085 & -0.0325 & -0.9459 \\
0.9269 & -0.5187 & 0.5237 & 0.1668 & -0.8606 & 0.4709 \\
0.6865 & 0.1340 & -0.3676 & 0.4559 & 0.8899 & -0.4498 \\
0.1286 & 0.0511 & -0.5529 & -0.0767 & 0.9817 & -0.9635 \\
0.2845 & -0.1546 & 0.8728 & -0.7310 & -0.2867 & -0.9462 \\
-0.3333 & 0.8429 & -0.6309 & 0.8661 & -0.5424 & -0.9813 \\
-0.2998 & 0.8419 & -0.0831 & -0.9935 & 0.3540 & 0.0070 \\
-0.4301 & 0.8902 & -0.4835 & 0.5583 & -0.2792 & -0.6442 \\
-0.1941 & -0.4154 & 0.1969 & -0.1388 & 0.5978 & -0.0162 \\
-0.1585 & -0.0047 & 0.3297 & -0.9324 & 0.7765 & 0.9993 \\
0.9794 & -0.6219 & -0.7241 & -0.4927 & 0.2107 & 0.5643 \\
0.3663 & -0.5114 & -0.7814 & -0.3536 & 0.4784 & 0.2400 \\
0.4391 & -0.3123 & -0.4460 & -0.4094 & 0.5830 & -0.4980 \\
0.2659 & 0.4385 & -0.4762 & 0.9414 & -0.9226 & -0.7044 \\
-0.2422 & 0.8008 & -0.6726 & 0.9360 & -0.0432 & 0.4455 \\
0.6741 & -0.4552 & -0.9438 & -0.0252 & -0.4626 & -0.1331 \\
0.2675 & -0.0007 & -0.2443 & 0.4314 & -0.7834 & -0.7321 \\
0.2400 & -0.9482 & -0.4688 & -0.5715 & -0.1893 & 0.4792 \\
-0.4291 & -0.2412 & 0.0102 & 0.1831 & 0.9659 & -0.7769 \\
-0.7092 & -0.8132 & -0.7978 & 0.1633 & 0.1369 & 0.6458 \\
0.7693 & 0.9648 & -0.7402 & -0.6547 & 0.9621 & -0.2571
\end{bmatrix}$$

$$\theta_1 = \begin{bmatrix} -0.4346 \\ 0.2551 \\ 0.6687 \\ 0.4239 \\ 0.5796 \\ 0.3578 \\ -0.3410 \\ -0.2098 \\ 0.0808 \\ -0.8631 \\ 0.9978 \\ -0.2047 \\ 0.4015 \\ 0.9724 \\ -0.5862 \\ 0.5770 \\ -0.9197 \\ -0.5717 \\ 0.8491 \\ -0.5232 \\ 0.2756 \\ 0.9595 \\ 0.5518 \end{bmatrix} \quad w_2^T = \begin{bmatrix} -41.0738 & 74.4303 & -44.9270 \\ -37.7507 & 46.8430 & -48.6175 \\ -6.3355 & -97.5883 & -79.2760 \\ -41.5267 & -5.9237 & 62.7760 \\ 14.9610 & -53.1857 & -36.3249 \\ 29.8704 & -9.3693 & 40.7466 \\ -45.2122 & -8.3780 & 77.5973 \\ -53.3863 & -7.1366 & 11.0640 \\ -22.1699 & -0.0711 & 7.3889 \\ -25.7650 & 73.9106 & 58.8391 \\ 57.7658 & -7.8811 & 47.1295 \\ 32.7479 & 17.4653 & -29.2237 \\ -40.8303 & -31.5257 & -76.5592 \\ -27.3672 & 75.5551 & -44.8241 \\ 24.7633 & -69.1695 & 4.9325 \\ 13.0352 & 40.2875 & -122.2603 \\ -28.9413 & -49.3222 & 26.6222 \\ 54.7596 & 33.2641 & -48.0328 \\ 39.4737 & -23.4507 & 58.4754 \\ -76.8339 & -136.3204 & -21.3997 \\ 15.8100 & 25.8781 & -10.4552 \\ 21.6393 & 6.4111 & 85.3883 \\ -38.4350 & 63.6614 & -2.7575 \end{bmatrix}$$

$$\theta_2 = \begin{bmatrix} 58.6962 \\ -7.9228 \\ 48.4529 \end{bmatrix}$$

7.3.6 *Linking of Neural Networks*

When the sub-nets are linked to each other, some neurons of the output layer may be merged with the input neurons of the next sub-net. For example, if an output neuron of a sub-net has a linear activation function, it may be merged with the input neuron of the next sub-net. As shown in Figure 7.14, the activation function of neuron *A* is linear, and its output is:

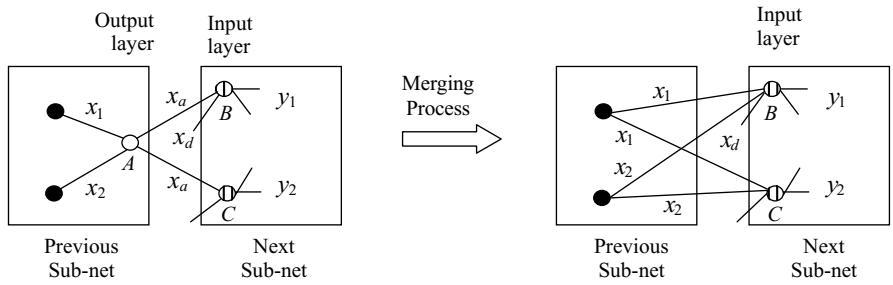


Figure 7.14 Merging of neurons with linear activation functions. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

$$x_a = w_1 x_1 + w_2 x_2 + \theta_1. \tag{7.32}$$

Let the basis function of neuron B be

$$u_1 = w_3 x_a + w_4 x_d + \theta_2. \tag{7.33}$$

Substituting (7.32) into (7.33),

$$u_1 = w_3 w_1 x_1 + w_3 w_2 x_2 + w_4 x_d + w_3 \theta_1 + \theta_2. \tag{7.34}$$

If the new weight and bias of neuron *B* are denoted by w' and θ'_2 respectively, then

$$w' = [w_3 w_1 \; w_3 w_2 \; w_4] \tag{7.35}$$

$$\theta'_2 = [w_3 \theta_1 + \theta_2]. \tag{7.36}$$

In this way, neuron *A* is merged with neuron *B* as well as the neuron *C*.

If an output neuron of a sub-net has a hard limit function, it can also be merged into next sub-net's input neuron that has also a hard limit function. Employing this strategy, the number of layers and neurons of the linked network can be decreased.

With the merging of input and output neurons, the five sub-nets (flux estimation, torque calculation, flux angle encoder and flux magnitude calculation, hysteresis comparator, and optimum switching table) are assembled into the DSC neural network as shown in Figure 7.15.

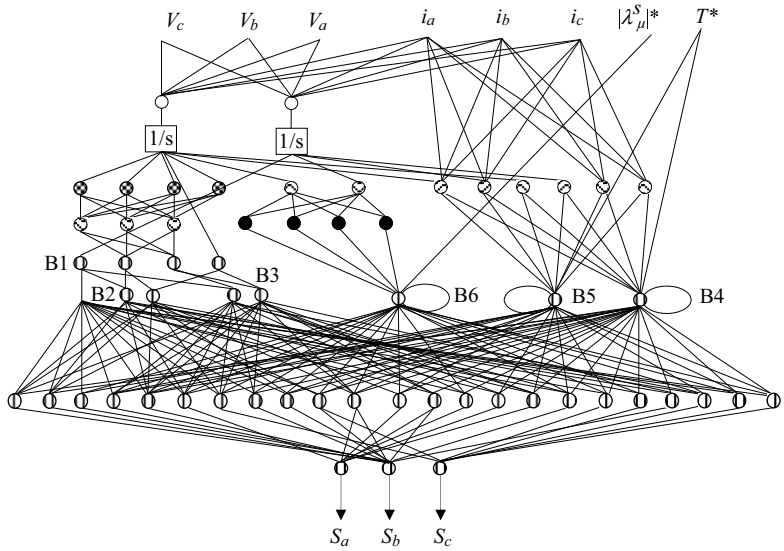


Figure 7.15 Neural-network implementation of DSC. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

The complete neural network consists of 7 layers and 58 neurons. It may be implemented using special neural devices. If the device 80170NX (Intel analog IC chip) is used, the processing time for the seven layers will be 21 μs. If the device MD-1220 (digital IC chip) is used, the processing time for 58 neurons will be 46.4 μs (at a clock rate of 20 MHz). Parallelism of neural device computation renders it extremely fast compared with the DSP serial computation.

7.4 Simulation of Neural-Network-based DSC

A MATLAB®/Simulink program with Neural Network Toolbox is used to simulate the neural-network DSC, which is shown in Figure 7.16.

The voltage-input model of an induction motor presented in Chapter 3 is used for the simulation studies. The parameters of the 7.5 kW induction motor are listed in Appendix B.

The stator flux command and torque command for ANN-DSC and classical DSC simulation are as follows:

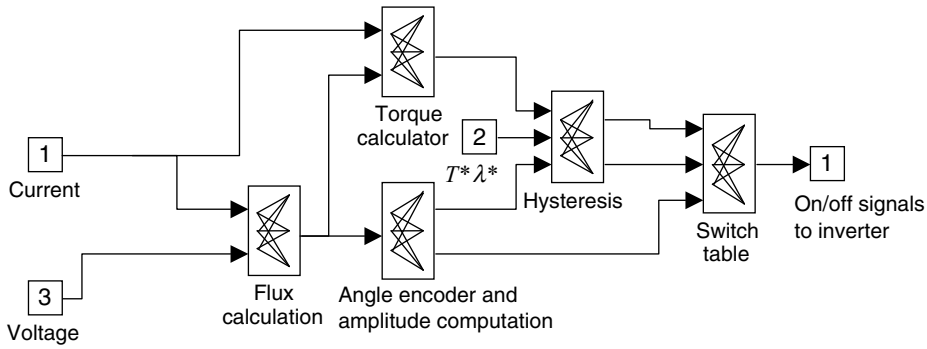


Figure 7.16 Neural-network-based DSC in Simulink. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

Stator flux commands:

$$|\lambda_{\mu}^{s}|^{*} = 0.86\ \text{Wb} \quad 0\ \text{s} < t \leq 4\ \text{s}$$

Torque commands:

$$T^{*} = 100\ \text{N m} \quad 0\ \text{s} < t \leq 0.8\ \text{s}$$

$$T^{*} = 20\ \text{N m} \quad 0.8\ \text{s} < t \leq 2\ \text{s}$$

$$T^{*} = -100\ \text{N m} \quad 2\ \text{s} < t \leq 2.3\ \text{s}$$

$$T^{*} = 20\ \text{N m} \quad 2.3\ \text{s} < t \leq 4\ \text{s}$$

Figures 7.17, 7.18 and 7.21 show the torque response, speed response, and flux response of the classical DSC system with 100 µs controller delay (typical of a DSP-based controller), while Figures 7.19, 7.20 and 7.22 show the torque response, speed response, and flux response

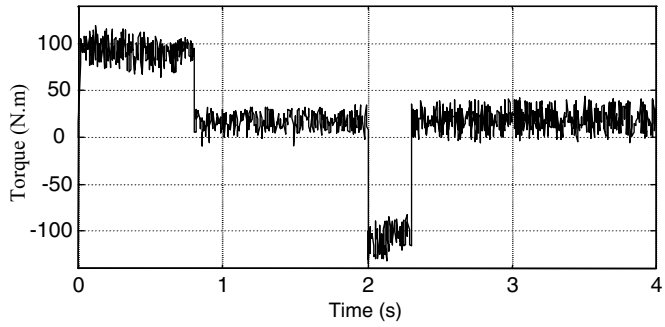


Figure 7.17 Torque response of DSC with 100 µs delay. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

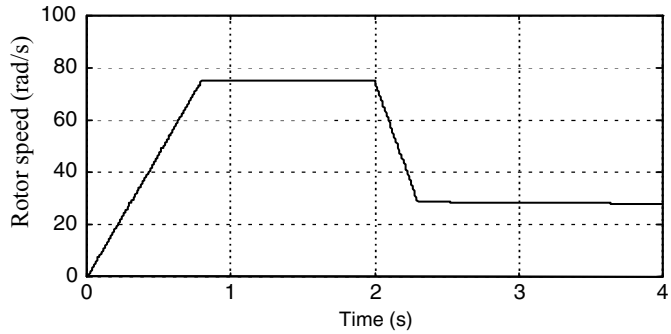


Figure 7.18 Speed response of DSC with 100 μs delay. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

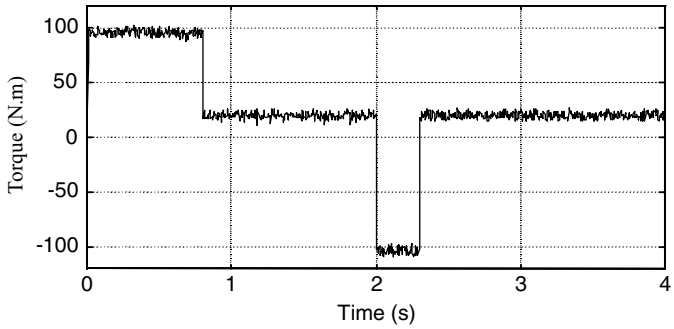


Figure 7.19 Torque response of ANN-DSC with 25 μs delay. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

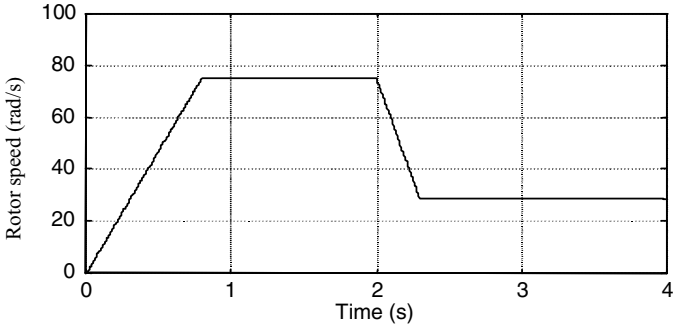


Figure 7.20 Speed response of ANN-DSC with 25 μs delay. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

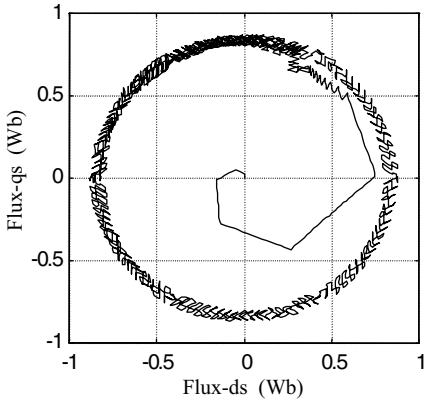


Figure 7.21 Flux response of DSC with 100 µs delay (time = 0–0.2 s). (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

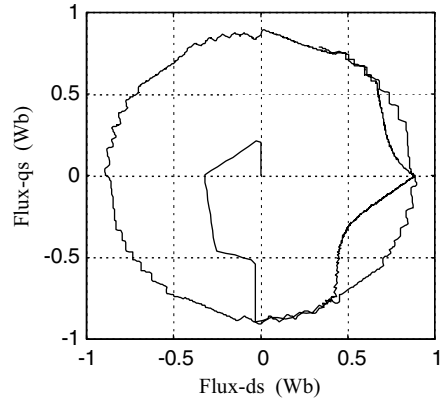


Figure 7.22 Flux response of ANN-DSC with 25 µs delay (time = 0–0.2 s). (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Direct self control of induction motor based on neural network," *IEEE Transactions on Industry Applications*, **37**(5), 2001: 1290–1298. © 2001 IEEE.)

of the neural-network control system with 25 µs controller delay. The results demonstrate that DSP-based DSC produces large torque and flux errors, whereas the neural-network controller eliminates almost all these errors. In the simulation studies, it is evident that (1) torque and flux errors increase with increasing controller delay time, (2) the large torque and flux errors decrease the robustness of the drive system against current noise and load changes. It can be concluded that neural-network-based DSC is a more effective algorithm for the control of an inverter-fed induction motor.

7.5 MATLAB®/Simulink Programming Examples

In this section, two examples are given to illustrate programming of a direct self controller (DSC) (Shi *et al.*, 2003) and a neural-network-based optimum switching table in MATLAB®/Simulink.

7.5.1 Programming Example 1: Direct Self Controller

The direct self control system shown in Figure 7.2 consists of the following function blocks: (1) flux estimation, (2) torque calculation, (3) flux angle encoder and magnitude calculation, (4) hysteresis comparator, and (5) optimum switching table. The realization of the neural-network-based controller in MATLAB®/Simulink involves the following steps.

Step 1 Building the Flux Estimation Model

Based on Equations (7.6) and (7.7), a flux estimation model is built as shown in Figure 7.23, where the gain block represents the stator resistance R_s of the motor (0.294 Ω).

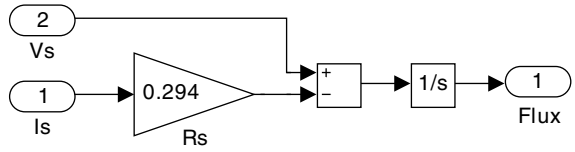


Figure 7.23 Simulink model of flux estimation.

In the Simulink model of flux estimation, the inputs are the *dq*-axis stator voltage vector and *dq*-axis stator current vector; while the output is the *dq*-axis flux vector.

Step 2 Building the Torque Calculation Model
Based on Equation (7.8), a torque calculation model is built for the induction motor (in which the number of poles $P = 6$) as shown in Figure 7.24.

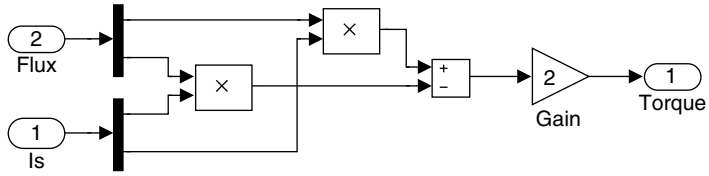


Figure 7.24 Simulink model of torque calculation.

In the torque calculation model, the input *dq*-axis flux vector is the output of the flux estimation model and the other input is the *dq*-axis stator current vector, while the output is the torque.

Step 3 Building the Flux Angle Encoder and Magnitude Calculation Model
With the *dq*-axis flux vector obtained in **Step 1** and based on Equations (7.10) and (7.11), the flux angle and magnitude is computed by employing a 'Cartesian to Polar' block in Simulink library, as shown in Figure 7.25.

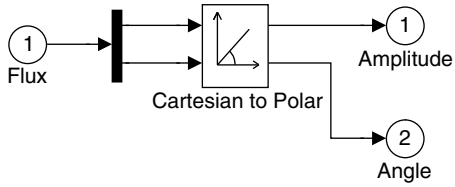


Figure 7.25 Flux angle and magnitude calculation.

Based on Equations (7.18), (7.19), and (7.20), a Simulink model of flux angle encoder is implemented as shown in Figure 7.26.

In the flux angle encoder model, the input is the flux angle and the output is the flux angle codes (B_1 B_2 B_3) of the DSC optimum switching table in Table 7.1.

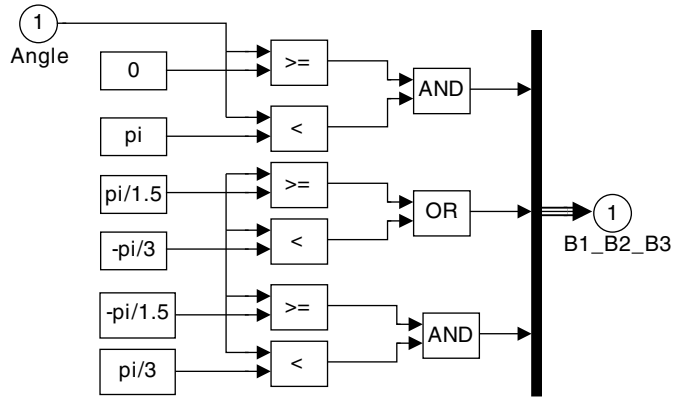


Figure 7.26 Simulink model of flux angle encoder.

Step 4 Building the Hysteresis Comparator Model

Based on Equations (7.28), (7.30), and (7.31), the hysteresis comparator may be built by 'Relay' blocks in Simulink library as shown in Figure 7.27.

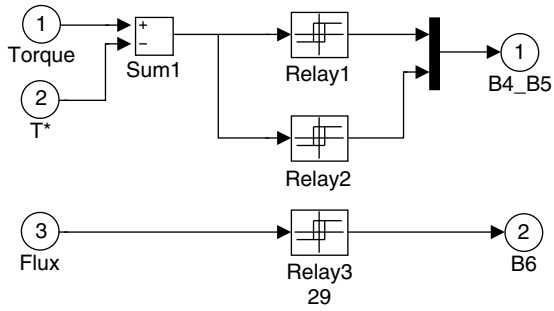


Figure 7.27 Simulink model of hysteresis comparator.

The outputs of the hysteresis comparator model are the codes ($B_4\,B_5\,B_6$) of the DSC optimum switching table in Table 7.1. When the flux command $|\lambda_\mu^s| = 0.86$ Wb, $\Delta|\lambda_\mu^s| = 0.2$ Wb, control error of torque $\Delta|T| = 5$ N m, the parameters of the hysteresis comparator model are as shown in Table 7.2.

Table 7.2 Parameters of Simulink model of the hysteresis comparator.

Block	Switch on point	Switch off point	Output when on	Output when off
Relay 1 Equation (7.30)	0	−5	0	1
Relay 2 Equation (7.31)	5	0	1	0
Relay 3 Equation (7.28)	0.88	0.84	1	0

Step 5 Building the Optimum DSC Switching Table

The DSC optimum switching table given in Table 7.1 is implemented by a 'Combinatorial Logic' block in Simulink library, as shown in Figure 7.28.

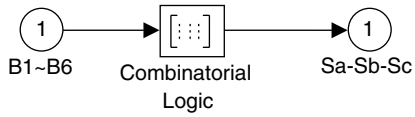


Figure 7.28 Simulink model of DSC optimum switching table.

The inputs of the 'Combinatorial Logic' block are the codes (B_1 B_2 B_3 B_4 B_5 B_6) and the outputs are the output voltage codes (S_a, S_b, S_c). The 36 states of the output voltage codes (S_a, S_b, S_c) listed in Table 7.1 are used as the parameters of the truth table in the 'Combinatorial Logic' block, as shown below:

[1 0 1;1 0 1;1 0 1;1 0 1;1 0 1;1 0 1;1 0 1;1 0 1;0 0 0;0 0 0;0 1 1;0 1 0;1 0 1;1 0 0;1 1 1;1 1 1;0 0 0;
0 0 0;1 1 0;1 0 0;0 1 1;0 0 1;1 1 1;1 1 1;0 0 0;0 0 0;0 1 0;1 1 0;0 0 1;1 0 1;1 1 1;1 1 1;0 0 0;0 0 0;
1 0 1;0 0 1;1 1 0;0 1 0;1 1 1;1 1 1;0 0 0;0 0 0;0 0 1;0 1 1;1 0 0;1 1 0;1 1 1;1 1 1;0 0 0;0 0 0;1 0 0;
1 0 1;0 1 0;0 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1;1 1 1]

After the above parameters have been input into the 'Combinatorial Logic' block, the block may be used to simulate the optimum switching table.

Step 6 Implementing the Direct Self Controller

The direct self controller is implemented by linking the models of flux estimation, torque calculation, flux angle encoder and magnitude calculation, hysteresis comparator, and optimum switching table obtained in **Step 1** to **Step 5**, as shown in Figure 7.29.

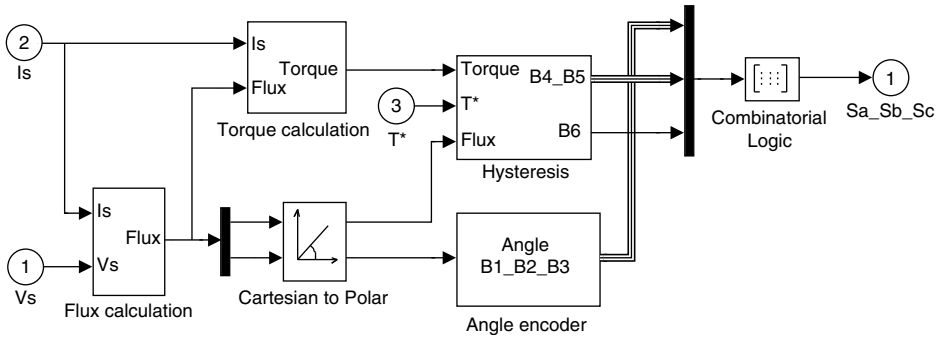


Figure 7.29 Simulink model of direct self controller.

Step 7 Building the Direct Self Control System

To study the performance of the DSC induction motor system, the direct self controller model is combined with the voltage-input model of the induction motor, a 'decode' block, a load block, a 'T*' block, and a 'Transport Delay' block, as shown in Figure 7.30.

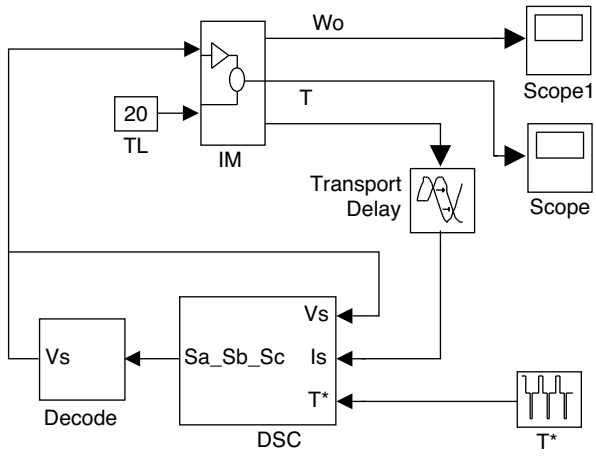


Figure 7.30 Simulink model of direct self control system.

The voltage-input model of the 7.5 kW induction motor used in this study has been built in Programming example 1 in Section 6.8.

The 'decode' block consists of a 'Combinatorial Logic' block and a 'Polar to Cartesian' block in Simulink library, which is used to transform the output voltage codes S_a, S_b, S_c to the corresponding dq-axis voltage vector, as shown in Figure 7.31.

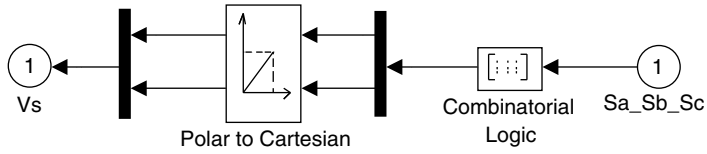


Figure 7.31 'Decode' block for transforming the output voltage codes S_a, S_b, S_c to dq-axis voltage vector.

The parameters in the truth table of the 'Combinatorial Logic' block represent the amplitudes and angles of the six voltage vectors and zero voltage vector. When the amplitude of nonzero voltage vector is 270 V, the parameters will be as follows.

[0 0; 270 −pi/6; 270 −5* pi/6; 270 −pi/2; 270 pi/2; 270 pi/6; 270 5* pi/6; 0 0]

The truth table consists of eight vectors (six nonzero vectors and two zero vectors). The first value of each vector represents the amplitude of the voltage vector and the second value represents the angle of the voltage vector.

The 'T*' block in Figure 7.30 employs a 'Repeating Sequence' block in Simulink library to yield torque commands. To simulate the stator torque commands as listed in Section 7.4, the 'T*' block is provided with the following parameters.

Time value = [0,0.8, 0.8, 2, 2, 2.3, 2.3, 4]
Output value = [100 100, 23, 23, −100, −100, 22, 22]

The 'Transport Delay' block in Figure 7.30 simulates the controller delay. When the controller delay is 100 μs, the parameter of the block is 0.0001.

Step 8 Running the Simulink Model
The Simulink model of direct self control system shown in Figure 7.29 is run with following parameters.

Simulation type: Fixed-step
Fixed-step size = 0.00 002 s
Simulation time = 4 s
Controller delay = 100 μs

The simulation results are obtained as shown in Figures 7.17 and 7.18. When a 'XY Graph' block in Simulink library is connected to the two outputs of the flux angle and magnitude calculation in Figure 7.25, the flux response shown in Figure 7.21 is obtained. To simulate the control system without the controller delay, the parameter of the 'Transport Delay' block should be set to 0.

7.5.2 Programming Example 2: Neural-Network-based Optimum Switching Table

In this example, the optimum switching table is implemented by a two-layer neural network with hard limit neurons. The neural network is trained by a supervised method with perceptron training rule (The MathWorks, Inc., 1994).

Step 1 Entering Samples of Input and Target into Workspace of MATLAB®
From the truth table in Table 7.1, the codes ($B_1\,B_2\,B_3\,B_4\,B_5\,B_6$) have 36 combinations and these are used as input samples for training the neural network. The codes are listed below:

```
B1 B2 B3 B4 B5 B6
0  0  1  0  1  0
0  0  1  0  0  0
0  0  1  1  0  0
0  0  1  0  1  1
0  0  1  1  0  1
0  0  1  0  0  1
0  1  0  0  1  0
```

```
0  1  0  0  0  0
0  1  0  1  0  0
0  1  0  0  1  1
0  1  0  1  0  1
0  1  0  0  0  1
0  1  1  0  1  0
0  1  1  0  0  0
0  1  1  1  0  0
0  1  1  0  1  1
0  1  1  1  0  1
0  1  1  0  0  1
1  0  0  0  1  0
1  0  0  0  0  0
1  0  0  1  0  0
1  0  0  0  1  1
1  0  0  1  0  1
1  0  0  0  0  1
1  0  1  0  1  0
1  0  1  0  0  0
1  0  1  1  0  0
1  0  1  0  1  1
1  0  1  1  0  1
1  0  1  0  0  1
1  1  0  0  1  0
1  1  0  0  0  0
1  1  0  1  0  0
1  1  0  0  1  1
1  1  0  1  0  1
1  1  0  0  0  1
```

Enter the above samples as a [6 × 36] array into the workspace of MATLAB® and name the array as variable 'P' which is used later as input samples for training the network.

In Table 7.1, the output voltage codes (S_a, S_b, S_c) have 36 states corresponding to the 36 input samples. The 36 states are used as target samples for training the neural network and they are listed below.

```
Sa Sb Sc
 0  1  1
 1  1  1
 1  0  1
 0  1  0
 1  0  0
 0  0  0
 1  1  0
 1  1  1
 0  1  1
 1  0  0
 0  0  1
 0  0  0
 0  1  0
 0  0  0
 0  0  1
 1  1  0
 1  0  1
```

```
1  1  1
1  0  1
1  1  1
1  1  0
0  0  1
0  1  0
0  0  0
0  0  1
0  0  0
1  0  0
0  1  1
1  1  0
1  1  1
1  0  0
0  0  0
0  1  0
1  0  1
0  1  1
1  1  1
```

Enter the above samples as a [3 × 36] array into the workspace of MATLAB® and name the array as variable 'T' which is used as target samples.

Step 2 Training the Neural-Network-based Optimum Switching Table

The following program is based on Neural Network Toolbox Version 1.0 (The MathWorks, Inc., 1994), 2.0 or 3.0 on MATLAB® 4.x or MATLAB® 5.x platforms.

```
S1 = 23;              % Set 23 neurons in first layer
tp = [100 500];         % Set training parameters to show results of 100 epochs
                % Max training epochs is 500
[W1,b1] = initp(P,S1);     % Initialize weight and bias of first-layer net
[W2,b2] = initp(S1,T);     % Initialize weight and bias of second-layer net
nntwarn off               % Turn off the warning
A1 = simup(P,W1,b1);       % Yield data from first-layer net
[W2,b2,epochs] = trainp(W2,b2,A1,T,tp);       % Training weight and bias
                                % of the second-layer net
```

The above training program is run repeatedly until the sum squared error (SSE) of the performance program arrived at zero. In some cases, SSE cannot arrive at zero within 500 epochs due to the random initial parameters in the training program. A successful training result with SSE = 0 in 463 epochs is listed as follows.

TRAINP: 0/500 epochs, SSE = 58.
TRAINP: 100/500 epochs, SSE = 31.
TRAINP: 200/500 epochs, SSE = 27.
TRAINP: 300/500 epochs, SSE = 21.
TRAINP: 400/500 epochs, SSE = 20.
TRAINP: 463/500 epochs, SSE = 0.

The network training performance is shown in Figure 7.32.

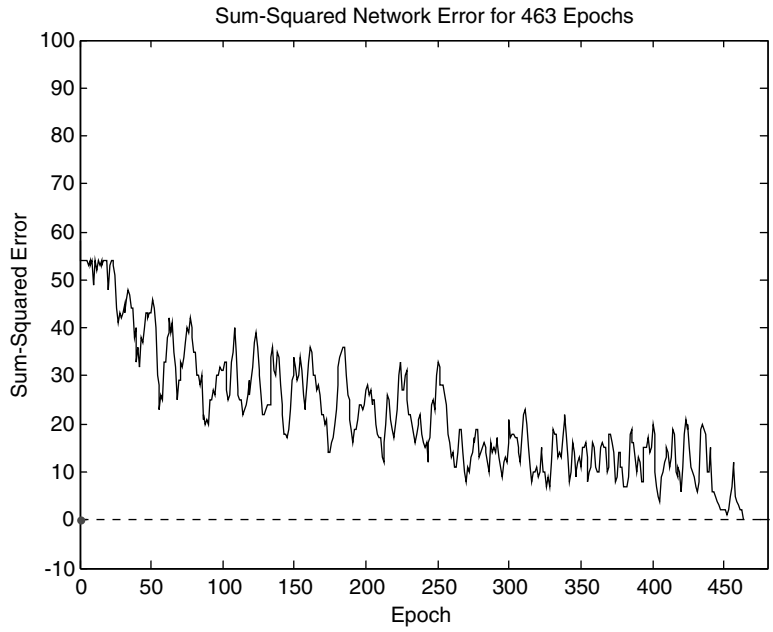


Figure 7.32 Network training performance: the sum squared error (SSE) arrived at zero after 463 training epochs.

Upon entering 'W1', 'b1', 'W2', and 'b2' into the MATLAB® window, the weight matrices and bias vectors of the network are shown on computer screen. They are listed following Figure 7.13 in Section 7.3 with name w_1, θ_1, w_2, and θ_2, respectively.

Step 3 Building the Neural-Network-based Optimum Switching Table

The neural-network-based optimum switching table consists of a 'Matrix Gain' block, a 'Constant' block, a 'Sum' block, and a 'hardlim' block in Simulink library, as shown in Figure 7.33.

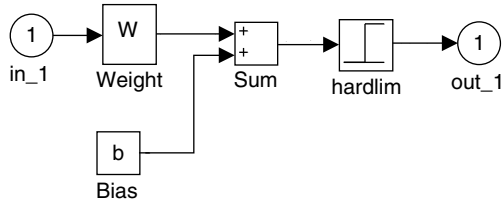


Figure 7.33 A neural network with hard limit neurons.

The 'Matrix Gain' block stores the weight matrix, the 'Constant' block stores the bias vector, and the 'hardlim' block simulates the hard limit neurons and the number of the neurons as determined by the size of weight matrix. The blocks are grouped and duplicated as two

neural-network blocks named as 'Net1' and 'Net2' to simulate the first layer and second layer of the network, respectively. The weight matrices and bias vectors obtained in **Step 2** are directly input into the 'Matrix Gain' blocks and the 'Constant' blocks in the 'Net1' and 'Net2', respectively.

Step 4 Verifying the Neural-Network-based Optimum Switching Table
To verify the neural-network-based optimum switching table, the 'Combinatorial Logic' block that represented the optimum switching table in Figure 7.29 are replaced by 'Net1' and 'Net2'. The modified Simulink model of direct self controller is shown in Figure 7.34.

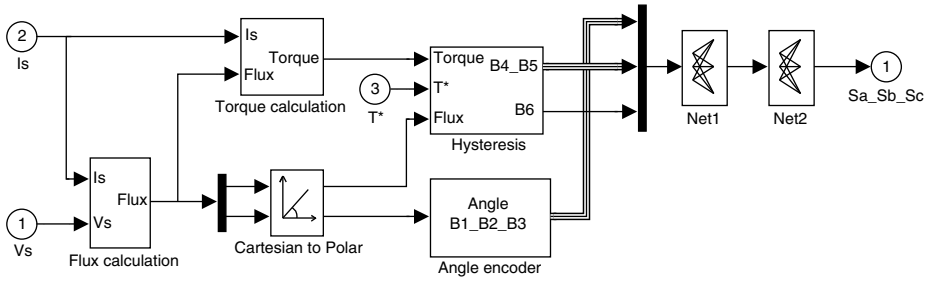


Figure 7.34 Simulink model of direct self controller with neural-network-based optimum switching table.

With a controller delay of 25 μs and other simulation parameters same as those in Example 1, the simulation results of the direct self control system with neural-network-based optimum switching table shown in Figures 7.19 and 7.20 are obtained. When a 'XY Graph' block is connected to the outputs of the flux angle and magnitude calculation in Figure 7.25, the flux response in Figure 7.22 can be viewed.

7.6 Summary

The flexible neural-network structures are used to implement the computationally intensive DSC principle. Based on the understanding of DSC, the fixed weight and supervised networks with individual training strategy are employed for the ANN controller design. Neural-network-based DSC can greatly reduce the execution time of the controller, hence the steady-state control error is eliminated almost completely. The results of simulation demonstrate that the neural-network algorithm has a better precision in the torque and flux responses than the classical DSC method. Using neural-network techniques, hardware implementation of DSC presents less problems and it is envisaged that ANN-DSC induction motor drives will gain wider acceptance in future. The possible developments of the neural-network-based direct self control are (1) to improve the neural network in order to decrease the neuron types, number and layers, (2) to design a neural-network-based sensorless DSC by developing a sub-network of speed estimation, (3) to develop a neural-network-based DSC controller whose performance is immune to disturbances and motor parameter variations.

References

Delgado, A., Kambhampati, C., and Warwick, K. (1995) Dynamic recurrent neural network for system identification and control. *IEE Proceedings – Control Theory and Applications*, **142**(4), 307–314.

Fausett, L. (1994) *Fundamentals of Neural Networks*, Prentice-Hall, Inc., New Jersey.

Habetler, T.G., Profumo, F., Pastorelli, M., and Tolbert, L.M. (1992) Direct torque control of induction machines using space vector modulation. *IEEE Transactions on Industry Applications*, **28**(5), 1045–1053.

Kazmierkowski, M.P. and Kasprowicz, A.B. (1995) Improved direct torque and flux vector control of PWM inverter-fed induction motor drives. *IEEE Transactions on Industrial Electronics*, **42**(4), 344–350.

Kung, S.Y. (1993) *Digital Neural Networks*, PTR Prentice-Hall, Inc., New Jersey.

The MathWorks, Inc. (1994) Neural Network Toolbox User's Guide. The MathWorks, Inc.

Narendra, K.S. and Parthasarathy, K. (1990) Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, **1**, 4–27.

Shi, K.L., Chan, T.F., and Wong, Y.K. (May 1998) Modelling and simulation of direct self control system. IASTED International Conference: Modelling and Simulation, Pittsburgh, USA.

Shi, K.L., Chan, T.F., Wong, Y.K., and Ho, S.L. (2003) Modeling and simulation of direct self control for induction motor. *International Journal of Engineering Education. U.K.*, **19**(4), 646–654.

Simoes, M.G. and Bose, B.K. (1995) Neural network based estimation of feedback signals for a vector controlled induction motor drive. *IEEE Transactions on Industry Applications*, **31**, 620–629.

Takahashi, I. and Noguchi, T. (1986) A new quick-response and high-efficiency control strategy of an induction motor. *IEEE Transactions on Industry Applications*, **22**(5), 820–827.

Zaghloul, M.E., Meador, J.L., and Newcomb, R.W. (1994) *Silicon Implementation of Pulse Coded Neural Networks*, Kluwer Academic Publishers, Boston.

8

Parameter Estimation Using Neural Networks

8.1 Introduction

A three-phase induction motor for drive applications may be described and studied by a set of differential equations. Generally speaking, the differential equations give satisfactory results in motor transient analysis when the input voltages are smooth functions. In a modern machine drive, however, the motor is invariably fed from a power electronic converter, typically a pulse-width modulated (PWM) inverter. Non-differentiable points therefore exist in the motor current due to rapid turn-on or turn-off of the ideal power electronic switches, resulting in possible divergences when the motor drive simulation program is executed. Design and simulation of an induction motor drive require an accurate knowledge of the machine parameters. When differential operators are used for parameter measurement, magnitude of noise may be amplified and the computational accuracy will be degraded. In order to secure convergence in performance analysis of an ac motor drive, integration loops are usually employed instead of the differential operations.

Methods of parameter estimation for an induction motor have been proposed in many publications. In (Moon and Keyhani, 1994), electrical parameters of induction motor are estimated under standstill conditions, using transfer functions with the acquired time-domain data. In (Attaianese *et al.*, 1998), model reference adaptive control (MRAC) is employed to estimate the parameters of an induction motor. In (Zai, Li-Cheng, DeMarco, and Lipo, 1992), the extended Kalman filter (EKF) is used to measure the rotor time constant of an induction motor. Recently, artificial intelligence (AI) techniques, such as fuzzy logic (Bose and Patel, 1998), Artificial Neural Network (ANN) (Wishart and Harley, 1995), and genetic algorithm (Pillay, Nolan, and Haque, 1997), have been applied to motor parameter estimation. Online parameter estimation of an induction motor has been reported (de Souza Ribeiro *et al.*, 2000). However, most published methods using the conventional differential equations have complex structures and some of them may be sensitive to noise. In (Zamora and Garcia-Cerrada, 2000), some integral operators have been used to simplify parameter estimation.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

Among the various AI techniques, neural networks are particularly suitable for motor drive applications as they have the ability to process and acquire information in a complex system. Nonlinearities and uncertainties in parameters of the motor drive system can therefore be modeled with ease with the aid of neural networks. In this chapter, integral models of an induction motor will be implemented by using an artificial neural network (ANN) approach. By using the proposed ANN-based integral models, almost all the machine parameters can be derived directly from the measured data, namely the stator currents, stator voltages and rotor speed. With the estimated parameters, load, stator flux, and rotor speed may be estimated.

8.2 Integral Equations Based on the 'T' Equivalent Circuit

In the stator reference frame, the 'T' equivalent circuit (Slemon, 1989) of an induction motor is shown in Figure A.2 of Appendix A, where

$$i_r = \frac{\lambda_s - L_s i_s}{L_M}. \tag{8.1}$$

Substituting (8.1) into the conventional differential equations in which the state variables are the stator currents and rotor currents, the induction motor 'T' equivalent circuit model (Slemon, 1989) can be expressed in terms of five nonlinear equations in the stator reference frame:

$$\frac{di_{ds}}{dt} = k_T(R_r L_s + L_r R_s) i_{ds} - P\omega_o i_{qs} - k_T R_r \lambda_{ds} - P\omega_o k_T L_r \lambda_{qs} - k_T L_r V_{ds} \tag{8.2}$$

$$\frac{di_{qs}}{dt} = k_T(R_r L_s + L_r R_s) i_{qs} + P\omega_o i_{ds} - k_T R_r \lambda_{qs} + P\omega_o k_T L_r \lambda_{ds} - k_T L_r V_{qs} \tag{8.3}$$

$$\frac{d\lambda_{ds}}{dt} = V_{ds} - R_s i_{ds} \tag{8.4}$$

$$\frac{d\lambda_{qs}}{dt} = V_{qs} - R_s i_{qs} \tag{8.5}$$

$$\frac{d\omega_o}{dt} = \frac{2}{3}\frac{P}{2J}(\lambda_{ds}\, i_{qs} - \lambda_{qs}\, i_{ds}) - \frac{T_L}{J} - \frac{F\omega_o}{J} \tag{8.6}$$

where $k_T = \dfrac{1}{L_M^2 - L_r L_s}$.

The stator fluxes may be obtained by integrating (8.4) and (8.5):

$$\lambda_{ds}(t) - \lambda_{ds}(0) = \int_0^t V_{ds} dt - R_s \int_0^t i_{ds} dt \tag{8.7}$$

$$\lambda_{qs}(t) - \lambda_{qs}(0) = \int_0^t V_{qs}dt - R_s \int_0^t i_{qs}dt. \tag{8.8}$$

With the initial conditions $\lambda_{ds}(0) = 0$ and $\lambda_{qs}(0) = 0$, substitution of Equations (8.7) and (8.8) into Equation (8.2), Equations (8.3) and (8.6) results in the integral forms of i_{ds}, i_{qs}, and ω_o, as follows:

$$i_{ds}(t) - i_{ds}(0) = k_T(R_rL_s + L_rR_s)\int_0^t i_{ds}dt - P\int_0^t \omega_o i_{qs}dt + k_TR_rR_s\int_0^t\left(\int_0^t i_{ds}dt\right)dt$$
$$+ Pk_TL_rR_s\int_0^t \omega_o\left(\int_0^t i_{qs}dt\right)dt - k_TL_r\int_0^t V_{ds}dt - k_TR_r\int_0^t\left(\int_0^t V_{ds}dt\right)dt - Pk_TL_r\int_0^t \omega_o\left(\int_0^t V_{qs}dt\right)dt \quad ;$$

$$i_{qs}(t) - i_{qs}(0) = k_T(R_rL_s + L_rR_s)\int_0^t i_{qs}dt + P\int_0^t \omega_o i_{ds}dt + k_TR_rR_s\int_0^t\left(\int_0^t i_{qs}dt\right)dt$$
$$- Pk_TL_rR_s\int_0^t \omega_o\left(\int_0^t i_{ds}dt\right)dt - k_TL_r\int_0^t V_{qs}dt - k_TR_r\int_0^t\left(\int_0^t V_{qs}dt\right)dt + Pk_TL_r\int_0^t \omega_o\left(\int_0^t V_{ds}dt\right)dt \quad ;$$

$$\omega_o(t) - \omega_o(0) = \frac{2}{3}\frac{P}{2J}R_s\int_0^t\left(i_{ds}\int_0^t i_{qs}dt - i_{qs}\int_0^t i_{ds}dt\right)dt + \frac{2}{3}\frac{P}{2J}\int_0^t\left(i_{qs}\int_0^t V_{ds}dt - i_{ds}\int_0^t V_{qs}dt\right)dt$$
$$- \frac{1}{J}\int_0^t T_Ldt - \frac{F}{J}\int_0^t \omega_o dt \quad .$$

The induction motor can therefore be modeled by the above equations that involve explicitly only three variables, namely, i_{ds}, i_{qs} and ω_o. For convenience, when $i_{ds}(0) = 0$, $i_{qs}(0) = 0$, and $\omega_o(0) = 0$, the above integral equations may be written in the following concise forms:

$$i_{ds} = A_1X_1 + A_2X_2 + A_3X_3 + A_4X_4 + A_5X_5 + A_6X_6 + A_7X_7 \tag{8.9}$$

$$i_{qs} = A_1Y_1 - A_2Y_2 + A_3Y_3 - A_4Y_4 + A_5Y_5 + A_6Y_6 - A_7Y_7 \tag{8.10}$$

$$\omega_o = C_1Z_1 + C_2Z_2 + C_3Z_3 + C_4Z_4 \tag{8.11}$$

where

$$X_1 = \int_0^t i_{ds}dt \qquad Y_1 = \int_0^t i_{qs}dt \qquad A_1 = k_T(R_rL_s + L_rR_s)$$

$$X_2 = \int_0^t \omega_o i_{qs}dt \qquad Y_2 = \int_0^t \omega_o i_{ds}dt \qquad A_2 = -P$$

$$X_3 = \int_0^t \left(\int_0^t i_{ds}dt\right)dt \qquad Y_3 = \int_0^t \left(\int_0^t i_{qs}dt\right)dt \qquad A_3 = k_T R_r R_s$$

$$X_4 = \int_0^t \omega_o \left(\int_0^t i_{qs}dt\right)dt \qquad Y_4 = \int_0^t \omega_o \left(\int_0^t i_{ds}dt\right)dt \qquad A_4 = Pk_T L_r R_s$$

$$X_5 = \int_0^t V_{ds}dt \qquad Y_5 = \int_0^t V_{qs}dt \qquad A_5 = -k_T L_r$$

$$X_6 = \int_0^t \left(\int_0^t V_{ds}dt\right)dt \qquad Y_6 = \int_0^t \left(\int_0^t V_{qs}dt\right)dt \qquad A_6 = -k_T R_r$$

$$X_7 = \int_0^t \omega_o \left(\int_0^t V_{qs}dt\right)dt \qquad Y_7 = \int_0^t \omega_o \left(\int_0^t V_{ds}dt\right)dt \qquad A_7 = -Pk_T L_r$$

$$Z_1 = \int_0^t \left(i_{ds}\int_0^t i_{qs}dt - i_{qs}\int_0^t i_{ds}dt\right)dt \qquad C_1 = \frac{2}{3}\frac{P}{2J}R_s$$

$$Z_2 = \int_0^t \left(i_{qs}\int_0^t V_{ds}dt - i_{ds}\int_0^t V_{qs}dt\right)dt \qquad C_2 = \frac{2}{3}\frac{P}{2J} \qquad (8.12)$$

$$Z_3 = \int_0^t T_L dt \qquad C_3 = -\frac{1}{J}$$

$$Z_4 = \int_0^t \omega_o dt \qquad C_4 = -\frac{F}{J}$$

Equations (8.9)–(8.11) expresses a set of linear functions of i_{ds} i_{qs} and ω_o with variables $X = [X_1, X_2, X_3, X_4, X_5, X_6, X_7]$, $Y = [Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7]$, and $Z = [Z_1, Z_2, Z_3, Z_4]$,

which may in turn be calculated by integral operations on the measured stator voltages, stator currents, and rotor speed. If the coefficients $A_1, A_2, A_3, A_5, A_6, C_2$, and C_4 are known, the parameters P, R_s, L_s, F, J, and the rotor time constant, may be obtained from Equation (8.12):

$$P = -A_2; \quad R_s = -\frac{A_3}{A_6}; \quad L_s = \frac{A_5A_3 - A_1A_6}{A_6^2}; \quad J = \frac{2}{3}\frac{P}{2C_2}; \quad F = -JC_4;$$

$$\frac{R_r}{L_r} = \frac{A_6}{A_5} \quad \text{(Rotor time constant)}. \tag{8.13}$$

If the coefficients, $A_1, A_2, A_3, A_4, A_5, A_6, A_7, C_1, C_2, C_3$, and C_4 are known, the induction motor model is completely described by Equations (8.9)–(8.11).

It is well known that the rotor parameters, R_r and L_r cannot be independently obtained (Stephan, Bodson, and Chiasson, 1994) from the 'T' equivalent circuit because the motor equations are not sufficient for identification of these parameters. Only when the assumption $L_r = L_s$ is made can all the parameters be obtained from Equation (8.12). In order to reduce the complexity without loss in computational accuracy, the 'Γ' equivalent circuit is frequently used instead (Slemon, 1989). Using the 'Γ' equivalent circuit, all the machine parameters can be derived from the corresponding integral equations as detailed in Section 8.3.

8.3 Integral Equations based on the 'Γ' Equivalent Circuit

The 'Γ' equivalent circuit (Slemon, 1989) of an induction motor in the stator reference frame is shown in Figure A.1 of Appendix. As in the case of the 'T' equivalent circuit, the induction motor model based on a 'Γ' equivalent circuit may be described by five nonlinear differential equations in the stator reference frame:

$$\frac{di_{ds}}{dt} = k_\Gamma(L_sR_R + L_RR_s)i_{ds} - P\omega_o\, i_{qs} - k_\Gamma R_R\lambda_{ds} - Pk_\Gamma L_R\omega_o\lambda_{qs} - k_\Gamma L_R V_{ds} \tag{8.14}$$

$$\frac{di_{qs}}{dt} = k_\Gamma(L_sR_R + L_RR_s)i_{qs} + P\omega_o\, i_{ds} - k_\Gamma R_R\lambda_{qs} + Pk_\Gamma L_R\omega_o\lambda_{ds} - k_\Gamma L_R V_{qs} \tag{8.15}$$

$$\frac{d\lambda_{ds}}{dt} = -R_s i_{ds} + V_{ds} \tag{8.16}$$

$$\frac{d\lambda_{qs}}{dt} = -R_s i_{qs} + V_{qs} \tag{8.17}$$

$$\frac{d\omega_o}{dt} = \frac{2}{3}\frac{P}{2J}(\lambda_{ds}i_{qs} - \lambda_{qs}i_{ds}) - \frac{T_L}{J} - \frac{F\omega_o}{J} \tag{8.18}$$

where $k_\Gamma = \dfrac{1}{L_sL_R - L_s^2}$.

The mechanical Equation (8.18) for the 'Γ' equivalent circuit is the same as Equation (8.6) for the 'T' equivalent circuit. The integrals of the stator fluxes Equations (8.16) and (8.17) for the 'Γ' equivalent circuit are the same as Equations (8.7) and (8.8) for the 'T' equivalent circuit. Substituting Equations (8.7) and (8.8) into Equations (8.14), (8.15), and (8.18), the integral forms of the 'Γ' equivalent circuit may be derived:

$$i_{ds}(t) - i_{ds}(0) = k_\Gamma(L_sR_R + L_RR_s)\int_0^t i_{ds}dt - P\int_0^t \omega_o i_{qs}dt + k_\Gamma R_R R_s \int_0^t \left(\int_0^t i_{ds}dt\right)dt$$
$$+ Pk_\Gamma R_s \int_0^t \omega_o \left(\int_0^t i_{qs}dt\right)dt - k_\Gamma L_R \int_0^t V_{ds}dt - k_\Gamma R_R \int_0^t \left(\int_0^t V_{ds}dt\right)dt - Pk_\Gamma L_R \int_0^t \omega_o \left(\int_0^t V_{qs}dt\right)dt \quad ;$$

$$i_{qs}(t) - i_{qs}(0) = k_\Gamma(L_sR_R + L_RR_s)\int_0^t i_{qs}dt + P\int_0^t \omega_o i_{ds}dt + k_\Gamma R_R R_s \int_0^t \left(\int_0^t i_{qs}dt\right)dt$$
$$- Pk_\Gamma R_s \int_0^t \omega_o \left(\int_0^t i_{ds}dt\right)dt - k_\Gamma L_R \int_0^t V_{qs}dt - k_\Gamma R_R \int_0^t \left(\int_0^t V_{qs}dt\right)dt + Pk_\Gamma L_R \int_0^t \omega_o \left(\int_0^t V_{ds}dt\right)dt \quad ;$$

$$\omega_o(t) - \omega_o(0) = \frac{2}{3}\frac{P}{2J}R_s \int_0^t \left(i_{ds}\int_0^t i_{qs}dt - i_{qs}\int_0^t i_{ds}dt\right)dt + \frac{2}{3}\frac{P}{2J}\int_0^t \left(i_{qs}\int_0^t V_{ds}dt - i_{ds}\int_0^t V_{qs}dt\right)dt$$
$$- \frac{1}{J}\int_0^t T_L dt - \frac{F}{J}\int_0^t \omega_o dt \quad .$$

The above integral equations may be written as:

$$i_{ds} = B_1X_1 + B_2X_2 + B_3X_3 + B_4X_4 + B_5X_5 + B_6X_6 + B_7X_7 \tag{8.19}$$

$$i_{qs} = B_1Y_1 - B_2Y_2 + B_3Y_3 - B_4Y_4 + B_5Y_5 + B_6Y_6 - B_7Y_7 \tag{8.20}$$

$$\omega_o = C_1Z_1 + C_2Z_2 + C_3Z_3 + C_4Z_4. \tag{8.21}$$

where $X_1 \sim X_7$, $Y_1 \sim Y_7$, $Z_1 \sim Z_4$ and $C_1 \sim C_4$ are the same as the corresponding variables in Equations (8.9)–(8.11), while $B_1 \sim B_7$ are defined by:

$$\begin{aligned} B_1 &= k_\Gamma(R_R L_s + L_R R_s) \\ B_2 &= -P \\ B_3 &= k_\Gamma R_R R_s \\ B_4 &= P k_\Gamma L_R R_s \\ B_5 &= -k_\Gamma L_R \\ B_6 &= -k_\Gamma R_R \\ B_7 &= -P k_\Gamma L_R \end{aligned} \tag{8.22}$$

It should be noted that the elements of coefficient matrix $[A]$ in Equation (8.9) and the elements of coefficient matrix $[B]$ in Equation (8.19) are identical, despite the fact that corresponding elements may be given by different expressions.

If the coefficients $B_1, B_2, B_3, B_4, B_5, B_6, B_7, C_1, C_2, C_3$, and C_4 are known, all the parameters of the induction motor may be estimated from Equation (8.22):

$$P = -B_2; \quad R_s = -\frac{B_3}{B_6}; \quad L_s = \frac{B_5 B_3 - B_1 B_6}{B_6^2}; \quad L_R = \frac{B_5 L_s^2}{B_5 L_s - 1}; \quad R_R = B_6(L_s L_R - L_s^2);$$

$$J = \frac{2}{3}\frac{P}{2C_2}; \quad F = -JC_4. \tag{8.23}$$

8.4 Parameter Estimation of Induction Motor Using ANN

Accurate knowledge of the parameters of the induction motor model is required for analysis and control purposes. Specifically, we wish to identify the coefficients $A_1 \sim A_7$ in Equation (8.9) or Equation (8.10), $B_1 \sim B_7$ in Equation (8.19) or Equation (8.20), and $C_1 \sim C_4$ in Equation (8.11) or Equation (8.21). An examination of the equations concerned reveals that the quantities i_{ds}, i_{qs}, and ω_o may each be modeled by a single-layer linear network with multiple inputs and a single output. The neural network contains only a neuron which is characterized by a linear transfer function. With obtained sample pairs of input and target, the linear network can be trained by a MATLAB® function 'newlind' in Neural Network Toolbox (The MathWorks Inc., 2008). Specific network weights and biases may be obtained to minimize the mean square error by using this MATLAB® function.

A Simulink model of induction motor drive is used for the parameter estimation as shown in Figure 8.1.

The Simulink model consists of a 'Signal Source' block, a 'SVPWM' block, a 'Load' block, and an 'Induction Motor' block. The 'Signal Source' block outputs three-phase 60 Hz

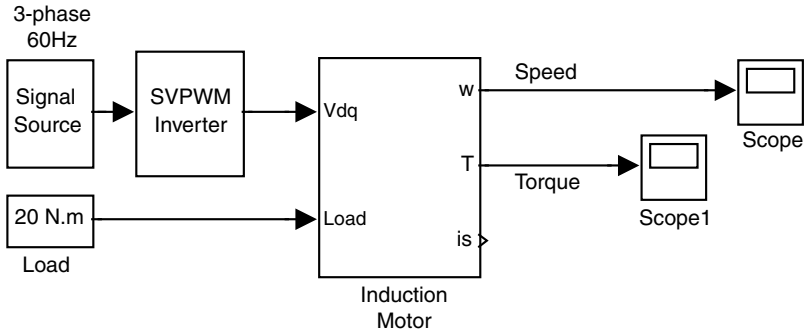


Figure 8.1 Simulink model of induction motor drive with a SVPWM inverter.

sinusoidal signals. The 'Load' block is a 'Constant' block in Simulink library to simulate a 20 N.m load. The 'SVPWM' block simulates a space-vector PWM inverter, which outputs *dq*-axis voltages V_{ds} and V_{qs} with a magnitude of 300 V to drive the induction motor. The 'Induction Motor' block is a voltage-input model of induction motor based on the 'T' equivalent circuit, which is described in Section 3.4 and a programming example in Section 6.8.

8.4.1 Estimation of Electrical Parameters

Figure 8.2 shows a training scheme of neural network with a single linear neuron for obtaining the coefficients $A_1 \sim A_7$ or $B_1 \sim B_7$. The stator voltages (V_{ds}, V_{qs}), stator currents (i_{ds}, i_{qs}), are input to the integral module which gives the seven variables $X_1, X_2, X_3, X_4, X_5, X_6$, and X_7 to the single-layer neural network ANN1. The error between the output of ANN1 and the stator current i_{ds} (target) is used to train the network parameters.

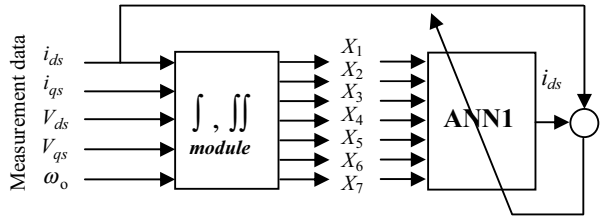


Figure 8.2 Training scheme of ANN-based electrical model of induction motor ($\omega_o \neq 0$).

Upon completion of the training, the coefficients $A_1 \sim A_7$ of the 'T' equivalent circuit (or $B_1 \sim B_7$ of the 'Γ' equivalent circuit) are obtained from the weights of the network ANN1 by Equations (8.13) and (8.23). The biases of the network should be very close to zero if the training process converges. Consequently, ANN1 may be used to give i_{ds} in the integral Equations (8.9) and (8.19) with a feedback line as shown in Figure 8.3.

By using a similar method, a second neural network ANN2 with only a linear neuron may be designed and trained to simulate i_{qs} in the integral Equations (8.10) or (8.20) with a feedback line as shown in Figure 8.3.

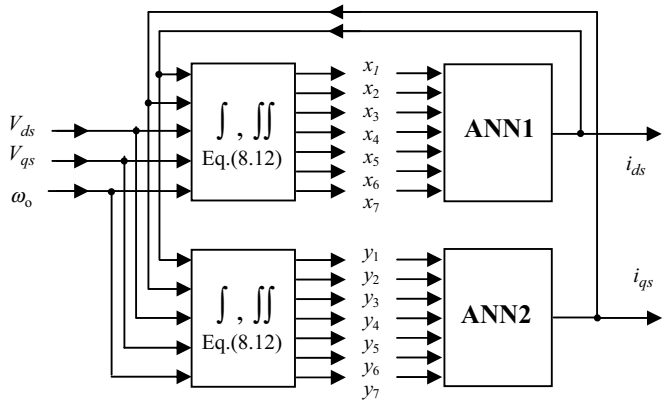


Figure 8.3 ANN-based electrical model of induction motor.

After the training is completed, the ANN-based electrical model of the induction motor is established as shown in Figure 8.3.

The electrical parameters of induction motor may be also estimated without rotor speed information. Under blocked-rotor conditions, that is, rotor speed $\omega_o = 0$, the electrical equations of the induction motor are also valid. Hence, a simple method to obtain the electrical parameters is presented in Figure 8.4. When the rotor speed $\omega_o = 0$, Equations (8.9) and (8.19) are reduced to

$$i_{ds} = A_1X_1 + A_3X_3 + A_5X_5 + A_6X_6 \tag{8.24}$$

$$i_{ds} = B_1X_1 + B_3X_3 + B_4X_4 + B_6X_6. \tag{8.25}$$

where $[A_1, A_3, A_5, A_6] = [B_1, B_3, B_5, B_6]$. Equation (8.24) or (8.25) therefore expresses i_{ds} as a linear function of the variables X_1, X_3, X_5, X_6, obtainable with the motor under blocked-rotor conditions.

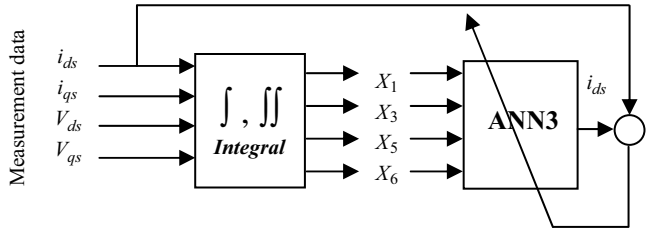


Figure 8.4 Training scheme of estimating electrical parameter of induction motor with condition $\omega_o = 0$.

As shown in Figure 8.4, the stator voltages (V_{ds}, V_{qs}) and stator currents (i_{ds}, i_{qs}) under blocked-rotor conditions are input into the integral module to yield the four outputs X_1, X_3, X_5,

and X_6 to the neural network ANN3. After the training of ANN3 is completed, the coefficients A_1, A_3, A_5, and A_6 are determined and the electrical parameters in the 'T' equivalent circuit are obtained from Equation (8.13).

If the number of poles P is known, the remaining three coefficients A_2, A_4, and A_7 in Equation (8.9) may be computed from the coefficients, A_1, A_3, A_5, and A_6 already obtained:

$$A_2 = -P; \quad A_4 = P\frac{A_3A_5}{A_6}; \quad A_7 = PA_5.$$

The coefficients B_1, B_3, B_5, and B_6 in the 'Γ' equivalent circuit are also equal to the weights of the linear network ANN3. Consequently, all the electrical parameters in the 'Γ' equivalent circuit may be obtained from Equation (8.23). If P is known, the remaining three coefficients B_2, B_4, and B_7 in Equation (8.19) can be determined from the coefficients B_1, B_3, B_5, B_6 already obtained:

$$B_2 = -P; \quad B_4 = P\frac{B_3B_5}{B_6} \quad B_7 = PB_5.$$

When the motor is under blocked-rotor condition, the rotor load equals the rotor torque. With a connection line from the torque port to load port, the Simulink model of induction motor drive in Figure 8.1 may simulate the motor under blocked-rotor condition as shown in Figure 8.5.

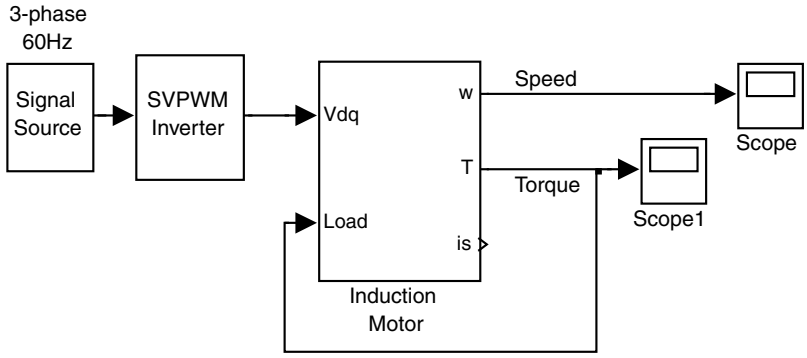


Figure 8.5 Simulink model of induction motor drive with motor under blocked-rotor condition.

8.4.2 *ANN-based Mechanical Model*

Figure 8.6 shows the proposed training scheme for the ANN-based mechanical model. With the load torque T_L set to zero, the network ANN4 with a single linear neuron is used to compute the rotor speed using the inputs Z_1, Z_2, and Z_4 from the integral module. The error between the output of ANN4 and the rotor speed (target) is used to train the network parameters.

After the training is completed, the coefficients C_1, C_2, and C_4 in Equation (8.11) are obtained from the weights of ANN4. All the mechanical parameters of the 'T' or 'Γ' equivalent circuit may be obtained from Equation (8.13). The input Z_3, which is the integral of load torque

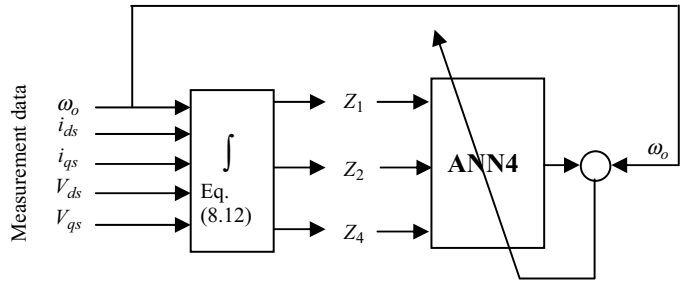


Figure 8.6 Training scheme of ANN-based mechanical model of induction motor with $T_L = 0$.

with weight $(-1/J)$, is then added into the trained network to complete the mechanical model as shown in Figure 8.7.

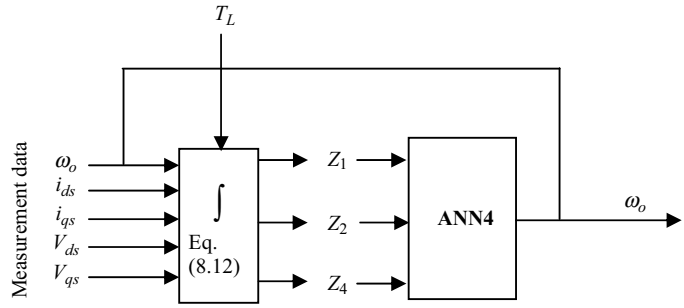


Figure 8.7 Completed ANN-based mechanical model of induction motor.

The four linear neural networks for induction motor parameter estimation are summarized in Table 8.1.

Table 8.1 Four linear neural networks for induction motor parameter estimation.

Name	Function	Input	Output	Neuron and training function
ANN1	Electrical model Equation (8.9), Equation (8.19)	X_1, X_2, X_3, X_4, X_5, X_6, and X_7	i_{ds}	One linear neuron Training function 'newlind'
ANN2	Electrical model Equation (8.10), Equation (8.20)	X_1, X_2, X_3, X_4, X_5, X_6, and X_7	i_{qs}	One linear neuron Training function 'newlind'
ANN3	Electrical model ($\omega_o = 0$) Equation (8.24)	X_1, X_3, X_5, and X_6	i_{ds}	One linear neuron Training function 'newlind'
ANN4	Mechanical model Equation (8.11), Equation (8.21)	Z_1, Z_2, and Z_4	ω_o	One linear neuron Training function 'newlind'

8.4.3 Simulation Studies

Simulation studies are carried out on a three-phase induction motor ('Motor 1' in Appendix B) with the following 'T' equivalent circuit parameters:

Stator resistance	$R_s = 0.294\,\Omega$
Stator inductance	$L_s = 0.0424\,\text{H}$
Mutual inductance	$L_M = 0.041\,\text{H}$
Rotor resistance	$R_r = 0.156\,\Omega$
Rotor inductance	$L_r = 0.0417\,\text{H}$
Total inertia of motor	$J = 0.8\,\text{kg m}^2$
Viscous friction coefficient	$F = 0.1\,\text{N m s/rad}$
Number of poles	$P = 6$

The corresponding electrical parameters of the 'Γ' equivalent circuit are:

Stator resistance	$R_s = 0.294\,\Omega$
Stator inductance	$L_s = 0.0424\,\text{H}$
Rotor resistance	$R_R = 0.167\,\Omega$
Rotor inductance	$L_R = 0.0446\,\text{H}$

The Simulink model of sampling data consists of the induction motor drive in Figure 8.1 and a 'Sample Model' block based on Equation (8.12), as shown in Figure 8.8.

To simulate current measurement noise, two 'Random Number' blocks are introduced to inject Gaussian white noise to the stator currents i_{ds} and i_{qs} before the data is sampled. The 'Equation (8.12)' block outputs the variables $X_1 \sim X_7$ and the variables $Z_1 \sim Z_4$ based on Equation (8.12). The four 'To Workspace Input' blocks are employed to store samples of vector $X = [X_1, X_2, X_3, X_4, X_5, X_6, X_7]$, variable Y (stator current i_{ds}), vector $Z = [Z_1, Z_2, Z_4]$, and variable W (rotor speed ω_o), respectively.

From the motor simulation results, the variables $X_1 \sim X_7$ are obtained from the 'Equation (8.12)' block based on ten sets of data acquired at 0.04 s intervals for training the network ANN1. Since the model in Equation (8.9) is linear, a fairly small number of data samples suffice to ensure convergence in the ANN training. The values of the input samples are listed below:

x_1	0.94692	−0.22678	0.72640	0.71875	−0.79240	0.43001	−0.49282	0.12109	0.14841	−0.72756
x_2	−1.98520	0.58155	−9.75612	15.27895	0.19296	−4.41782	7.49746	−20.83614	19.24096	−18.17365
x_3	0.00755	0.01918	0.03498	0.05994	0.06303	0.05244	0.04821	0.04847	0.04608	0.04035
x_4	0.11399	0.50539	1.19009	2.37681	4.15943	6.02133	8.04367	10.44622	13.30458	16.44377
x_5	0.46359	−0.68192	0.69432	−0.41458	−0.01956	0.42286	−0.69515	0.64079	−0.41185	−0.08396
x_6	0.00459	0.00311	0.00363	0.00600	0.00258	0.00572	0.00320	0.00237	0.00355	−0.00103
x_7	0.04492	0.20875	0.41957	0.79826	1.21903	1.72559	2.41915	3.03928	3.94426	4.82695

The following ten samples of the stator current extracted over the same time period are used as the targets for the training:

y	43.33115	−197.23973	292.50222	−278.07246	127.70671	40.97748	−190.20088	275.99749	−258.02238	138.12246

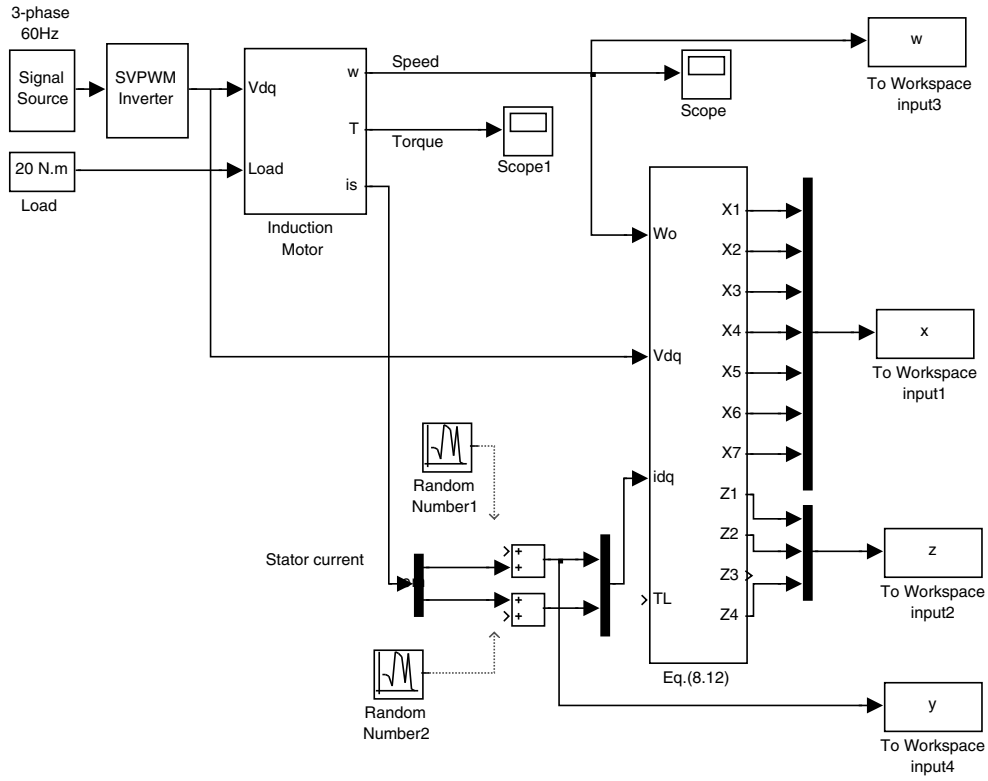


Figure 8.8 Simulink model of sampling data.

where y is i_{ds}.

With the above sample pairs of input and target, the linear network is trained by a MATLAB® function 'newlind' (The MathWorks Inc., 2008). The following is a program for training the linear network.

```
P=6;                                    %Pole number of motor is 6
y1=y(2:11)';
x1=x(2:11,:)';
net = newlind(x1,y1);                   %Training neural network
Y = sim(net,x1);                        %Y is outputs of the trained network
E=y1-Y;                                 %Error of Y and sample outputs
SSE=sumsqr(E)                           %Calculate sum squared error
Weight=net.IW{1,1}                      %Output weight of network
A=Weight;
Biase=net.b{1}                          %Output bias of network
Ls=(A(5)*A(3)-A(1)*A(6))/A(6)/A(6)      %Calculate Ls in 'T' equivalent
                                        %circuit based on Equation(8.13)
```

```
Rs=-A(3)/A(6)                          %Calculate Rs in 'T' equivalent
                                       %circuit based on Equation(8.13)
RrLr=A(6)/A(5)                         %Calculate Rr/Lr in 'T' equivalent
                                       %circuit based on Equation(8.13)
LR=A(5)*Ls*Ls/(A(5)*Ls-1)              %Calculate LR in 'Γ' equivalent
                                       %circuit based on Equation (8.23)
RR=A(6)*(Ls*LR-Ls*Ls)                  %Calculate LR in 'Γ' equivalent
                                       %circuit based on Equation (8.23)
```

It should be noted that value of matrix [*A*] in 'T' equivalent circuit and matrix [*B*] in 'Γ' equivalent circuit are identical.

After running above MATLAB® program, the following results are shown in computer screen.

SSE = 1.7565e-020
Weight = [− 216.7; − 3; − 526.7; − 422.4; 478.9; 1791.5; 1436.6]
Bias = − 3.9154e-011
Ls = 0.0424
Rs = 0.2940
RrLr = 3.7410
LR = 0.0446
RR= 0.1668

The sum squared error (SSE) between the output of ANN1 and the target i_{ds} arrives at 1.7565e-020. The network weights are found to be [− 216.7 − 3 − 526.7 − 422.4 478.9 1791.5 1436.6] and the network bias is − 3.9154e-011, which may be considered to be equal to zero. The network weights are the coefficients of [*A*] in the 'T' equivalent circuit or of [*B*] in the 'Γ' equivalent circuit, that is,

$$[A_1, A_2, A_3, A_4, A_5, A_6, A_7] = [-216.7 \quad -3 \quad -526.7 \quad -422.4 \quad 478.9 \quad 1791.5 \quad 1436.6] \tag{8.26}$$

$$[B_1, B_2, B_3, B_4, B_5, B_6, B_7] = [-216.7 \quad -3 \quad -526.7 \quad -422.4 \quad 478.9 \quad 1791.5 \quad 1436.6] \tag{8.27}$$

With the above coefficients of [*A*], the parameters of the 'T' equivalent circuit are obtained from Equation (8.13):

$$P = 6, R_S = 0.294\,\Omega, L_s = 0.0424\,\text{H}, \text{and rotor time constant } R_r/L_r = 3.741\,\text{s}.$$

In a similar manner, with the above coefficients of [*B*], the parameters of the 'Γ' equivalent circuit are obtained from Equation (8.23):

$$P = 6, R_S = 0.294\,\Omega, L_s = 0.0424\,\text{H}, R_R = 0.167\,\Omega, L_R = 0.0446\,\text{H}.$$

For training of ANN4 in Figure 8.7, the following data sets, which are simulation results obtained from the 'Equation (8.12)' block in Figure 8.8, are used as inputs,

z_1	−8.57757	−20.20110	−27.29074	−37.56554	−50.62150	−57.04781	−68.12547	−75.90780	−84.74541	−95.57365
z_2	3.86035	8.91320	12.53849	16.88888	22.20011	25.73052	30.69521	34.73047	39.12689	44.17282
z_4	0.07599	0.29626	0.66223	1.17180	1.82807	2.63694	3.60462	4.73717	6.04035	7.52038

while the following samples of rotor speed, also obtained from the 'Equation (8.12)' block in Figure 8.8, are used as the targets.

ω_o	3.33687	7.39816	11.20479	14.46507	18.06499	22.06658	26.21527	30.44186	34.77437	39.24546

The program for training ANN4 is shown below:

```
P=6;                                    %Pole number of motor is 6
Z= z(2:11,:)';
w1=w(2:11)';
net = newlind(Z,w1);                    %Training neural network
Y = sim(net,Z);                         %Y is outputs of the trained network
E=w1-Y;                                 %Error of Y and sample outputs
SSE=sumsqr(E)                           %Calculate sum squared error
Weight = net.IW{1,1}                    %Output bias of network
Bias = net.b{1}                         %Output bias of network
D=net.IW{1,1};
J=P/3/D(2)                              %Calculate J based on Equation (8.13)
F=-J*D(3)                               %Calculate F based on Equation (8.13)
C=[D(1), D(2), -1/J, D(3)]              %See Equation(8.28)
```

After running the above code, the following results are displayed on the computer screen.

SSE = 8.7306e-025
Weight = [0.7350, 2.5000, −0.1250]
Bias = 7.5608e-013
J = 0.8000
F = 0.1000
C = 0.7350 2.5000 −1.2500 −0.1250

With the sample pairs of input and target obtained, the linear network is trained by the MATLAB® function 'newlind,' the sum squared error (SSE) between the output of ANN4 and the target ω_o arrives at 8.7306e-25. The network weights are [0.735 2.500 −0.1250], and the network bias is 7.5608e-13. The network bias may be considered to be equal to zero. The network weights are the coefficients [*C*] in the mechanical model in Equation (8.11), that is,

$[C_1,\ C_2,\ C_4] = [0.7350 \quad 2.5000 \quad -0.1250]$. From Equation (8.12), $C_3 = -1/J$, hence

$$[C_1, C_2, C_3, C_4] = [0.7350 \quad 2.5000 \quad -1/J \quad -0.1250] \tag{8.28}$$

Since the pole number $P = 6$ (obtained from the weights of ANN1), the following mechanical parameters of the induction motor are obtained from Equation (8.13):

$$J = \frac{2}{3}\frac{P}{2C_2} \text{ and } F = -JC_4, \text{i.e., } J = 0.8 \text{ kg.m}^2 \text{ and } F = 0.1 \text{ N.m.s/rad.}$$

8.5 ANN-based Induction Motor Models

The induction motor model may be implemented by using the ANN-based electrical model (Figure 8.3) and mechanical model (Figure 8.7), as shown in Figure 8.9.

A double integral of a sinusoidal function may result in data overflow in the computer memory, caused by the second integral operation on an integration constant which has arisen from the first definite integral. When the period of program simulation is long, data overflow may occur when evaluating the double integrals in Equation (8.12).

To overcome the above difficulty, Equations (8.9) to (8.12) are rewritten as

$$i_{ds}(t) - i_{ds}(0) = \int_0^t (A_1X_1' + A_2X_2' + A_3X_3' + A_4X_4' + A_5X_5' + A_6X_6' + A_7X_7')dt \tag{8.29}$$

$$i_{qs}(t) - i_{qs}(0) = \int_0^t (A_1Y_1' - A_2Y_2' + A_3Y_3' - A_4Y_4' + A_5Y_5' + A_6Y_6' - A_7Y_7')dt \tag{8.30}$$

$$\omega_o(t) - \omega_o(0) = \int_0^t (C_1Z_1' + C_2Z_2' + C_3Z_3' + C_4Z_4')dt \tag{8.31}$$

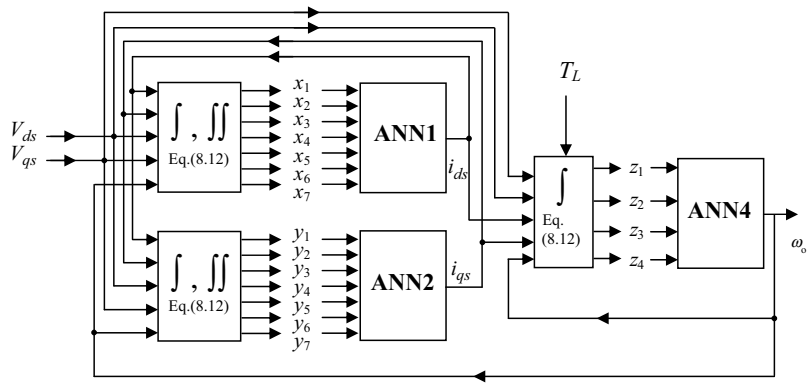


Figure 8.9 ANN model of induction motor.

where

$$
\begin{aligned}
X'_1 &= i_{ds} & Y'_1 &= i_{qs} & A_1 &= k_T(R_rL_s + L_rR_s)\\
X'_2 &= \omega_o i_{qs} & Y'_2 &= \omega_o i_{ds} & A_2 &= -P\\
X'_3 &= \int_0^t i_{ds}dt & Y'_3 &= \int_0^t i_{qs}dt & A_3 &= k_T R_r R_s\\
X'_4 &= \omega_o \int_0^t i_{qs}dt & Y'_4 &= \omega_o \int_0^t i_{ds}dt & A_4 &= Pk_T L_r R_s\\
X'_5 &= V_{ds} & Y'_5 &= V_{qs} & A_5 &= -k_T L_r\\
X'_6 &= \int_0^t V_{ds}dt & Y'_6 &= \int_0^t V_{qs}dt & A_6 &= -k_T R_r\\
X'_7 &= \omega_o \int_0^t V_{qs}dt & Y'_7 &= \omega_o \int_0^t V_{ds}dt & A_7 &= -Pk_T L_r
\end{aligned}
\tag{8.32}
$$

$$
Z'_1 = i_{ds}\int_0^t i_{qs}dt - i_{qs}\int_0^t i_{ds}dt \qquad C_1 = \frac{2}{3}\frac{P}{2J}R_s
$$

$$
Z'_2 = i_{qs}\int_0^t V_{ds}dt - i_{ds}\int_0^t V_{qs}dt \qquad C_2 = \frac{2}{3}\frac{P}{2J}
$$

$$
Z'_3 = T_L \qquad C_3 = -\frac{1}{J}
$$

$$
Z'_4 = \omega_o \qquad C_4 = -\frac{F}{J}
$$

Note that only simple integral operators are involved when evaluating the variables X, Y and Z in Equation (8.32). Based on Equations (8.29)–(8.32), an improved ANN model for the induction motor is shown in Figure 8.10.

For comparison in simulation studies, the ANN model in Figure 8.10 is packed in a Simulink block (ANN IM) and is connected in parallel with the conventional induction motor model (Figure 8.1), as shown in Figure 8.11.

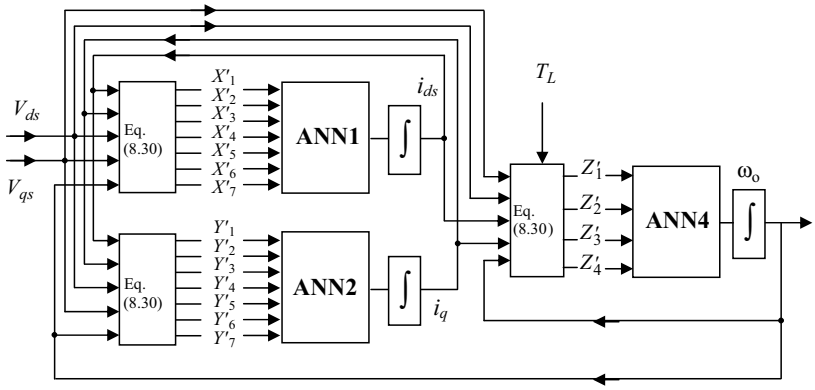


Figure 8.10 ANN model of induction motor without involving double integrals.

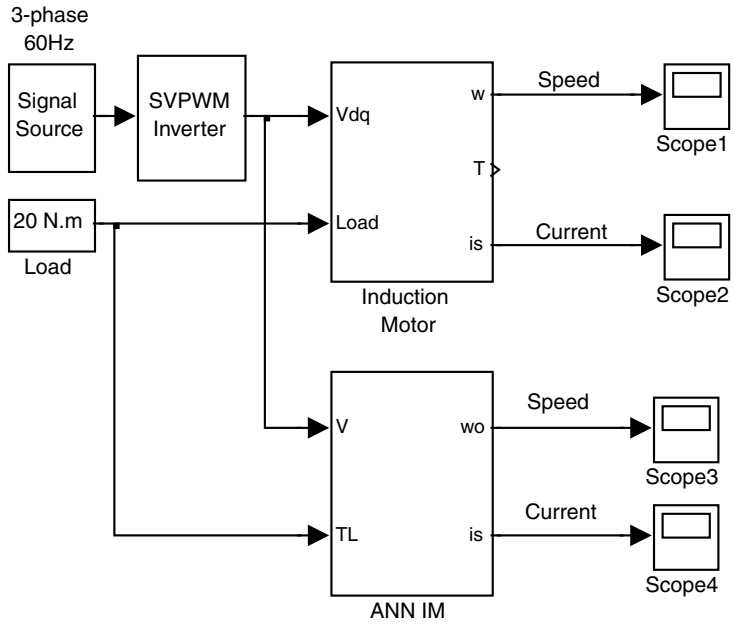


Figure 8.11 Neural network and conventional models of induction motor drive.

A study of direct-on-line starting of the induction motor is made in order to compare the performance of the ANN model with the conventional model in predicting motor transient operation. At time $t = 0$ the motor is at standstill with a constant load torque.

Pertinent data for the simulation study are shown below:

Parameters of the induction motor: listed under 'Motor 1' of Appendix B.
Frequency of three-phase sinusoidal signal source = 60 Hz

Amplitude of three-phase sinusoidal signal source = 0.866 V
Switching frequency of SVPWM = 20 kHz
Modulation index = 0.866
Magnitude of the SVPWM output = 300 V
Load torque = 20 N m
The moment of inertia J_L of the load = 0.4 kg m^2
Simulation type: Variable-step
Max-step size = 0.00002 s
Simulation time = 2 s.

The rotor speed computed from the conventional model is shown in Figure 8.12a and that computed from the ANN model is shown in Figure 8.12b. It is noticed that the ANN model gives similar motor run-up performance as the conventional model.

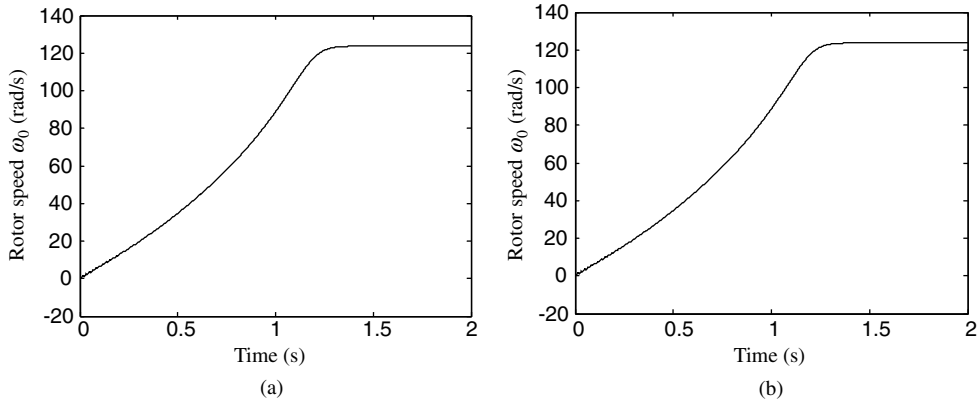


Figure 8.12 Computed motor run-up performance: (a) Conventional model; (b) ANN model.

The above comparison of the conventional model and the ANN model has shown that the ANN modeling approach is feasible. The ANN model can be established through proper training of the neural networks using the data acquired from a simulation model or from an actual motor drive.

8.6 Effect of Noise in Training Data on Estimated Parameters

The data acquired in a practical inverter-fed induction motor drive will usually contain noise. In order to test the efficacy of the ANN-based model under such conditions, Gaussian white noise with various variance values is injected into the stator currents i_{ds} and i_{qs} before the data samples are input to the neural networks for training. The two 'Random Number' blocks in Figure 8.8 are connected and added into the stator currents with various variances. Figure 8.13a shows the current waveform of i_{ds} without noise and Figure 8.13b shows the corresponding waveform when noise with a variance value of unity is present. From Table 8.2, it is noticed that the estimated mechanical parameters J and F are practically unaffected by noise. The error in

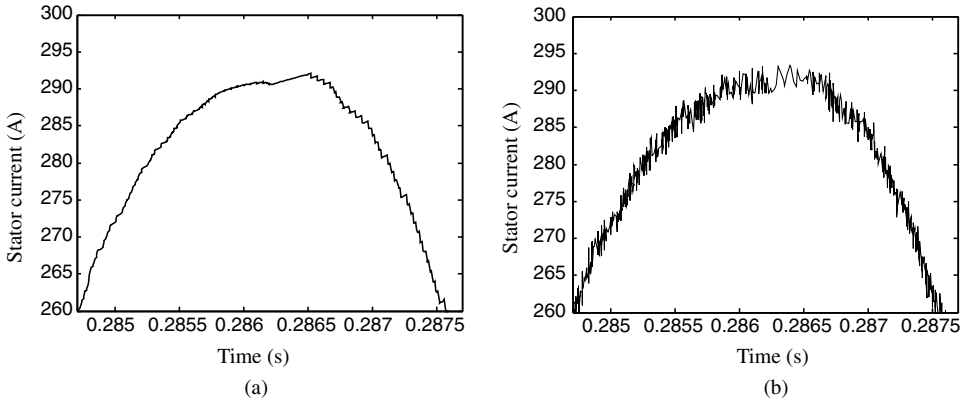


Figure 8.13 Waveform of i_{ds} when motor is fed from an SVPWM inverter: (a) without noise; (b) noise variance equal to 1.

the estimated electrical parameters, on the other hand, increases when the noise content increases. With a noise variance of unity, the estimated values of L_s, L_R and R_R are smaller than the ideal values by 6.1 %, 5.8 %, and 13.4 %, respectively, whereas the estimated value of R_s is larger than the ideal value by 8.4 %. The value of the estimated rotor resistance is therefore more sensitive to noise.

Despite the errors in the individual estimated machine parameters, the ANN model gives satisfactory prediction of the speed response as reflected by the small rotor speed error shown in the right-most column of Table 8.2, where ω_0 and ω_{01} are the rotor speed computed from the ANN model and the conventional model, respectively.

8.7 Estimation of Load, Flux and Speed

Using the estimated parameters of the induction motor obtained in Section 8.5, load, stator flux, and rotor speed may be estimated, respectively.

8.7.1 Estimation of Load

To estimate the load torque, Equation (8.6) is written as

$$T_L = \frac{2}{3}\frac{P}{2}(\lambda_{ds} i_{qs} - \lambda_{qs} i_{ds}) - J\frac{d\omega_o}{dt} - F\omega_o \tag{8.33}$$

Substituting Equations (8.7) and (8.8) into Equation (8.33), load torque T_L may be estimated by the following equation.

$$T_L(t) - T_L(0) = \frac{2}{3}\frac{P}{2}R_s\left(i_{ds}\int_0^t i_{qs}dt - i_{qs}\int_0^t i_{ds}dt\right) + \frac{2}{3}\frac{P}{2}\left(i_{qs}\int_0^t V_{ds}dt - i_{ds}\int_0^t V_{qs}dt\right) - J\frac{d\omega_o}{dt} - F\omega_o \tag{8.34}$$

Table 8.2 Error in parameter estimation when Noise is present in training data.

Variance of noise	Motor Parameters Estimated by ANN1 and ANN4						Rotor speed error (simulation time 0∼2 s, speed range 0∼130 rad/s)	
	Electrical parameters				Mechanical parameters		Steady-state (t = 1.5 s) value of rotor speed ω_o (rad/s)	Average error (t = 0∼2 s) of ω_{o1} and ω_o
	L_s (H)	R_s (Ω)	L_R (H)	R_R (Ω)	J (kg m^2)	F (N m s/rad)		
0	0.0424	0.2940	0.0446	0.1667	0.8000	0.1000	125.35	0.0201
0.2	0.0415	0.3046	0.0437	0.1573	0.7999	0.1001	125.65	0.5168
0.4	0.0410	0.3092	0.0432	0.1531	0.7998	0.1002	125.79	0.6998
0.6	0.0406	0.3129	0.0428	0.1498	0.7998	0.1002	125.91	0.8666
0.8	0.0402	0.3161	0.0424	0.1470	0.7998	0.1003	126.00	0.9942
1	0.0398	0.3188	0.0420	0.1444	0.7998	0.1003	126.07	1.1085

1. Rotor speed error $E = \frac{1}{n}\sum_{i=1}^{n} |\omega_o - \omega_{o1}|$, where ω_o is rotor speed computed using the ANN model, ω_{o1} is rotor speed computed using the conventional model, and n is sample number during 0∼2 s.
2. Without current noise injection, steady-state (t = 1.5 s) value of rotor speed using the conventional model, $\omega_{o1} = 125.335$ rad/s.

With the motor parameters obtained in Section 8.4, the load torque of the induction motor may be calculated by Equation (8.34). A Simulink model of load estimation is shown in Figure 8.14.

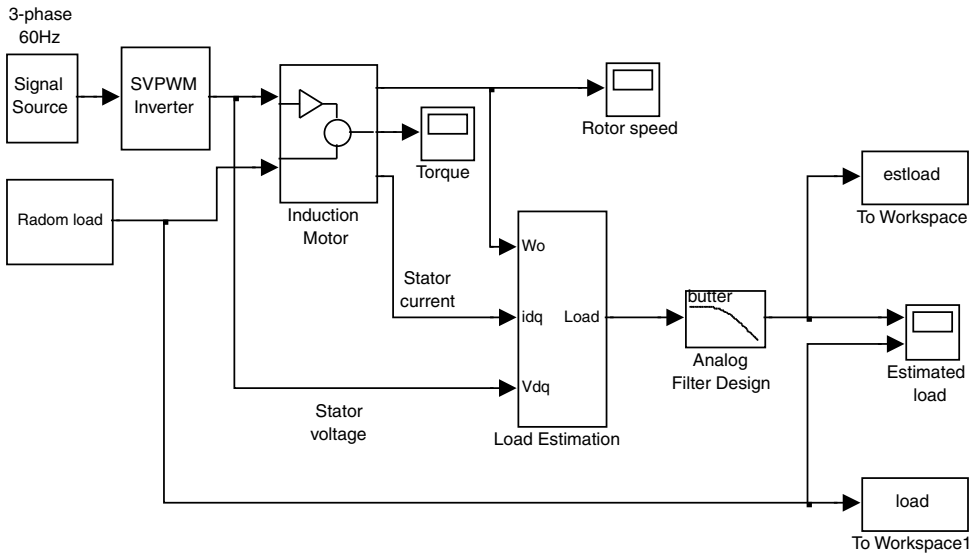


Figure 8.14 A Simulink model of load estimation.

The model consists of the induction motor drive (Figure 8.1), a 'Load Estimation' block, and an 'Analog Filter Design' block in Simulink library. The 'Load Estimation' block is used to estimate the load torque based on Equation (8.34). The 'Analog Filter Design' block is used to reduce the differential noise in the estimated load and noise caused by the differential operation of rotor speed in Equation (8.34).

The Simulink model of load estimation as shown in Figure 8.14 is run with the following parameters.

Parameters of the induction motor: listed under 'Motor 1' of Appendix B.
Frequency of three-phase sinusoidal signal source = 60 Hz
Amplitude of three-phase sinusoidal signal source = 0.866 V
Switching frequency of SVPWM = 20 kHz
Modulation index = 0.866
Magnitude of the SVPWM output = 300 V
The moment of inertia J_L of the load = 0.4 kg m^2
Simulation type: Variable-step
Max-step size = 0.00002 s
Simulation time = 0.4 s

With a desired passband frequency of 500 Hz (3142 rad/s), the 'Analog Filter Design' block has following parameters.

Design method: Butterworth
Filter type: Lowpass
Filter order = 1
Passband edge frequency (rad/s) = 3142 rad/s

The simulation results are shown in Figure 8.15 and two arrays 'load' and 'Est_load' are stored in MATLAB® workspace at the end of the simulation.

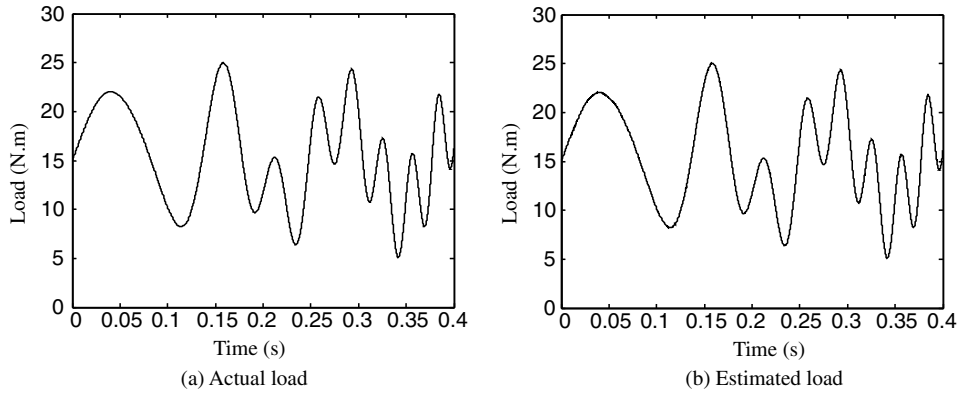


Figure 8.15 Actual load and estimated load from simulation; (a) Actual load; (b) Estimated load.

Mean square error between the actual load torque and the estimated load torque may be calculated by

$$error = \frac{1}{n}\sum_{1}^{n}(load - estload)^2 \tag{8.35}$$

where n is the number of data samples.

The following MATLAB® code is used to calculate the mean square error between the actual load torque and the estimated load torque:

```
s=size(load);                    %Acquire sample number
e=0;
for i=1:s(1)
  e=e+(load(i)-Est_load(i))^2;
end
error=e/s(1)                     %Yield mean square error
```

The calculation result shows that the mean square error of actual load and estimated load is 0.0153 with 155 276 samples taken in 0.4 s.

When measurement noise with variance of 1 A and mean of 0 A are added into the measured stator currents i_{ds} and i_{qs}, the mean square error of actual load and estimated load is 0.1503 with 155 239 samples taken in 0.4 s. The simulation result is shown in Figure 8.16.

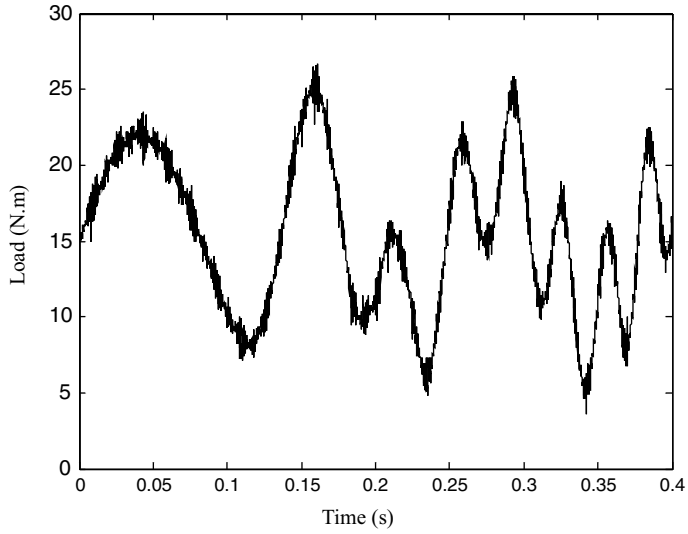


Figure 8.16 Estimated load with noise.

8.7.2 Estimation of Stator Flux

To estimate the stator flux, Equations (8.2) and (8.3) are written in the following concise forms:

$$A_\alpha \lambda_{ds} + B_\alpha \lambda_{qs} = C_\alpha \tag{8.36}$$

$$A_\beta \lambda_{ds} + B_\beta \lambda_{qs} = C_\beta \tag{8.37}$$

where

$$\begin{aligned}
A_\alpha &= k_T R_r \\
B_\alpha &= P\omega_o k_T L_r \\
C_\alpha &= k_T(R_r L_s + L_r R_s) i_{ds} - P\omega_o i_{qs} - k_T L_r V_{ds} - \frac{di_{ds}}{dt} \\
A_\beta &= -P\omega_o k_T L_r \\
B_\beta &= k_T R_r \\
C_\beta &= k_T(R_r L_s + L_r R_s) i_{qs} + P\omega_o i_{ds} - k_T L_r V_{qs} - \frac{di_{qs}}{dt}
\end{aligned}$$

Because $B_\beta = A_\alpha$ and $A_\beta = -B_\alpha$, Equations (8.36) and (8.37) may be written as

$$\begin{bmatrix} A_\alpha & B_\alpha \\ -B_\alpha & A_\alpha \end{bmatrix} \begin{bmatrix} V \\ \lambda_{qs} \end{bmatrix} = \begin{bmatrix} C_\alpha \\ C_\beta \end{bmatrix}. \tag{8.38}$$

The stator flux in Equation (3.38) is solved as follows:

$$\begin{bmatrix} \lambda_{ds} \\ \lambda_{qs} \end{bmatrix} = \begin{bmatrix} \dfrac{A_\alpha C_\alpha - B_\alpha C_\beta}{A_\alpha^2 + B_\alpha^2} \\ \dfrac{B_\alpha C_\alpha + A_\alpha C_\beta}{A_\alpha^2 + B_\alpha^2} \end{bmatrix}. \tag{8.39}$$

From Equation (8.12), the elements in Equation (8.39) are as follows:

$$A_\alpha = -A_6 \tag{8.40}$$

$$B_\alpha = -A_7 \omega_o \tag{8.41}$$

$$C_\alpha = A_1 i_{ds} + A_2 \omega_o i_{qs} + A_5 V_{ds} - \frac{di_{ds}}{dt} \tag{8.42}$$

$$C_\beta = A_1 i_{qs} - A_2 \omega_o i_{ds} + A_5 V_{qs} - \frac{di_{qs}}{dt} \tag{8.43}$$

The following values are obtained from Equation (8.26),

$$A_1 = -216.7$$

$$A_2 = -3$$

$$A_5 = 478.9$$

$$A_6 = 1791.5$$

$$A_7 = 1436.6$$

The elements in Equation (8.39) may be expressed as follows.

$$A_\alpha = -1791.5 \tag{8.44}$$

$$B_\alpha = -1436.6 \omega_o \tag{8.45}$$

$$C_\alpha = -216.7 i_{ds} - 3\omega_o i_{qs} + 478.9\, V_{ds} - \frac{di_{ds}}{dt} \tag{8.46}$$

$$C_\beta = -216.7 i_{qs} + 3\omega_o i_{ds} + 478.9\, V_{qs} - \frac{di_{qs}}{dt} \tag{8.47}$$

Based on Equation (8.39), a Simulink model for estimating stator flux is built as shown in Figure 8.17.

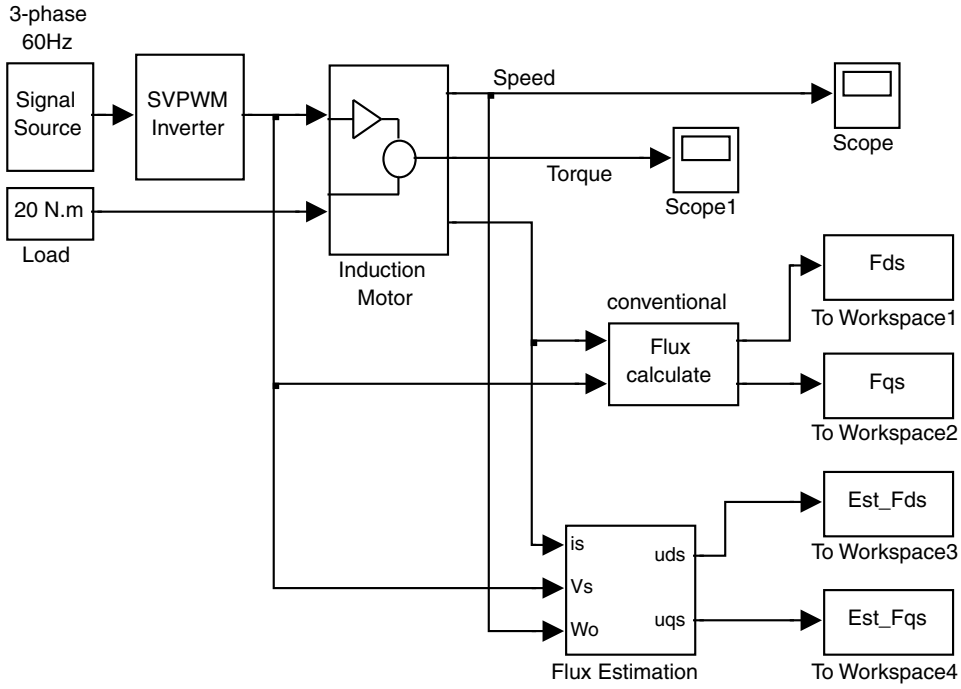


Figure 8.17 Simulink model of stator flux estimation.

The Simulink model consists of the induction motor drive as shown in Figure 8.1, a 'Flux calculate' block and a 'Flux Estimation' block. The 'Flux calculate' block performs a conventional flux estimation based on Equations (8.7) and (8.8). The 'Flux Estimation' block performs the differential flux estimation based on Equation (8.39). Stator flux vector (λ_{ds}, λ_{qs}) yielded by the conventional flux estimation method and flux vector ($\hat{\lambda}_{ds}$, $\hat{\lambda}_{qs}$) yielded by the differential flux estimation method are stored into MATLAB® workspace with four arrays named 'Fds,' 'Fqs,' 'Est_Fds,' and 'Est_Fqs,' respectively.

Due to the SVPWM inverter and the differential operations of *dq-axis* stator currents in Equations (8.50) and (8.51), four low-pass filters are used to reduce ripples of input (V_{ds}, V_{qs}, i_{ds}, i_{qs}) caused by the PWM inverter and two output low-pass filters are used to reduce the ripples of output ($\hat{\lambda}_{ds}$, $\hat{\lambda}_{qs}$) caused by the differential operations. The six filters are simulated by 'Analog Filter Design' block in Simulink library and they are set inside of the 'Flux Estimation' block in Figure 8.17.

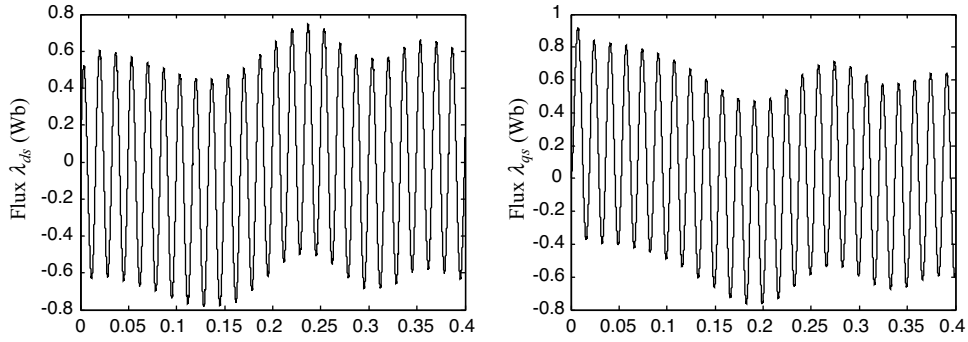
The Simulink model of stator flux estimation in Figure 8.17 is run with following parameters.

Parameters of the induction motor are listed under 'Motor 1' of Appendix B.
Frequency of three-phase sinusoidal signal source is 60 Hz
Amplitude of three-phase sinusoidal signal source is 0.866 V
Switching frequency of SVPWM is 20 kHz
Modulation index is 0.866
Magnitude of the SVPWM output is 300 V
Load = 20 N m
The moment of inertia J_L of the load is 0.4 kg m^2
Simulation type: Variable-step
Max-step size = 0.00002 s
Simulation time = 0.4 s

With a desired passband frequency of 500 Hz (3142 rad/s), the six 'Analog Filter Design' blocks in the 'Flux Estimation' block have following parameters.

Design method: Butterworth
Filter type: Lowpass
Filter order = 1
Passband edge frequency (rad/s) = 3142 rad/s

The simulation results are shown in Figure 8.18 and Figure 8.19 and the four arrays 'uds,' 'uqs,' 'Est_uds,' and 'Est_uqs' are stored in MATLAB® workspace after the simulation run.

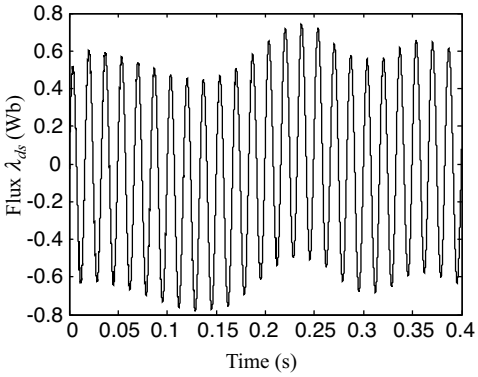


(a) Flux λ_{ds} of conventional estimation (b) Flux λ_{qs} of conventional estimation

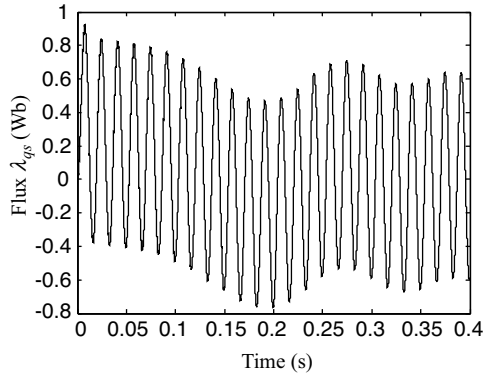
Figure 8.18 Flux vector components λ_{ds} and λ_{qs} of conventional estimation method; (a) Flux λ_{ds} of conventional estimation; (b) Flux λ_{qs} of conventional estimation.

With the two arrays 'Fds' and 'Est_Fds' in MATLAB® workspace, the following MATLAB® program is used to calculate the mean square error of direct-axis flux obtained by the conventional estimation method and the differential estimation method:

```
s=size(Fds);                    %Acquire sample number
e=0;
for i=1:s(1)
  e=e+(Fds(i)-Est_Fds(i))^2;
end
error=e/s(1)                    %Yield mean square error
```



(a) Flux λ_{ds} of differential estimation



(b) Flux λ_{qs} of differential estimation

Figure 8.19 Flux vector components $\hat{\lambda}_{ds}$ and $\hat{\lambda}_{qs}$ of differential estimation method; (a) Flux $\hat{\lambda}_{ds}$ of differential estimation; (b) Flux $\hat{\lambda}_{qs}$ of differential estimation.

The computed result shows that the mean square error of direct-axis flux is 0.0016 with 289 274 samples during 0.4 s.

In a similar manner, the following MATLAB® program is used to calculate the mean square error of quadrature-axis flux obttained from the conventional estimation method and the differential estimation method:

```
s;=size(Fqs);                   %Acquire sample number
e=0;
for i=1:s(1)
  e=e+(Fqs(i)-Est_Fqs(i))^2;
end
error=e/s(1)                    %Yield mean square error
```

The computed result shows that the mean square error of quadrature-axis flux is also 0.0016 with 289 274 samples during 0.4 s.

The differential estimation method does not suffer from error accumulation caused by integrators in conventional flux estimation (Shi *et al.*, 1999).

8.7.3 Estimation of Rotor Speed

Using the estimated parameters of the induction motor obtained in Section 8.5, we can also estimate the rotor speed.

With $\lambda_{ds}(0)=0$, $\lambda_{qs}(0)=0$, substitution of Equations (8.7) and (8.8) into Equations (8.2) and (8.3) separately enables the following equations to be derived:

$$\frac{di_{ds}}{dt} = k_T(R_rL_s + L_rR_s)i_{ds} - P\omega_o i_{qs} + k_T R_r R_s \int_0^t i_{ds}dt + Pk_T L_r R_s \omega_o \int_0^t i_{qs}dt - k_T L_r V_{ds} - k_T R_r \int_0^t V_{ds}dt - Pk_T L_r \omega_o \int_0^t V_{qs}dt \quad ; \tag{8.48}$$

$$\frac{di_{qs}}{dt} = k_T(R_rL_s + L_rR_s)i_{qs} + P\omega_o i_{ds} + k_T R_r R_s \int_0^t i_{qs}dt - Pk_T L_r R_s \omega_o \int_0^t i_{ds}dt - k_T L_r V_{qs} - k_T R_r \int_0^t V_{qs}dt + Pk_T L_r \omega_o \int_0^t V_{ds}dt \quad ; \tag{8.49}$$

$$\omega_o\left(Pk_T L_r \int_0^t V_{qs}dt + Pi_{qs} - Pk_T L_r R_s \int_0^t i_{qs}dt\right) = -k_T L_r V_{ds} - k_T R_r \int_0^t V_{ds}dt + k_T(L_sR_r + L_rR_s)i_{ds} + k_T R_r R_s \int_0^t i_{ds}dt - \frac{di_{ds}}{dt} \quad ; \tag{8.50}$$

$$\omega_o\left(-Pk_T L_r \int_0^t V_{ds}dt - Pi_{ds} + Pk_T L_r R_s \omega_o \int_0^t i_{ds}dt\right) = -k_T L_r V_{qs} - k_T R_r \int_0^t V_{qs}dt + k_T(L_sR_r + L_rR_s)i_{qs} + k_T R_r R_s \int_0^t i_{qs}dt - \frac{di_{qs}}{dt} \quad . \tag{8.51}$$

For convenience, the above integral equations may be written in the following concise forms.

$$\omega_o(-A_7X_1 - A_2X_2 - A_4X_3) = A_5X_4 + A_6X_5 + A_1X_6 + A_3X_7 + A_8X_8 \tag{8.52}$$

$$\omega_o(A_7Y_1 + A_2Y_2 + A_4Y_3) = A_5Y_4 + A_6Y_5 + A_1Y_6 + A_3Y_7 + A_8Y_8 \tag{8.53}$$

where

$$\begin{aligned}
X_1 &= \int_0^t V_{qs}dt \quad Y_1 = \int_0^t V_{ds}dt \quad A_7 = -Pk_TL_r \\
X_2 &= i_{qs} \quad Y_2 = i_{ds} \quad A_2 = -P \\
X_3 &= \int_0^t i_{qs}dt \quad Y_3 = \int_0^t i_{ds}dt \quad A_4 = Pk_TL_rR_s \\
X_4 &= V_{ds} \quad Y_4 = V_{qs} \quad A_5 = -k_TL_r \\
X_5 &= \int_0^t V_{ds}dt \quad Y_5 = \int_0^t V_{qs}dt \quad A_6 = -k_TR_r \\
X_6 &= i_{ds} \quad Y_6 = i_{qs} \quad A_1 = k_T(R_rL_s + L_rR_s) \\
X_7 &= \int_0^t i_{ds}dt \quad Y_7 = \int_0^t i_{qs}dt \quad A_3 = k_TR_rR_s \\
X_8 &= \frac{di_{ds}}{dt} \quad Y_8 = \frac{di_{qs}}{dt} \quad A_8 = -1
\end{aligned} \tag{8.54}$$

Equations (8.52) and (8.53) are functions of the rotor speed and these may further be written as:

$$\omega_o Da = Db \tag{8.55}$$

$$\omega_o Qa = Qb \tag{8.56}$$

where

$$\begin{aligned}
Da &= -A_7X_1 - A_2X_2 - A_4X_3 \\
Db &= A_5X_4 + A_6X_5 + A_1X_6 + A_3X_7 + A_8X_8 \\
Qa &= A_7Y_1 + A_2Y_2 + A_4Y_3 \\
Qb &= A_5Y_4 + A_6Y_5 + A_1Y_6 + A_3Y_7 + A_8Y_8
\end{aligned} \tag{8.57}$$

From Equations (8.49) and (8.50), it is seen that Da and Qb are *q-axis* sinusoidal variables, whereas Qa and Db are *d-axis* sinusoidal variables. From Equations (8.55) and (8.56), $\omega_o = \frac{Db}{Da}$ and $\omega_o = \frac{Qb}{Qa}$. However, they cannot be used to estimate the rotor speed directly because the sinusoidal variables Da and Qa may equal zero. In order to avoid division by zero, the magnitudes of the sinusoidal variables $\sqrt{Da^2 + Qa^2}$ and $\sqrt{Db^2 + Qb^2}$ are employed for estimation of the rotor speed.

Squaring Equations (8.55) and (8.56) adding the resulting equations,

$$\omega_o^2(Da^2 + Qa^2) = (Db^2 + Qb^2) \tag{8.58}$$

Equation (8.58) can thus be written as follows:

$$\omega_o \sqrt{Da^2 + Qa^2} = \sqrt{Db^2 + Qb^2} \tag{8.59}$$

where $\sqrt{Da^2 + Qa^2}$ is magnitude of the *q-axis* sinusoidal variables and $\sqrt{Db^2 + Qb^2}$ is magnitude of the *d-axis* sinusoidal variables.

When magnitude of the sinusoidal variables is larger than zero, the rotor speed may be estimated by

$$\omega_o = \frac{\sqrt{Db^2 + Qb^2}}{\sqrt{Da^2 + Qa^2}}. \tag{8.60}$$

Finally, substitution of Equations (8.57) into (8.60) gives the following equation:

$$\omega_o = \frac{\sqrt{(A_5X_4 + A_6X_5 + A_1X_6 + A_3X_7 + A_8X_8)^2 + (A_5Y_4 + A_6Y_5 + A_1Y_6 + A_3Y_7 + A_8Y_8)^2}}{\sqrt{(-A_7X_1 - A_2X_2 - A_4X_3)^2 + (A_7Y_1 + A_2Y_2 + A_4Y_3)^2}} \tag{8.61}$$

With the value of the matrix [*A*] obtained in Equation (8.26), Equation (8.61) may be used to estimate the rotor speed.

A Simulink model of rotor speed estimation is built as shown in Figure 8.20.

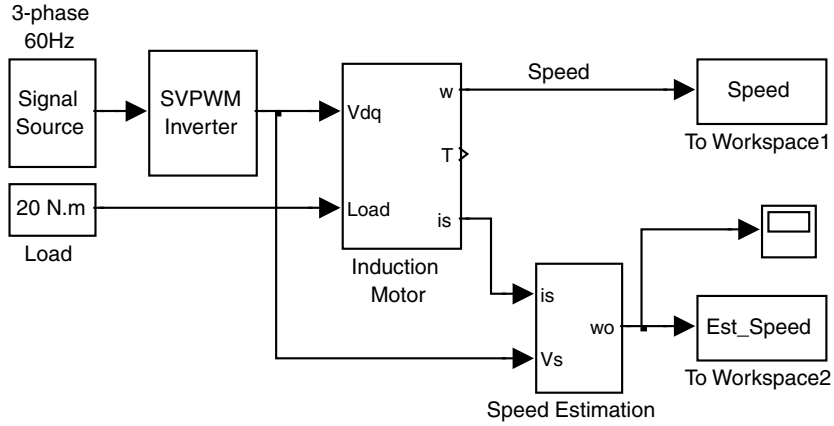


Figure 8.20 Simulink model of rotor speed estimation.

The Simulink model consists of the induction motor block (Figure 8.1) and a 'Speed Estimation' block. The 'Speed Estimation' block is used to estimate the rotor speed according

to Equation (8.61). The 'To Workspace1' and 'To Workspace2' blocks in Figure 8.20 are used to store the actual rotor speed and the estimated speed into MATLAB® workspace with the two arrays 'Speed' and 'Est_Speed,' respectively.

Four low-pass filters are used to reduce ripples of inputs (V_{ds}, V_{qs}, i_{ds}, i_{qs}) caused by the SVPWM inverter (Equations (8.50) and (8.51)) and one output low-pass filter is used to reduce the ripples of the output ω_o. These five filters are simulated by 'Analog Filter Design' blocks in Simulink library and they are set inside of the 'Speed Estimation' block in Figure 8.20.

The Simulink model of rotor speed estimation in Figure 8.20 is run with following parameters.

Parameters of the induction motor are listed under 'Motor 1' of Appendix B.
Frequency of three-phase sinusoidal signal source is 60 Hz
Amplitude of three-phase sinusoidal signal source is 0.866 V
Switching frequency of SVPWM is 20 kHz
Modulation index is 0.866
Magnitude of the SVPWM output is 300 V
Torque = 20 N m
The moment of inertia J_L of the load is 0.4 kg m^2
Simulation type: Variable-step
Max-step size = 0.000001 s
Simulation time = 1.5 s

When the desired passband frequency is 500 Hz (3142 rad/s), the five 'Analog Filter Design' blocks have the following parameters:

Design method: Butterworth
Filter type: Lowpass
Filter order = 1
Passband edge frequency (rad/s) = 3142 rad/s

The simulation results are shown in Figure 8.21 and the two arrays 'Speed' and 'Est_Speed' are stored in MATLAB® workspace.

With the two arrays 'Speed' and 'Est_Speed' in MATLAB® workspace, the following MATLAB® program is used to calculate the mean square error between the actual rotor speed and the estimated speed.

```
s=size(Speed);                    %Acquire sample number
e=0;
for i=1:s(1)
  e=e+(Speed(i)-Est_Speed(i))^2;
end
error=e/s(1)                      %Yield mean square error
```

The result shows that the mean square error of the actual rotor speed the and estimated speed is 0.0132 with 1 908 527 samples during 1.5 s.

The simulation results demonstrate that the rotor speed estimation method has a wide estimation range even near zero speed and the estimation error is small. However, the

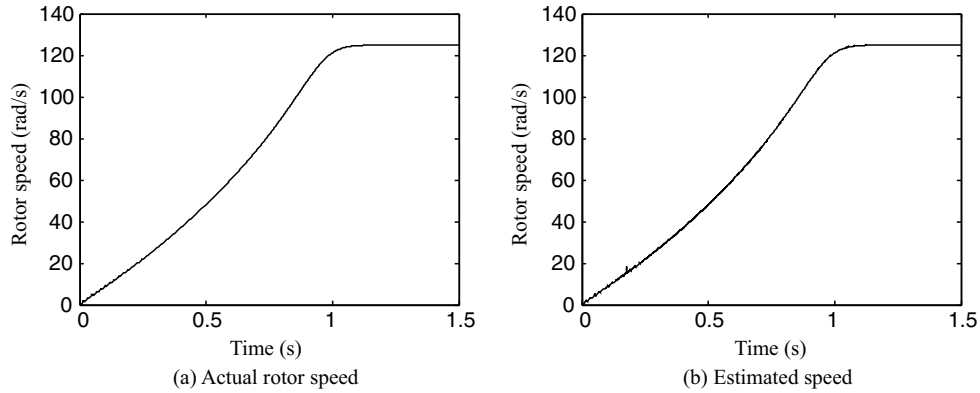


Figure 8.21 Simulation result of rotor speed estimation; (a) Actual rotor speed; (b) Estimated speed.

estimation method yields only absolute value of the rotor speed. Using the model of rotor speed estimation, a sensorless FOC will be given in next section.

8.8 MATLAB®/Simulink Programming Examples

In this section, two examples are given to illustrate programming of a field-oriented control (FOC) system and sensorless control of induction motor using MATLAB®/Simulink.

8.8.1 Programming Example 1: Field-Oriented Control (FOC) System

Field-oriented control (FOC) is a vector control method as both the magnitude and the phase of current and voltage vectors are regulated. Using field-oriented control, an induction motor drive can be made to exhibit satisfactory torque and speed responses because both the flux magnitude and the torque are controlled to desired values. A Simulink model of field-oriented control system of induction motor is built with following steps.

Step 1 Building a Flux Estimation Model
With the given value of stator resistance R_s (0.294 Ω in this example) and based on Equation (8.7), Equation (8.8), and Cartesian to polar transformation block, a Simulink model of flux estimation is built as shown in Figure 8.22.

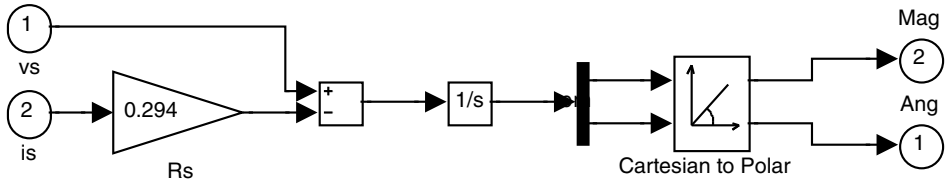


Figure 8.22 Simulink model of flux estimation.

In the flux estimation model, the inputs are the *dq*-axis stator voltage vector and *dq*-axis stator current vector; while the output is magnitude and angle of *dq*-axis flux vector.

Step 2 Building a Field-Oriented Control System Model

A Simulink model of field-oriented control (FOC) system is shown in Figure 8.23.

The field-oriented control system model consists of a 'Speed*' block, a 'Flux*' block, four 'PI' blocks, a 'Park(for)' block, a 'Park(rev)' block, a 'Flux calculation' block, an 'IM' block, and a 'Load' block.

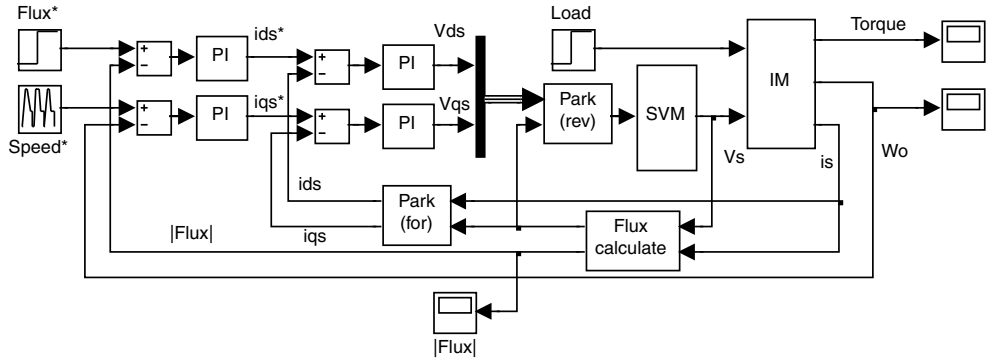


Figure 8.23 Simulink model of field-oriented control system.

The 'IM' block is a voltage-input induction motor model described in Section 3.4 and programmed in Section 6.8.1. The 'Park(for)' and 'park(rev)' blocks are Park's transformation expressed in Equations (3.2) and (3.3). The programming of Park's transformation is described in Section 3.8. The 'Flux calculate' block is built in **Step 1**. The 'SVM' block simulates a space vector PWM inverter which is given on the book companion website. The 'Speed*' block employs a 'Repeating Sequence' block in Simulink library. Pertinent details of these blocks, including their functions and programming references, are summarized in Table 8.3.

Table 8.3 Function and parameters of the blocks in field-oriented control system.

Block Name	Function	Programming References
IM	Voltage-input model of induction motor	Section 3.4 and Section 6.8.1
Park(for)	Park's transformation	Equation (3.2), Section 3.8
Park(rev)	Park's inverse transformation	Equation (3.3), Section 3.8
Flux calculate	Estimation of stator flux	**Step 1**, Equations (8.7) and (8.8)
SVM	space vector PWM inverter	Book companion website
Speed*	Rotor speed command	'Repeating Sequence' block in Simulink library
Flux*	Stator flux command	'Step' block in Simulink library
Load	Load of rotor	'Step' block in Simulink library
PI	Control rotor speed, stator flux, stator currents	'PID' block in Simulink library

Step 3 Running the Simulink Model of Field-Oriented Controller with Parameter Set
The parameters of the motor are listed in Section 8.4.3.

The flux command employs a 'Step' block with following stator flux commands.

$$\left|\lambda_{\mu}^{s}\right|^{*} = 0.86\,\text{Wb} \quad 0\,\text{s} < t \leq 2\,\text{s}$$

The following parameters are input into the 'Step' block.

Step time = 0
Initial value = 0
Final value = 0.86

The speed command employed the 'Repeating Sequence' block with following speed commands.

$\omega_o^* = 0\,\text{rad/s}$	$0\,\text{s} < t \leq 0.1\,\text{s}$	Building flux stage
$\omega_o^* = 0 \sim 50\,\text{rad/s}$	$0.1\,\text{s} < t \leq 0.65\,\text{s}$	Speed ramp rise stage
$\omega_o^* = 50\,\text{rad/s}$	$0.65\,\text{s} < t \leq 1\,\text{s}$	Steady-state stage
$\omega_o^* = 50 \sim 20\,\text{rad/s}$	$1\,\text{s} < t \leq 1.5\,\text{s}$	Speed ramp fall stage
$\omega_o^* = 20\,\text{rad/s}$	$1.5\,\text{s} < t \leq 2\,\text{s}$	Steady-state stage

The following parameters are entered into the 'Repeating Sequence' block.

$$\text{Time values} = [0\ 0.1\ 0.65\ 0.65\ 1\ 1.5\ 2]$$

$$\text{Output values} = [0\ 0\ 50\ 50\ 50\ 20\ 20]$$

The 20 N m load employs a 'Step' block with following parameters.

Step time = 0.1
Initial value = 0
Final value = 20

The parameters of the four PI controllers are listed in Table 8.4.

The parameters of the flux PI controller and the current i_{ds} PI controller are first tuned to guarantee that the magnitude of flux of the motor reaches the flux command value when the speed command equals zero. In this example, the speed command equals zero before 0.1 s during which the flux is building up. From 0.1 s onwards, the flux is controlled. The variation of the stator flux from 0 s to 0.4 s is shown in Figure 8.24. The parameters of the speed PI controller and the current i_{qs} PI controller are next tuned to guarantee a good torque response.

The Simulink model of field-oriented control system shown in Figure 8.23 is run with the following parameters.

Switching frequency of SVPWM = 20 kHz
Magnitude of the SVPWM output = 300 V

Load = 20 N m
Simulation type: Variable-step
Max-step size = 0.000001 s
Simulation time = 2 s

Table 8.4 Parameters of the four PI controllers.

Name	Input	Output	Proportional	Integral
Flux PI controller	Error of magnitude of flux command and estimated flux	current i_{ds} reference	400	1000
Speed PI controller	Error of speed command and rotor speed	current i_{qs} reference	800	100
Current i_{ds} PI controller	Error of current i_{ds} reference and measured current i_{ds}	Stator voltage V_{ds}	1	10
Current i_{qs} PI controller	Error of current i_{qs} reference and measured current i_{qs}	Stator voltage V_{qs}	80	100

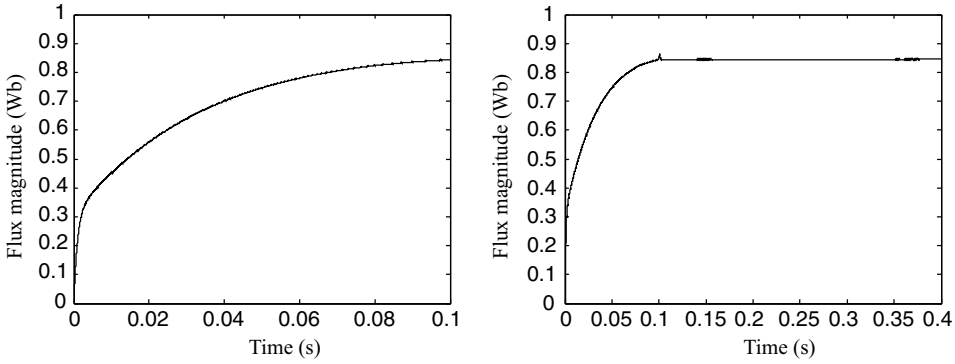


Figure 8.24 Stator flux build-up during 0 ~ 0.1 s and controlled during 0.1 ~ 0.4 s.

The simulation results are shown in Figures 8.25 and 8.26.

8.8.2 Programming Example 2: Sensorless Control of Induction Motor

By connecting the estimator of rotor speed described in Section 8.7.3 to the field-oriented control system in programming example 1, sensorless control of the induction motor may be simulated.

Step 1 Building the Model of a Sensorless Control System
A Simulink model of sensorless control of induction motor is built by a 'Speed Estimation' block and the field-oriented control system described in the programming example 1 as shown in Figure 8.27.

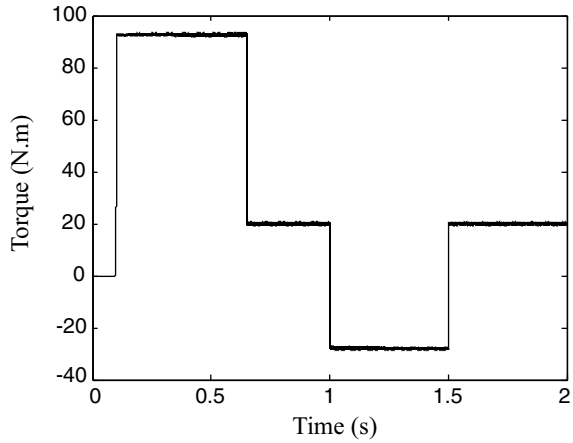


Figure 8.25 Torque of field-oriented control system.

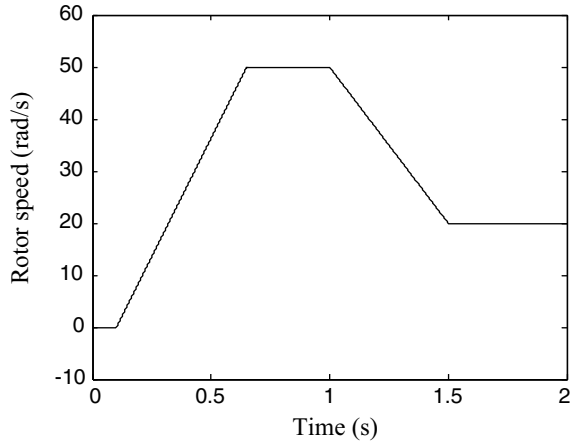


Figure 8.26 Rotor speed of field-oriented control system.

Step 2 Building Sub-models *Da, Db, Qa* and *Qb* in the 'Speed Estimation' Block
Simulink models of *Da*, *Db*, *Qa*, and *Qb* in the 'Speed Estimation' block are built based on Equation (8.57), using the values of A_1 to A_7 obtained from Equation (8.26) and the value of A_8 obtained from Equation (8.54). The models are shown in Figures 8.28–8.31.

Step 3 Building the Speed Estimation Model
With the models of *Da*, *Db*, *Qa*, and *Qb* developed in **Step 2**, the speed estimation model is built as shown in Figure 8.32.

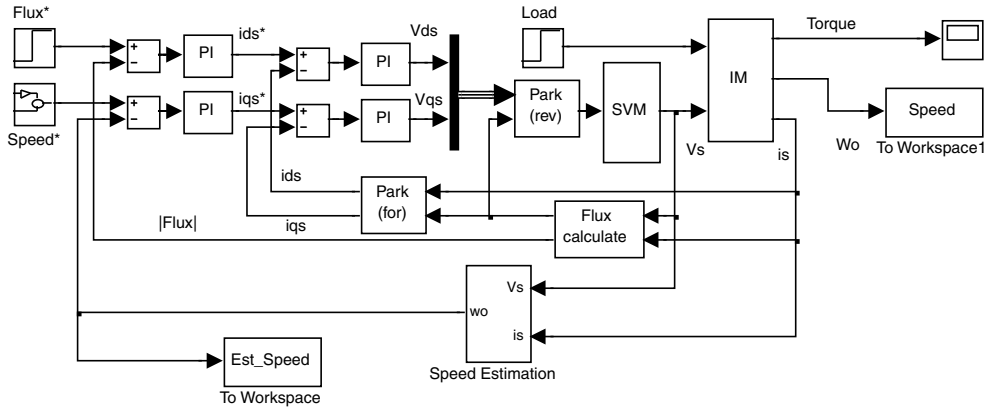


Figure 8.27 Simulink model of sensorless control of induction motor.

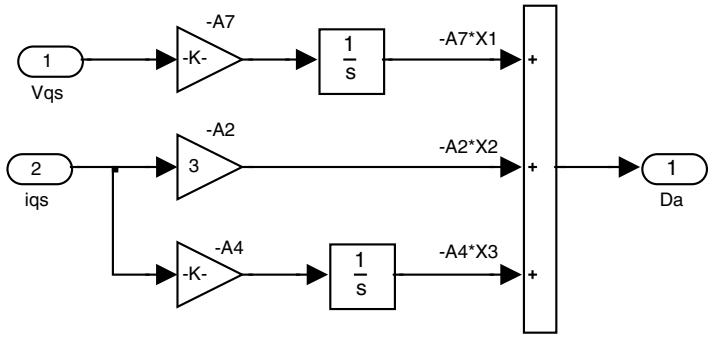


Figure 8.28 Simulink model of *Da*.

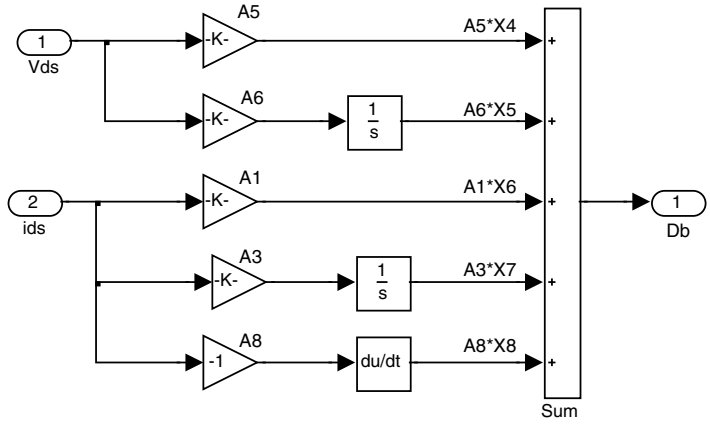


Figure 8.29 Simulink model of *Db*.

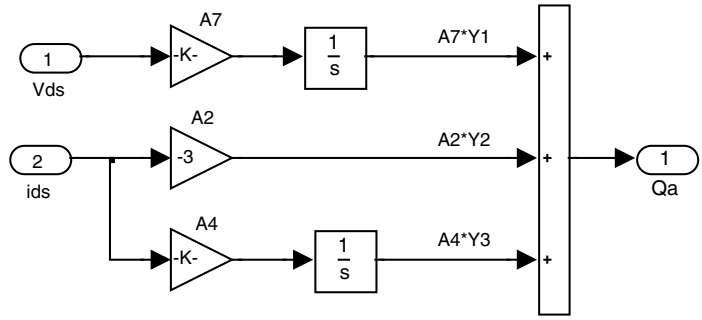


Figure 8.30 Simulink model of *Qa*.

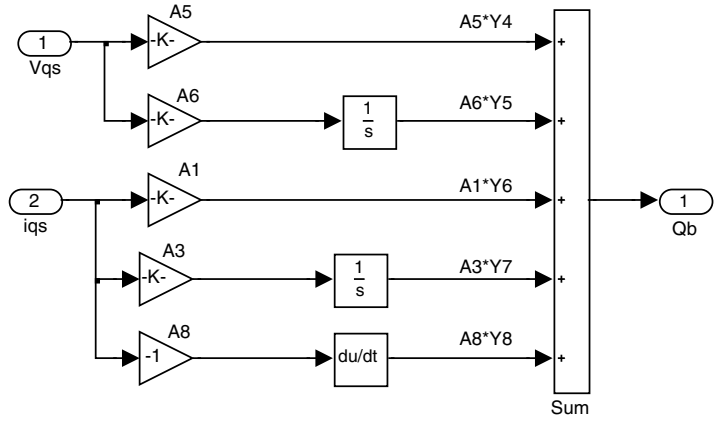


Figure 8.31 Simulink model of *Qb*.

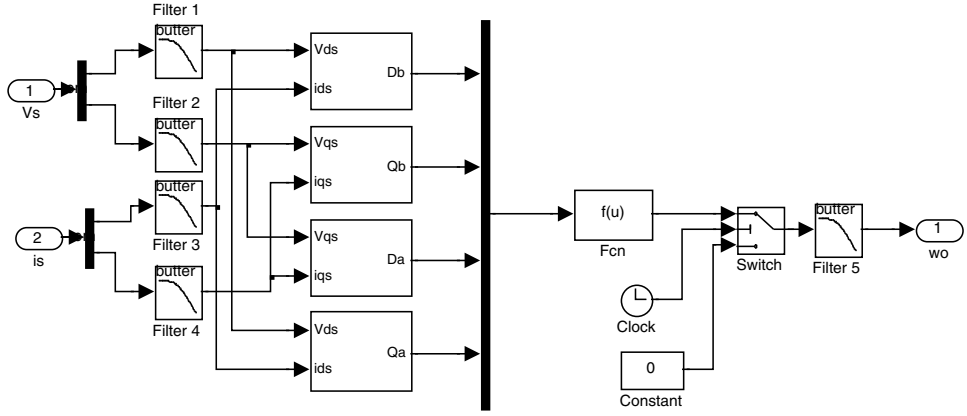


Figure 8.32 Speed estimation model.

The speed estimation model consists of five filter blocks which employ 'Analog Filter Design' block in Simulink library, blocks of *Da*, *Db*, *Qa*, and *Qb* built in **Step 2**, a 'Fcn' block in Simulink library to implement Equation (8.60) with parameters, (sqrt(u(1)^2 + u(2)^2))/ sqrt(u(3)^2 + u(4)^2), a 'Switch' block in Simulink library, a 'Clock' block in Simulink library, and a 'Constant' block in Simulink library. The last three blocks are used to avoid output estimated speed when time is less than 0.1 s. while the speed command equals zero before 0.1 s which is flux setup time. Value of the 'Constant' block is 0 and threshold value of the 'Switch' block is 0.1.

Five analog low-pass Butterworth filters are employed to filter noises of input stator current (i_{ds},i_{qs}), stator voltage, (V_{ds}, V_{qs}), and estimated speed (ω_o). With a desired frequency passband of 500 Hz (3142 rad/s), the five filters have the following parameters.

Design method: Butterworth
Filter type: Lowpass
Filter order = 1
Passband edge frequency (rad/s) = 3142 rad/s

The above parameters are input into the five 'Analog Filter Design' blocks in Figure 8.33.

Step 4 Building the Induction Motor Drive Model with Sensorless Control

With the Simulink model of field-oriented control system built in the example 1 and the Simulink model of speed estimation built in **Step 3**, a Simulink model of sensorless control of induction motor is built as shown in Figure 8.32.

With the two 'To Workspace' blocks in Figure 8.33, realistic rotor speed and estimated rotor speed may be stored into MATLAB® workspace with two arrays named 'Speed' and 'Est_Speed,' respectively.

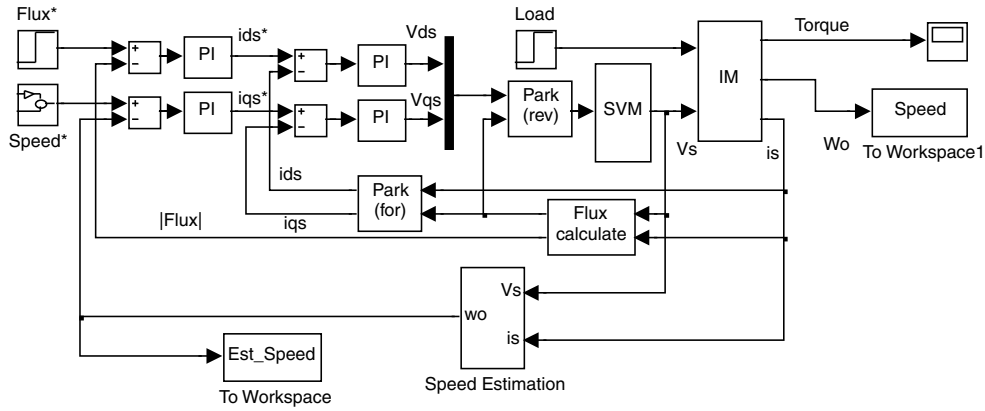


Figure 8.33 Simulink model of sensorless control of induction motor.

Step 5 Running the Simulation Model

The Simulink model of sensorless control of induction motor shown in Figure 8.33 is run with same parameters as in **Step 3** of example 1, but the simulation time is changed to 1 s. Since speed estimation is implemented, the loop gains of the FOC drive are different from those in example 1, as listed in Table 8.5. Using these adjusted parameters for the PI controllers, the FOC sensorless drive is able to give satisfactory performance.

Table 8.5 Parameters of the four PI controllers.

Name	Input	Output	Proportional	Integral
Flux PI controller	Error of magnitude of flux command and estimated flux	current i_{ds} reference	600	1000
Speed PI controller	Error of speed command and rotor speed	current i_{qs} reference	200	400
Current i_{ds} PI controller	Error of current i_{ds} reference and measured current i_{ds}	Stator voltage V_{ds}	1	100
Current i_{qs} PI controller	Error of current i_{qs} reference and measured current i_{qs}	Stator voltage V_{qs}	50	1000

The simulation results are shown in Figures 8.34 and 8.35. The two arrays 'Speed' and 'Est_Speed' are stored in MATLAB® workspace, after the simulation.

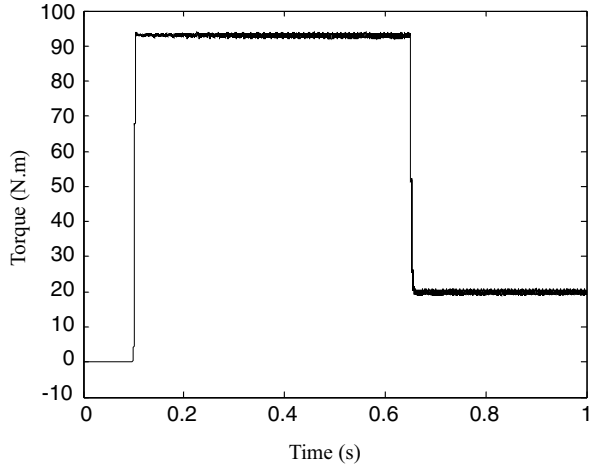


Figure 8.34 Torque of the sensorless control system.

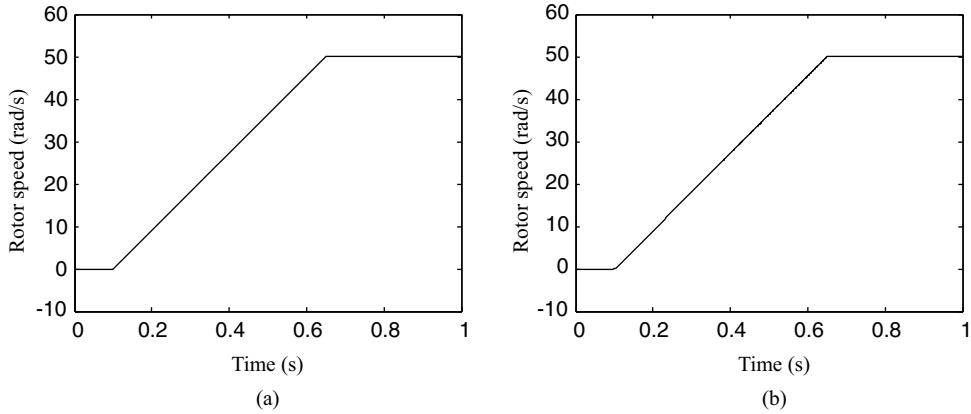


Figure 8.35 (a) Rotor speed of the sensorless control system; (b) Estimated speed used to control the induction motor.

With the two arrays 'Speed' and 'Est_Speed' in MATLAB® workspace, the following MATLAB® program is used to calculate the mean square error between the actual rotor speed and the estimated speed.

```
s=size(Speed);                    %Acquire sample number
e=0;
for i=1:s(1)
  e=e+(Speed(i)-Est_Speed(i))^2;
end
error=e/s(1)                      %Yield mean square error
```

The result shows that the mean square error of actual rotor speed and estimated speed is 0.0019 with 1 224 441 samples during 1 s.

8.9 Summary

The chapter describes an integral model for the induction motor which can be implemented conveniently using artificial neural networks. In the integral model, all the variables are directly measurable from the actual machine. The coefficients of the integral equations may be obtained by proper training of the linear neural networks and a realistic simulation model of the induction motor may be constructed. With the estimated coefficients of the integral equations, load, stator flux, and rotor speed of an induction motor may be estimated. Sensorless control of induction motor is given as a programming example. The simplicity of the neural network structure implies that implementation of online parameter identification is possible based on a general digital signal processor (DSP) technique, in which case the algorithm of network training should be written into the processor for online training. Simulation studies on a typical induction motor have confirmed the validity of the ANN-based integral model.

References

Attaianese, C., Damiano, A., Gatto, G., *et al.* (1998) Induction motor drive parameters identification. *IEEE Transactions on Power Electronics*, **13**(6), 1112–1122.

Bose, B.K., and Patel, N.R., (1998) Quasi-fuzzy estimation of stator resistance of induction motor. *IEEE Transactions on Power Electronics*, **13**(6), 401–409.

de Souza Ribeiro, L.A., Jacobina, C.B., Lima, A.M.N., and Oliveira, A.C., (2000) Real-time estimation of the electric parameters of an induction machine using sinusoidal PWM voltage waveforms. *IEEE Transactions on Industry Applications*, **36**(3), 743–754.

The MathWorks Inc. (2008). Neural Network Toolbox User's Guide. The MathWorks Inc.

Moon, S., and Keyhani, A., (1994) Estimation of induction machine parameters from standstill time-domain data. *IEEE Transactions on Industry Applications*, **30**(6), 1609–1615.

Pillay, P., Nolan, R., and Haque, T., (1997) Application of genetic algorithms to motor parameter determination for transient torque calculations. *IEEE Transactions on Industry Applications*, **33**(5), 1273–1282.

Shi, K.L., Chan, T.F., Wong, Y.K., and Ho, S.L., (1999) A new acceleration control scheme for an inverter-fed induction motor. *Electric Machines and Power Systems*, **27**(5), 527–541.

Slemon, G.R., (1989) Modelling of induction machine for electric drives. *IEEE Transactions on Industry Applications*, **25**(6), 1126–1131.

Stephan, J., Bodson, M., and Chiasson, J., (1994) Real-time estimation of the parameters and flux of induction motors. *IEEE Transaction on Industry Applications*, **30**(3), 746–759.

Wishart, M.T., and Harley, R.G., (1995) Identification and control of induction machines using artificial neural networks. *IEEE Transactions on Industry Applications*, **31**(3), 612–619.

Zai, Li-Cheng, DeMarco, C.L., and Lipo, T.A., (1992) An extended kalman filter approach to rotor time constant measurement in pwm induction motor drives. *IEEE Transactions on Industry Applications*, **28**(1), 96–104.

Zamora, J.L., and Garcia-Cerrada, A., (2000) Online estimation of the stator parameters in an induction motor using only voltage and current measurements. *IEEE Transactions on Industry Applications*, **36**(3), 805–816.

9

GA-Optimized Extended Kalman Filter for Speed Estimation[1]

9.1 Introduction

Elimination of the speed sensor and measurement cables results in a lower cost and increased reliability of an induction motor drive system. Hence, in recent years, speed estimation methods have aroused great interest among induction motor control researchers. Kalman filter is a special kind of observer which provides optimal filtering of the noises in measurement and inside the system if the covariances of these noises are known. When rotor speed (as an extended state) is added into the dynamic model of an induction motor, the extended Kalman filter (EKF) can be used to re-linearize the nonlinear state model for each new estimate as it becomes available. Recently, some advances have been obtained for the extended Kalman filter of induction motor drives (Henneberger, Brunsbach and Klepsch, 1996; Kim, Sul and Park, 1996; Salvatore, Stasi and Tarchioni, 1993; Texas Instruments Incorporated, 1997) and the extended Kalman filter has been considered to be the best solution for the speed estimation of induction motor (Manes, Parasiliti and Tursini, 1994). However, these extended Kalman filters do not yield the best drive performance, due to the fact that the correct noise matrices cannot be obtained from traditional theories. The noise matrices are usually tuned experimentally using a trial-and-error method (Manes, Parasiliti and Tursini, 1994; Bolognani, Oboe and Zigliotto, 1999). In this chapter, a real-coded GA method (Goldberg, 1989; Wright, 1991) is used to optimize the noise matrices for improving the EKF performance. Simulation studies are carried out on a closed-loop V/Hz controller, a sensorless direct self controller (DSC), and a field-oriented controller (FOC) and the effects of stator current noise and machine parameter changes are investigated.

[1] (a) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of induction motor using extended Kalman filter," *IEEE 2000 Winter Meeting*, vol. 1, pp. 243–248, January 23–27, 2000, Singapore. © 2000 IEEE.
(b) Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

9.2 Extended State Model of Induction Motor

A dynamic electrical model for a three-phase induction motor has four state variables, namely the stator currents (i_{ds}, i_{qs}) and the rotor fluxes (λ_{dr}, λ_{qr}). An extended induction motor model results if the rotor speed is included as an extended state variable. The extended model can be expressed as follows (Lewis, 1992):

$$\dot{x} = Ax + Bu + G(t)w(t) \quad \text{(System)} \tag{9.1}$$

$$y = Cx + v(t) \quad \text{(Measurement)} \tag{9.2}$$

where

$$x_n = \begin{bmatrix} i_{ds}^{(n)} \\ i_{qs}^{(n)} \\ \lambda_{dr}^{(n)} \\ \lambda_{qr}^{(n)} \\ \omega_o^{(n)} \end{bmatrix}; \quad y_n = \begin{bmatrix} i_{ds}^{(n)} \\ i_{qs}^{(n)} \end{bmatrix}; \quad u_n = \begin{bmatrix} V_{ds}^{(n)} \\ V_{qs}^{(n)} \end{bmatrix};$$

$$A_n = \begin{bmatrix} 1-\frac{K_r}{K_l}M & 0 & \frac{L_M R_r}{L_r^2 K_l}M & \frac{PL_M\omega_o^{(n)}}{2L_r K_l}M & 0 \\ 0 & 1-\frac{K_r}{K_l}M & \frac{PL_M\omega_o^{(n)}}{2L_r K_l}M & \frac{L_M R_r}{L_r^2 K_l}M & 0 \\ \frac{L_M}{\tau_r}M & 0 & 1-\frac{1}{\tau_r}M & -\frac{P}{2}\omega_o^{(n)}M & 0 \\ 0 & \frac{L_M}{\tau_r}M & \frac{P}{2}\omega_o^{(n)}M & 1-\frac{1}{\tau_r}M & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix};$$

where $\tau_r = L_r/R_r$, $K_r = R_s + L_M^2 R_r/L_r^2$ and $K_l = (1-L_M^2/L_r/L_s)*L_s$.

$$B_n = \begin{bmatrix} \frac{M}{K_l} & 0 \\ 0 & \frac{M}{K_l} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}; C_n = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix};$$

In Equations (9.1) and (9.2), $G(t)$ is the noise-weight matrix, $w(t)$ is noise matrix of state model (system noise), and $v(t)$ is noise matrix of output model (measurement noise). The covariance matrices Q and R of these noises are defined as:

$$Q = \text{cov}(w) = E\{ww^t\} \tag{9.3}$$

$$R = \text{cov}(v) = E\{vv^t\} \tag{9.4}$$

where $E\{\cdot\}$ denotes the expected value.

9.3 Extended Kalman Filter Algorithm for Rotor Speed Estimation

The recursive form of Kalman filter may be expressed by the following system of equations, where all symbols in the formulations denote matrices or vectors (Goldberg, 1989):

9.3.1 Prediction of State

$$x_{n+1\,n} = \Phi(n+1, n, x_{nn-1}, u_n) \tag{9.5}$$

where

$$\Phi(n+1, n, x_{nn-1}, u_n) = A_n(x_{nn})x_{nn} + B_n(x_{nn})u_n. \tag{9.6}$$

9.3.2 Estimation of Error Covariance Matrix

$$P_{n+1\,n} = \left.\frac{\partial \Phi}{\partial x}\right|_{x=x_{nn}} P_{nn} \left.\frac{\partial \Phi^T}{\partial x}\right|_{x=x_{nn}} + \Gamma_n Q \Gamma_n^T \tag{9.7}$$

where

$$\Gamma = \int_n^{n+1} \Phi(t_{n+1}, \tau) G(\tau) d\tau. \tag{9.8}$$

and initial value of P_{nn} is a constant matrix.

9.3.3 Computation of Kalman Filter Gain

$$K_n = P_{nn-1} \left.\frac{\partial H^T}{\partial x}\right|_{x=x_{nn-1}} \left(\left.\frac{\partial H}{\partial x}\right|_{x=x_{nn-1}} P_{nn-1} \left.\frac{\partial H^T}{\partial x}\right|_{x=x_{nn-1}} + R \right)^{-1} \tag{9.9}$$

where

$$H(x_{nn-1}, n) = C_n(x_{nn-1})x_{nn-1}. \tag{9.10}$$

9.3.4 State Estimation

$$x_{nn} = x_{nn-1} + K_n(y_n - H(x_{nn-1}, k)) \tag{9.11}$$

9.3.5 Update of the Error Covariance Matrix

$$P_{nn} = P_{nn-1} - K_n \left.\frac{\partial H}{\partial x}\right|_{x=x_{nn-1}} P_{nn-1} \tag{9.12}$$

Using Equations (9.1), (9.2), (9.6), and (9.10), the matrices Φ, H, $\partial\Phi/\partial x$ and $\partial H/\partial x$ are obtained as follows:

$$\Phi = \begin{bmatrix} \left(1 - \frac{K_r}{K_l}M\right) i_{ds}^{(n)} + \frac{L_M R_r}{L_r^2 K_l} M\lambda_{dr}^{(n)} + \frac{PL_M\omega_o^{(n)}}{2L_r K_l} M\lambda_{qr}^{(n)} + \frac{M}{K_l} V_{ds}^{(n)} \\ \left(1 - \frac{K_r}{K_l}M\right) i_{qs}^{(n)} - \frac{PL_M\omega_o^{(n)}}{2L_r K_l} M\lambda_{dr}^{(n)} + \frac{L_M R_r}{L_r^2 K_l} M\lambda_{qr}^{(n)} + \frac{M}{K_l} V_{qs}^{(n)} \\ \frac{L_M}{\tau_r} M i_{ds}^{(n)} + \left(1 - \frac{1}{\tau_r}M\right)\lambda_{dr}^{(n)} - \frac{P}{2}\omega_o^{(n)} M\lambda_{qr}^{(n)} \\ \frac{L_M}{\tau_r} M i_{qs}^{(n)} + \frac{P}{2}\omega_o^{(n)} M\lambda_{dr}^{(n)} + \left(1 - \frac{1}{\tau_r}M\right)\lambda_{qr}^{(n)} \\ \omega_o^{(n)} \end{bmatrix} \tag{9.13}$$

$$H = C_n x_n = \begin{bmatrix} i_{ds}^{(n)} \\ i_{qs}^{(n)} \end{bmatrix} \tag{9.14}$$

$$\frac{\partial\Phi}{\partial x} = \begin{bmatrix} 1 - \frac{K_r}{K_l}M & 0 & \frac{L_M R_r}{L_r^2 K_l} M & \frac{PL_M\omega_o^{(n)}}{2L_r K_l} M & \frac{L_M}{L_r K_l} M\lambda_{qr}^{(n)} \\ 0 & 1 - \frac{K_r}{K_l}M & -\frac{PL_M\omega_o^{(n)}}{2L_r K_l} M & \frac{L_M R_r}{L_r^2 K_l} M & -\frac{PL_M}{2L_r K_l} M\lambda_{dr}^{(n)} \\ \frac{L_M}{\tau_r} M & 0 & 1 - \frac{1}{\tau_r}M & -\frac{P}{2}\omega_o^{(n)} M & -M\lambda_{qr}^{(n)} \\ 0 & \frac{L_M}{\tau_r} M & \frac{P}{2}\omega_o^{(n)} M & 1 - \frac{1}{\tau_r}M & M\lambda_{dr}^{(n)} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.15}$$

$$\frac{\partial H}{\partial x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{9.16}$$

The speed estimation algorithm of the extended Kalman filter can be simulated by using MATLAB®/Simulink, as shown in Figure 9.1. The EKF algorithm is coded in an M-file (written in the MATLAB® language) which is then placed in the S-function block. The M-file is listed in Appendix F.

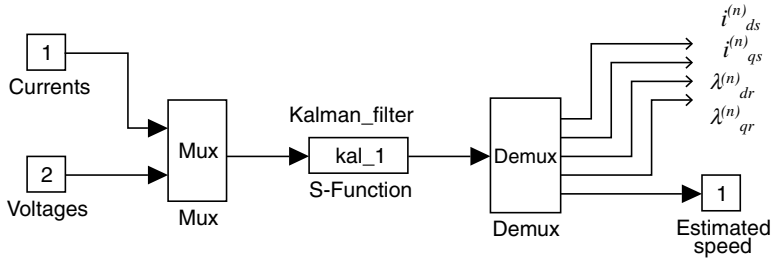


Figure 9.1 Simulink model of Extended Kalman filter speed estimator. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

9.4 Optimized Extended Kalman Filter

To justify the need for an optimized extended Kalman filter, the EKF speed estimation algorithm is applied to a closed-loop constant V/Hz controller (Shi *et al.*, 2002) as shown in Figure 9.2.

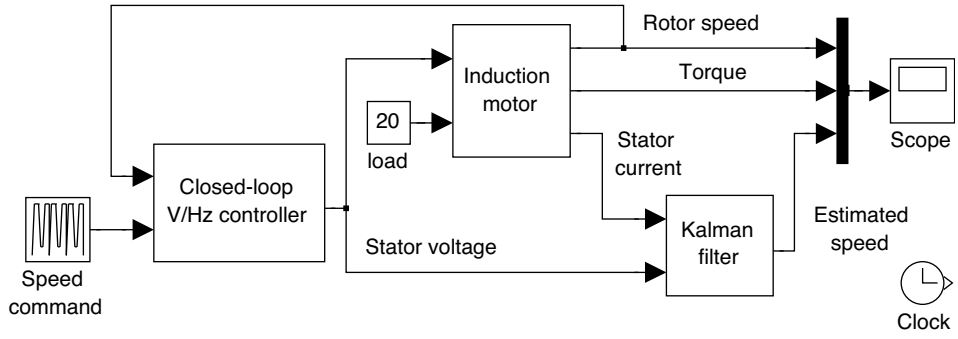


Figure 9.2 Extended Kalman filter speed estimation for a voltage-frequency controlled drive system. Simulink model of Extended Kalman filter speed estimator. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

Details of the Kalman filter block are shown in Figure 9.1 and the induction motor block is the voltage-input model built in Chapter 3, while the block of closed-loop V/Hz controller is shown in Figure 9.3.

Parameters of the induction motor chosen for the simulation studies are listed in 'Motor 1' of Appendix B. It is assumed that the induction motor is taken through the following control cycle:

$$\begin{aligned}
\omega_o^* &= 0 \sim 120\ (\text{rad/s}) && 0\ \text{s} < t \leq 1.5\ \text{s} \\
\omega_o^* &= 120\ (\text{rad/s}) && 1.5\ \text{s} < t \leq 3\ \text{s} \\
\omega_o^* &= 120 \sim 20\ (\text{rad/s}) && 3\ \text{s} < t \leq 4.25\ \text{s} \\
\omega_o^* &= 20\ (\text{rad/s}) && 4.25\ \text{s} < t \leq 6\ \text{s} \\
\omega_o^* &= 20 \sim 120\ (\text{rad/s}) && 6\ \text{s} < t \leq 7.25\ \text{s} \\
\omega_o^* &= 120\ (\text{rad/s}) && 7.25\ \text{s} < t \leq 9\ \text{s}
\end{aligned}$$

In the simulation, initial values of the error covariance matrix P of EKF is set as a unit matrix while the noise covariance matrices R, Q, and noise-weight matrix G of EKF are assumed as follows:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 10^{-3} & 0 \\ 0 & 10^{-3} \end{bmatrix}$$

$$Q = \begin{bmatrix} \xi & 0 & 0 & 0 & 0 \\ 0 & \xi & 0 & 0 & 0 \\ 0 & 0 & \xi & 0 & 0 \\ 0 & 0 & 0 & \xi & 0 \\ 0 & 0 & 0 & 0 & \delta \end{bmatrix} \quad G = \begin{bmatrix} \lambda & 0 & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 & 0 \\ 0 & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & \mu \end{bmatrix}$$

Traditionally, the values of ξ, δ, λ and μ are determined using a trial-and-error process which is time consuming and may not yield the best filter performance. Figure 9.4 shows the performance of the EKF speed estimation algorithm when applied to the closed-loop constant V/Hz controller, with ξ, δ, λ and μ set at different values. The example demonstrates that the EKF speed estimation algorithm is sensitive to the noise covariance matrix Q and the noise-weight matrix G.

The 'goodness' of speed estimation of the EKF with various compositions of Q and G may be evaluated by the mean squared error between the actual rotor speed and the estimated speed.

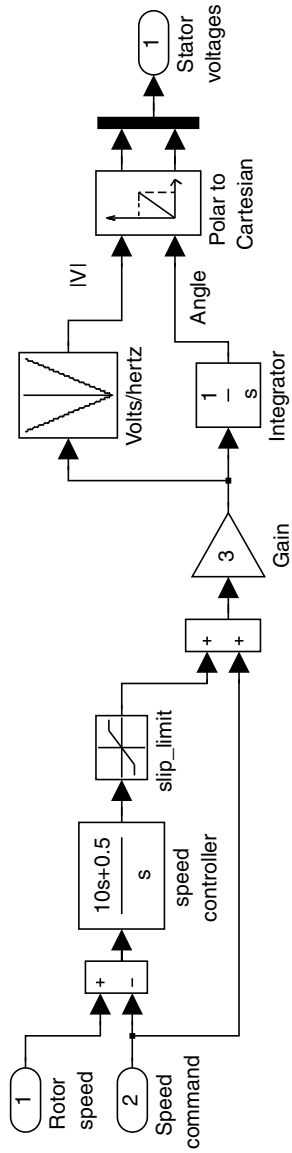


Figure 9.3 Simulink program of closed-loop V/Hz controller.

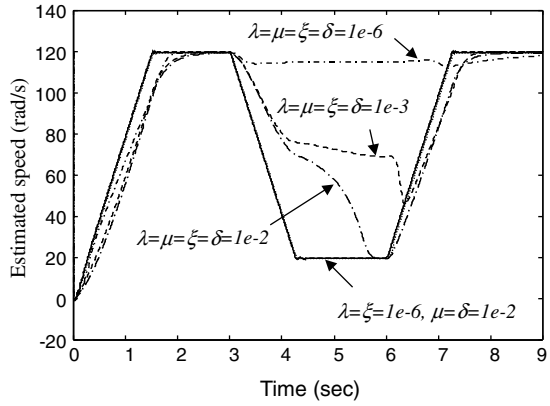


Figure 9.4 Estimated speed of constant V/Hz induction motor drive with the various matrices. Simulink model of Extended Kalman filter speed estimator. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

Using this criterion, it is found that very good performance of the EKF is obtained with the matrices $Q = Diag[10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 10^{-2}]$ and $G = Diag[10^{-6}, 10^{-6}, 10^{-6}, 10^{-6}, 10^{-2}]$, as illustrated in Table 9.1. Fine tuning of the elements of G, however, is tedious and it is difficult to check whether the optimum values have been obtained.

Table 9.1 Performance of EKF for a constant V/Hz induction motor drive in terms of the mean squared error between the actual rotor speed and the estimated speed.

Matrices G and Q	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$	Estimation Results
$\lambda = \mu = \xi = \delta =$ 1e-2	417.1220	Poor
$\lambda = \mu = \xi = \delta =$ 1e-3	772.4852	Poor
$\lambda = \mu = \xi = \delta =$ 1e-6	2.5927e + 003	Poor
$\lambda = \xi = \delta =$ 1e-3, $\mu =$ 1e-2	1.5331	Good
$\lambda = \xi =$ 1e-3, $\mu = \delta =$ 1e-2	1.0164	Good
$\lambda = \xi =$ 1e-6, $\mu = \delta =$ 1e-2	0.9985	Very good

s: actual rotor speed; e: estimated speed; n: number of data samples (= 45 000); E: mean squared error of estimated speed.
(Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

9.5 Optimizing the Noise Matrices of EKF Using GA

In order to find the best matrices G, Q, and R for the EKF, a real-coded GA is employed. The real-coded GA has many advantages (Wright, 1991) in numerical function optimization over

binary encoding. Efficiency of the real-coded GA is increased as there is no need to convert chromosomes to phenotypes before each fitness evaluation; less memory is required; there is no loss in precision by the conversion between binary and real values. The procedures of the real-coded are outlined as follows:

a. **Population Representation of the Natural Parameter**. The five diagonal elements (G_d) of the matrix G, five diagonal elements (Q_d) of covariance matrix Q, and two diagonal elements (R_d) of covariance matrix R are coded into a long real-coded string, *chromosome*. A coding example of the real-coded GA is given as follows:

$$G_d = [0.0637,\ 0.0769,\ 0.0054,\ 0.0115,\ 0.0846]$$
$$Q_d = [0.0172,\ 0.0037,\ 0.0313,\ 0.0817,\ 0.0235]$$
$$R_d = [0.0587,\ 0.0924]$$

$$\begin{aligned} chromosome &= [G_d,\ Q_d, R_d] \\ &= [0.0637,\ 0.0769,\ 0.0054,\ 0.0115,\ 0.0846,\ 0.0172, \\ &\quad 0.0037,\ 0.0313,\ 0.0817,\ 0.0235,\ 0.0587,\ 0.0924] \end{aligned}$$

b. **Initial Generation**. It begins by randomly generating an initial population of the long real-coded strings.
c. **Fitness Evaluation**. In the current generation, each of the strings is decoded back to the corresponding diagonal elements of the three matrices, G_d, Q_d, and R_d. Then, these diagonal elements from each string are separately sent to the EKF speed estimator of the induction motor drive to yield the objective function (which is the mean squared error of the estimated speed). Finally, these strings are ranked according to the value of the objective function by a linear ranking method.
d. **Reproduction**. Reproduction is a process in which parent structures are selected to form new offspring. In the present study, the stochastic universal sampling method is employed.
e. **Recombination (Crossover)**. The single-point recombination method is used to exchange the information between two chromosomes.
f. **Mutation**. Breeder Genetic Algorithm (Muhlenbein and Schlierkamp-Voosen, 1993) is used to implement the mutation operator for the real-coded GA, which uses a nonlinear term for the distribution of the range of mutation applied to gene values. This mutation algorithm is able to generate most points in the hypercube defined by the variables of the individual and range of the mutation. By biasing mutation towards smaller changes in gene values, the mutation can be used in conjunction with recombination as a foreground search process.
g. **Iteration**. The real-coded GA runs iteratively repeating the processes (c) to (g) until a population convergence condition is met or the given maximum number of iterations is reached.

The real-coded GA for EKF (GA-EKF) can be implemented on a PC by MATLAB® language. For optimizing the matrices G, Q, and R of EKF for the constant V/Hz controller, the parameters of the GA are set as follows.

1. Initial population size: 100
2. Maximum number of generations: 20
3. Probability of crossover: 0.8
4. Mutation probability: 0.01
5. Initial range of real-coded strings: [0.0001; 0.1]
6. Performance measure: the mean squared error between the actual rotor speed and the estimated speed.

Table 9.2 shows a gradient convergence process of the real-coded GA. At the twentieth generation, the mean squared error of the rotor speed and the estimated speed has decreased to 0.1543 with the optimized matrices $G = Diag([0.0020\ 0.0050\ 0.0010\ 0.0246\ 0.1000])$, $Q = Diag([0.0024\ 0.0875\ 0.0527\ 0.0001\ 0.0978])$, and $R = Diag([0.0524\ 0.0094])$. Comparing with the results of the trial-and-error method (Table 9.1), the real-coded GA method has found better matrices G, Q, and R.

Table 9.2 Iteration process of the GA.

Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$	Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$
0	8.3137		
1	5.2311	11	0.5713
2	3.8212	12	0.5247
3	2.9570	13	0.3943
4	2.3951	14	0.3306
5	1.5648	15	0.2425
6	0.9142	16	0.1794
7	0.8853	17	0.1731
8	0.7271	18	0.1618
9	0.6370	19	0.1662
10	0.6286	20	0.1543

s: actual rotor speed; e: estimated speed; n: number of data samples (= 45 000); E: mean squared error of estimated speed. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

Figure 9.5 shows the simulation results for the closed-loop constant V/Hz controller with the optimized matrices G, Q, and R.

- *Effect of current noise and machine parameter changes*

Figures 9.6 and 9.7 show the rotor speed responses of the constant V/Hz controller and the estimated speed of the GA-EKF when the rotor resistance of the induction motor, R_r, is changed

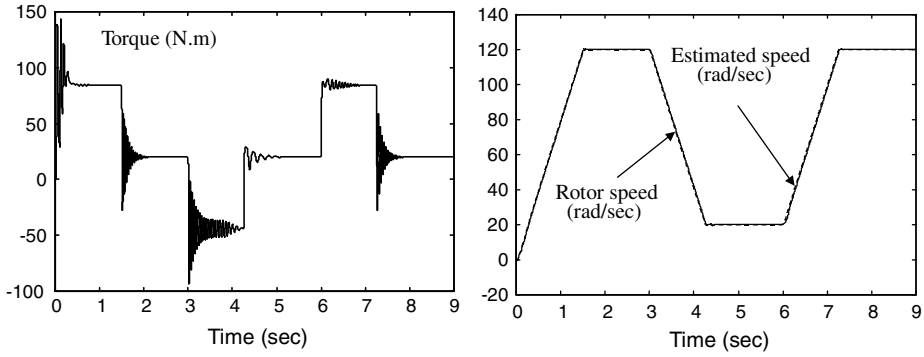


Figure 9.5 Torque, rotor speed, and estimated speed of the closed-loop constant V/Hz controller by GA-EKF.

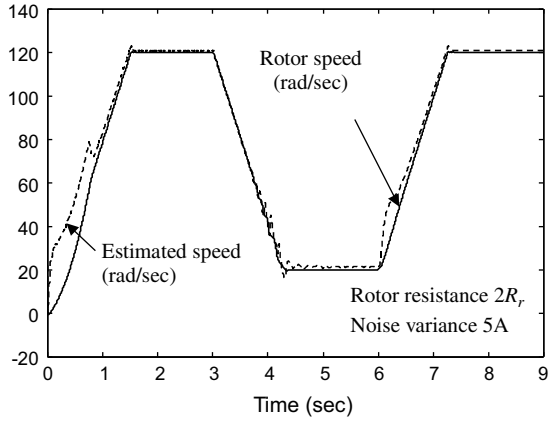


Figure 9.6 Rotor speeds and the estimated speeds of the constant V/Hz controller with the machine parameter variations and the current sensor noise.

separately to $2R_r$ and $0.8R_r$ and a random noise (variance 5 A) is present in the current sensor of the controller.

The extended Kalman filter is less sensitive to the machine parameter variations because these variations are handled as noise. Figures 9.5 and 9.6 and Table 9.3 also show that the EKF has disturbance rejection for the current sensor noise.

9.6 Speed Estimation for a Sensorless Direct Self Controller

To further investigate the performance of the optimized EKF, simulation studies on a sensorless direct self controller (DSC) (Shi, Chan and Wong, 1998) for an induction motor drive is carried out. Figure 9.8 shows the MATLAB®/Simulink program developed.

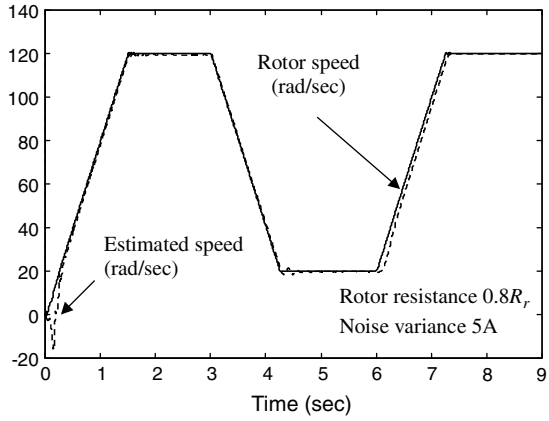


Figure 9.7 Rotor speeds and the estimated speeds of the constant V/Hz controller with the machine parameter variations and the current sensor noise.

Table 9.3 Mean squared error of rotor speed and estimated speed with the machine parameter variations and the current noise for the closed loop V/Hz controller.

Stator Resistance $R_s = 0.288\,\Omega$	Rotor Resistance $R_r = 0.161\,\Omega$	Variance of Current Noise	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$
R_s	R_r	0 A	0.1543
$0.8R_s$	$0.8R_r$	0 A	8.9731
$2R_s$	$2R_r$	0 A	25.0374
$0.8R_s$	$0.8R_r$	5 A	10.2616
$2R_s$	$2R_r$	5 A	29.7503

The induction motor, the EKF parameters, and the matrices G, Q, and R used for the simulation studies are the same as those used in the closed-loop constant V/Hz controller simulation. The speed, stator flux and torque commands are set as:

$$\omega_o{}^* = 60\,\text{rad/s} \quad 0\text{s} < t \leq 0.8\,\text{s}$$

$$\omega_o{}^* = 20\,\text{rad/s} \quad 0.8\text{s} < t \leq 1.6\,\text{s}$$

$$\omega_o{}^* = 60\,\text{rad/s} \quad 1.6\text{s} < t \leq 2.5\,\text{s}$$

$$|\lambda_s|^* = 0.86\,\text{Wb} \quad 0\text{s} < t \leq 4\,\text{s}$$

$$T^* = 200\ N\,\text{m} \quad \omega_o < \omega_o{}^*$$

$$T^* = 200\ N\,\text{m} \quad \omega_o \approx \omega_o{}^*$$

$$T^* = 20\,\text{N m} \quad \omega_o > \omega_o{}^*$$

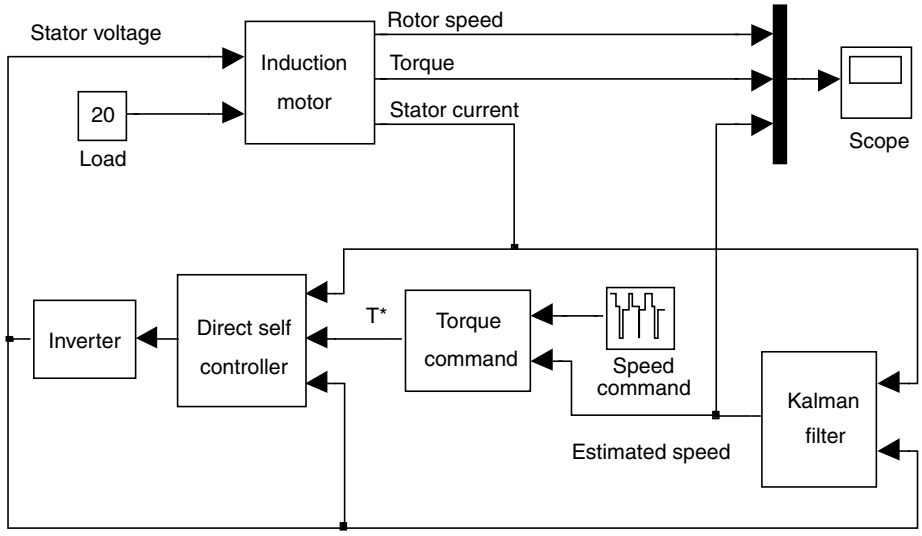


Figure 9.8 Speed estimation for a sensorless direct self controlled induction motor drive using GA-EKF.

Figure 9.9 shows the simulation results of the sensorless DSC induction motor drive using EKF with the GA-optimized matrices G, Q, and R while the mean squared error of the rotor speed and the estimated speed is 0.1528.

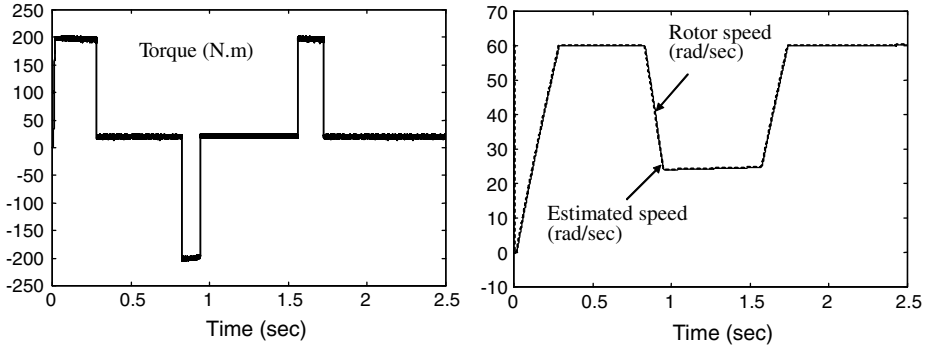


Figure 9.9 Torque, rotor speed, and estimated speed of the sensorless direct self controller.

9.7 Speed Estimation for a Field-Oriented Controller

To further explore the feasibility of the extended Kalman filter for speed estimation, simulation studies are also carried out on a field-oriented controlled (FOC) induction motor drive with direct stator flux orientation (Xu and Novotny, 1991).

Figure 9.10 shows the block diagram of the FOC system. From the stator voltages (V_{ds}, V_{qs}) and stator currents (i_{ds}, i_{qs}), the stator flux can be obtained by a 'Flux calculate' block based on Equation (2.6). Two PI blocks control the stator flux and rotor speed, while another two PI blocks control the stator currents (i_{DS}, i_{Qs}) in field coordinates. Coordinate transformation of voltages from the field frame (V_{Ds},V_{Qs}) to the stator frame (V_{ds}, V_{qs}) is implemented by an inverse Park calculation. At the same time, the 3-phase voltage signals and 3-phase current signals are sent to a GA-EKF program for rotor speed estimation. A voltage-input model of induction motor built in Chapter 3 is used to simulate a 0.147 kW induction motor (USA Bodine Electric Company model 295).

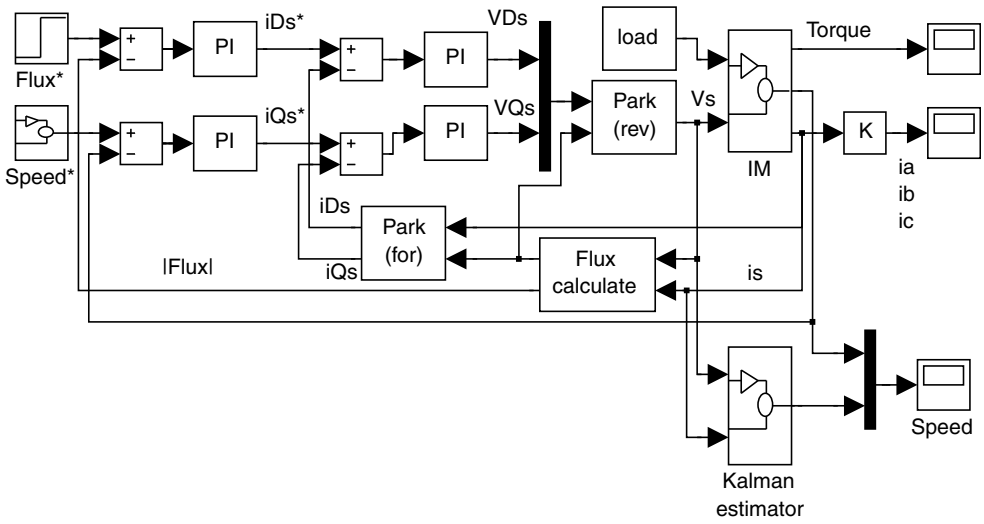


Figure 9.10 Speed estimation of FOC induction motor drive using EKF. Simulink model of Extended Kalman filter speed estimator. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

The four PI controllers (one flux controller, one speed controller, and two current controllers) in the control system are tuned individually by adjusting the proportional constant K_p and the integral constant K_I. The goal of this tuning is to choose parameters that would enable the controller to drive the error signal to zero. The four signals related to each controller (reference, actual, error and output) may be observed by 'Scope' blocks and examined during the simulation studies.

The inner current loops are tuned first with constant values on their reference inputs (i.e. the speed and flux controllers disabled). For the 0.147 kW induction motor, the parameters of the current PI controllers can be obtained from Equations (6.26)–(6.40) in Chapter 6.

$$K_p = 42$$

$$K_I = 9256$$

After the parameters of the current PI controllers have been set, the speed and flux PI controllers in Figure 9.10 are tuned by observing the responses of the 'Scope' blocks. It is found that the parameters of the speed and flux PI controllers should be tuned as follows:

$$\text{Parameters of the speed PI controller}: K_p = 100;\ K_I = 1$$

$$\text{Parameters of the flux PI controller}: \quad K_p = 2; \quad K_I = 1$$

The parameters of the 0.147 kW induction motor for the simulation studies are listed in 'Motor 3' of Appendix B. The load torque is constant at 0.5 N m.

It is assumed that the induction motor is taken through the following control cycle:

Period	Speed Command	Flux Command
$0\,\text{s} \le t < 0.1\,\text{s}$	$\omega_o^* = 0$ rad/s	$\lambda_s^* = 0.6$ Wb
$t = 0.1\,\text{s} \sim 0.25\,\text{s}$	$\omega_o^* = 0 \sim 180$ rad/s	$\lambda_s^* = 0.6$ Wb
$t = 0.25\,\text{s} \sim 1\,\text{s}$	$\omega_o^* = 180$ rad/s	$\lambda_s^* = 0.6$ Wb
$t = 1\,\text{s} \sim 1.1\,\text{s}$	$\omega_o^* = 180 \sim 20$ rad/s	$\lambda_s^* = 0.6$ Wb
$t = 1.1\,\text{s} \sim 2\,\text{s}$	$\omega_o^* = 20$ rad/s	$\lambda_s^* = 0.6$ Wb

Figure 9.11 shows the phase-A stator voltage and current of the field-oriented control system in Figure 9.10. The torque and rotor speed responses are shown in Figure 9.12.

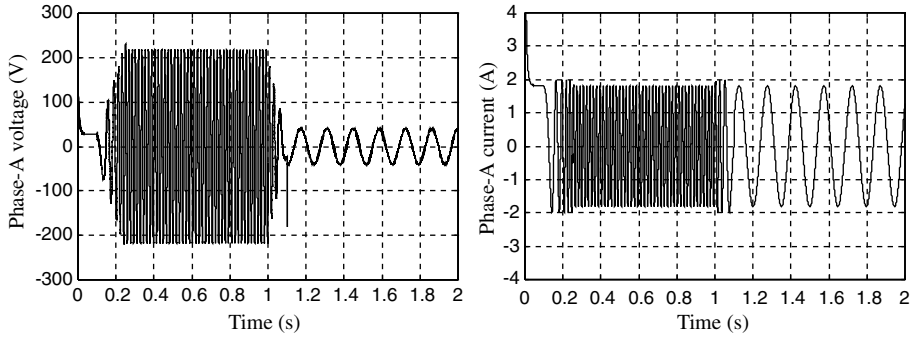


Figure 9.11 Phase-A voltage and current of the field-oriented control system.

The EKF program in Figure 9.10 is listed in Appendix F with the following 0.147 kW motor parameters used:

$$\text{Lr\_K} = 0.3481;\ \text{Ls\_K} = 0.3185;\ \text{Lh\_K} = 0.2963;\ \text{Rs\_K} = 14.6;\ \text{Rr\_K} = 12.77;$$

$$\text{H\_pole} = 4/2\ (\text{pole number P} = 4).$$

Figure 9.13 shows rotor speed estimated by the EKF, when the matrices G, Q, and R of the EKF are initially set as:

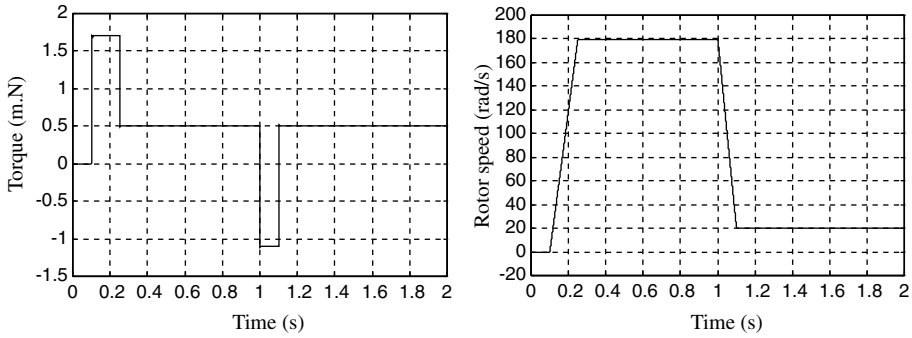


Figure 9.12 Torque and rotor speed responses of the field-oriented control system.

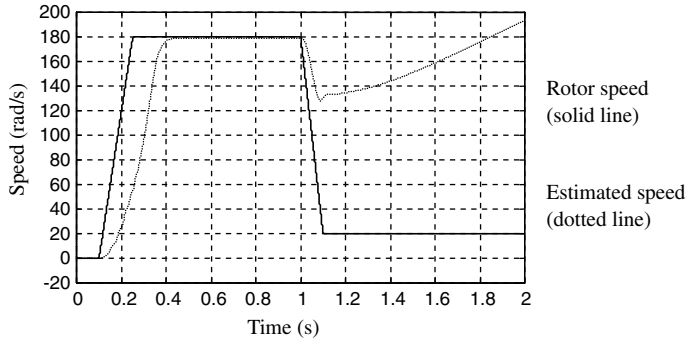


Figure 9.13 Rotor speed and estimated speed of induction motor drive with EKF.

$$G = Diag([0.01,\ 0.01,\ 0.01,\ 0.01,\ 0.5]);$$

$$Q = Diag([0.01,\ 0.01,\ 0.01,\ 0.01,\ 0.5]);$$

$$R = Diag([0.01,\ 0.01]);$$

With these selected matrices, the EKF fails to give a good speed estimation and divergence occurs eventually.

Figure 9.14 shows the rotor speed estimated by the EKF when the matrices G, Q, and R of the EKF are tuned, using a trial-and-error method, as follows:

$$G = Diag([0.01,\ 0.01,\ 0.01,\ 0.01,\ 1.5]);$$

$$Q = Diag([0.01,\ 0.01,\ 0.01,\ 0.01,\ 1.5]);$$

$$R = Diag([0.01,\ 0.01]);$$

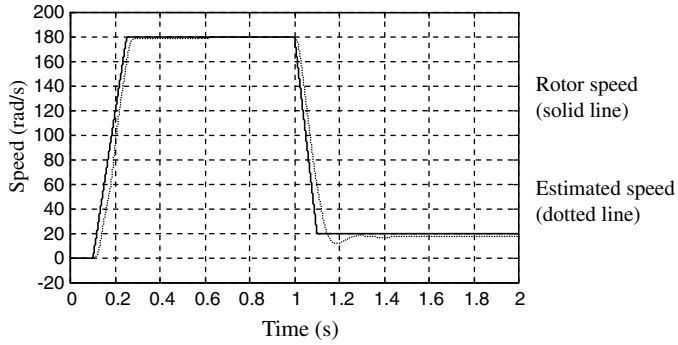


Figure 9.14 Rotor speed and estimated speed of induction motor drive with EKF.

In this case the estimation error (i.e. mean squared error between the estimated speed and the actual rotor speed) is 106.5209. The trial-and-error method has improved the speed estimation accuracy but fails to yield the best EKF performance.

In order to obtain the optimum matrices G, Q, and R for the EKF, the real-coded GA employed in Sections 9.5 and 9.6 is used. The parameters of the GA for optimizing the matrices G, Q, and R of EKF from the experimental data are the same as those in Section 9.5 except that the initial range of real-coded strings is set as [0.01; 5]. A different initial range is necessary in order to guarantee convergence and short iteration time. Table 9.4 shows the convergence process. Figure 9.15 shows the actual speed and the speed estimated using GA-EKF with the optimized matrices. The GA optimization method has improved the EKF performance by decreasing the mean squared error of estimated speed from 106.5209 (Figure 9.14) to 0.1105 (Figure 9.15). The optimized matrices are: $G = Diag$[0.0525, 0.0602, 0.0319, 0.01302, 1.809], $Q = Diag$([0.108, 0.081, 0.0716, 0.1308, 2.307]), and $R = Diag$([0.0101, 0.0104]).

Table 9.4 Iteration process of the GA.

Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$	Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$
0	42.3431		
1	23.6518	11	0.4632
2	12.7427	12	0.4728
3	7.1624	13	0.3572
4	3.4261	14	0.3147
5	1.6421	15	0.2143
6	0.8934	16	0.2315
7	0.9518	17	0.1623
8	0.6615	18	0.1372
9	0.5933	19	0.1252
10	0.4821	20	0.1105

s: actual rotor speed; e: estimated speed; n: number of data samples (= 40 000); E: mean squared error of estimated speed.

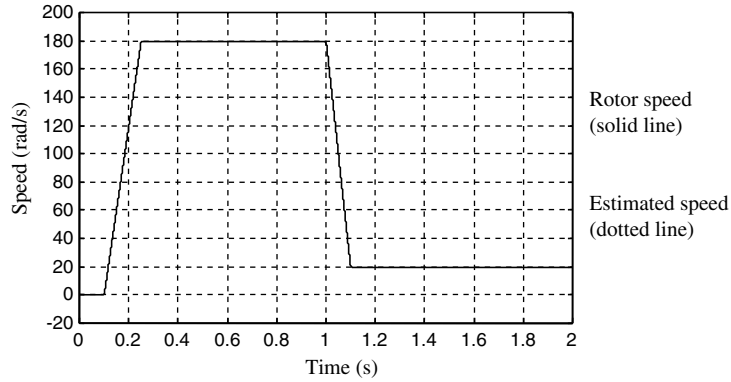


Figure 9.15 Rotor speed and estimated speed using GA-EKF with optimized matrices.

9.8 MATLAB®/Simulink Programming Examples

Four programming examples are given to illustrate the development of a voltage-frequency controlled (VFC) drive, a GA-optimized EKF for speed estimation, an EKF-based sensorless VFC of induction motor, and an EKF-based sensorless field-oriented control (FOC) of induction motor.

9.8.1 Programming Example 1: Voltage-Frequency Controlled (VFC) Drive

A voltage-frequency controlled drive is based on constant volts/Hz operation to maintain constant torque in the induction motor. A typical constant volts/Hz characteristic (The MathWorks, Inc., 2008) is shown in Figure 9.16. The straight line has a small voltage boost in order to compensate for resistance drop at low frequencies.

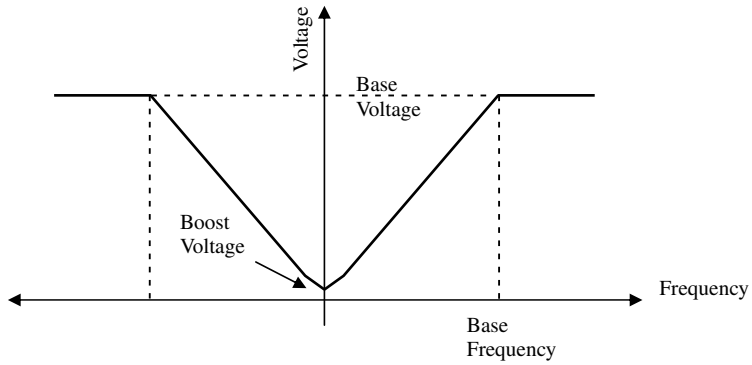


Figure 9.16 Volts/Hz characteristic.

Step 1 Building a Lookup Table of Constant Volts/Hz Characteristic
The volts/Hz characteristic of a 7.5 kW induction motor is assumed as follows:

Base frequency Fb = 60 Hz ($2\pi \times 60$ Hz = 377 rad/s)
Base voltage Vb = 300 V
Slope rate = Vb/Fb = 300/377 = 0.7958
Boost voltage Vboost = 4 V

The following MATLAB® program is first written to generate the lookup table of constant volts/Hz characteristic:

```
% Program 'Give_VF_table.m'
Base_F=60*pi*2;
Slop=300/377;
F_input=[-1*Base_F*2:4:Base_F*2];
B3=[8*Slop:Slop*4: 300];
B2=fliplr(B3);
[sF1 sF2]=size(F_input);
[sB1 sB2]=size(B3);
M= (sF2-1)/2-sB2-1;
B1=ones(1,M)*300;
V_output=[B1,B2,5,4,5,B3,B1];
plot(F_input, V_output);
axis([-800 800 -20 350]);
```

The result of running the above program is shown in Figure 9.17.

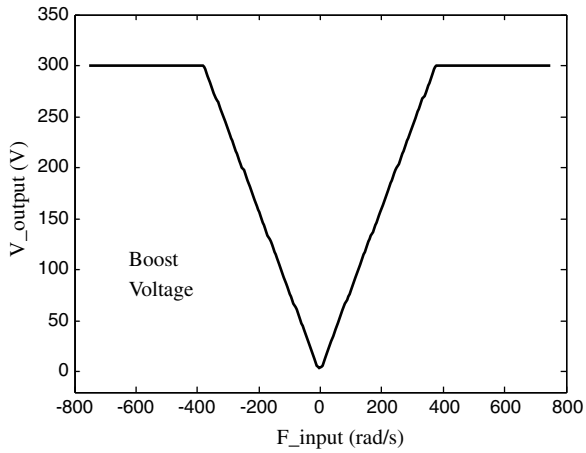


Figure 9.17 Lookup table of constant volts/Hz characteristic for the 7.5-kW motor.

The two arrays 'F_input' and 'V_output' are stored in MATLAB® workspace after the program 'Give_VF_table.m' has been run. The array 'F_input' contains the values of frequency and the 'V_output' contains the corresponding magnitudes of output voltage.

Constant volts/Hz operation can then be simulated by using a 'Lookup' block in Simulink library, as detailed in the next step.

Step 2 Building a Simulink Model of the Constant Volts/Hz Controller
The Simulink model of the constant volts/Hz controller is built as shown in Figure 9.18.

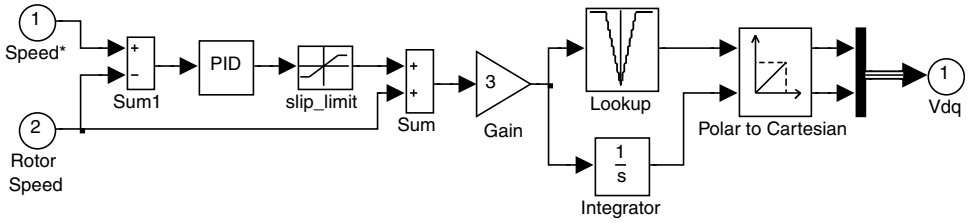


Figure 9.18 Simulink model of constant volts/Hz controller.

The Simulink model consists of a 'PID' block, a 'slip_limit' block, a 'Gain' block, a 'Lookup' block, an 'Integrator' block, and a 'Polar to Cartesian' block. The inputs are the speed command and the rotor speed, while the output is the stator voltage vector (V_{ds}, V_{qs}).

The parameters of the 'slip_limit' block are set as follows:

Upper limit: 37.7
Lower limit: −37.7

The parameters of the 'PID' block are set as follows:

Proportional: 3
Integral: 0.02
Derivative: 0

Since the induction motor has 3 pole-pairs, the parameter of the 'Gain' block is set as follows:

Gain: 3

The 'Lookup' block has two parameters set as follows:

Vector of input values: F_input
Table data: V_output

Input of the 'Lookup' block is the array 'F_input' and the output is the array 'V_output'. With the two arrays in workspace (created in **Step 1**), the 'Lookup' table can be used to simulate the constant volts/Hz characteristic.

The 'Integrator' block is used to calculate the angle of the stator voltage vector. Using the magnitude from the 'Lookup' block and the angle from the 'Integrator' block, the 'Polar to Cartesian' block outputs the stator voltage vector (V_{ds}, V_{qs}).

Step 3 Building a Simulink Model of the Voltage-Frequency Controlled Drive
The Simulink model of voltage-frequency controlled drive is built as shown in Figure 9.19.

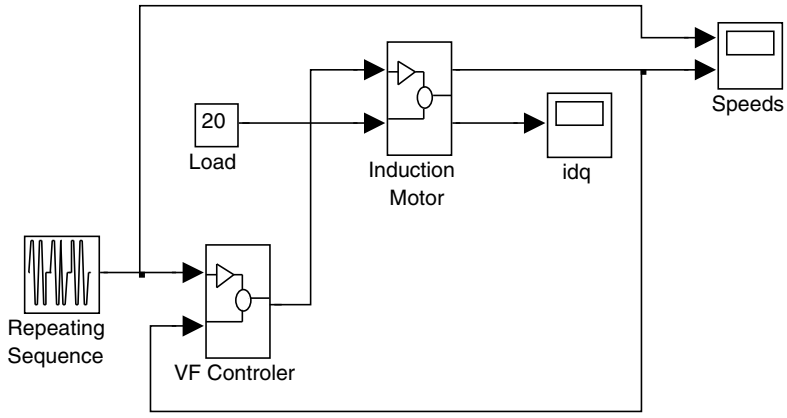


Figure 9.19 Simulink model of voltage-frequency controlled (VFC) drive.

In Figure 9.19, the 'VF Controller' block is the Simulink model of constant volts/Hz controller built in **Step 2**. The 'Induction Motor' block is a 7.5 kW voltage-input model of induction motor described in Section 3.4 with parameters of 'Motor 1' listed in Appendix B.

The 'Repeating Sequence' block outputs the speed commands with following parameters:

Time values = [0 1 2 4 6 8 10 12 14 16 17 20]
Output values = [0 0 100 100 −100 −100 100 100 −100 −100 0 0]

The 'Load' block outputs a constant load torque of 20 N m and the parameter of the block is set as follows:

Constant = 20

Step 4 Simulating the Voltage-Frequency Controlled Drive
Before performing the simulation of the voltage-frequency controlled drive in Figure 9.19, the MATLAB® program 'Give_VF_table.m' in **Step 1** should first be run to create the two arrays 'F_input' and 'V_output' for use by the 'Lookup' block in the constant volts/Hz controller. The Simulink model is run with following parameters.

Simulation type: Variable-step
Max-step size = 0.001 s
Simulation time = 18 s

The simulation results are shown in Figure 9.20.

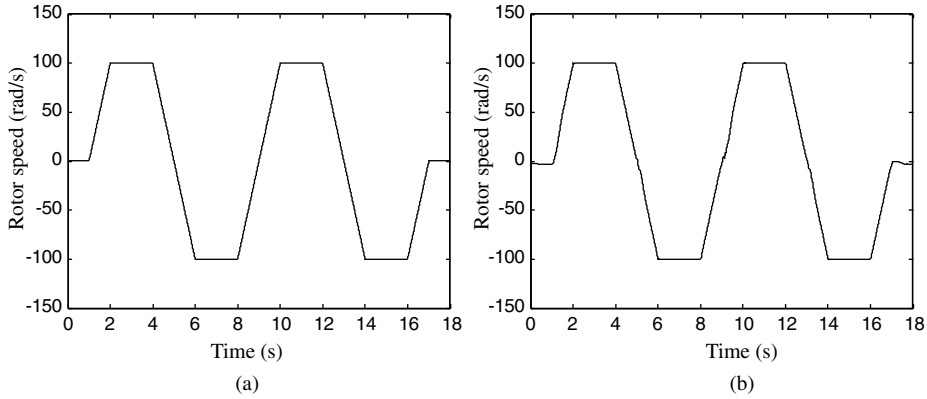


Figure 9.20 (a) Speed command and (b) Rotor speed.

9.8.2 *Programming Example 2: GA-Optimized EKF for Speed Estimation*

This example describes how the matrices $[G_d, Q_d, R_d]$ described in Section 9.5 of an extended Kalman filter (EKF) can be optimized for speed estimation of an induction motor. The MATLAB® program consists of a GA program 'Start_GA.m', a fitness evaluation program 'Call_kal.m', and a Simulink model 'kal_train.mdl' of an induction motor drive with EKF speed estimation. Their relationship is shown in Figure 9.21.

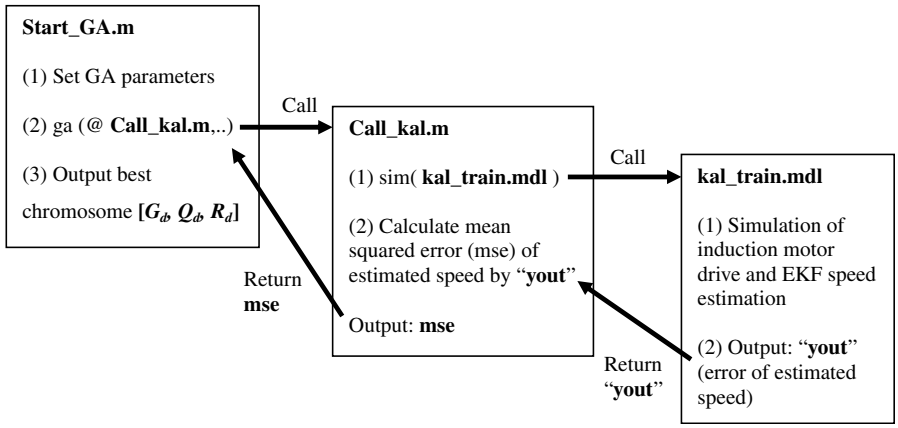


Figure 9.21 Relationship between programs in GA-optimized EKF.

The functions of programs in the GA-optimized EKF are listed in Table 9.5.

Table 9.5 Functions of programs in GA-optimized EKF.

Name of Program	Function	Input	Output
Start_GA.m	Set GA parameters and perform 'ga' function in MATLAB® which calls the fitness evaluation function 'Call_kal.m'	N/A	best chromosome matrix [G_d, Q_d, R_d]
Call_kal.m	Fitness evaluation which calls the Simulink model 'kal_train.mdl' and calculates the mean squared error (mse).	Chromosome [G_d, Q_d, R_d]	'mse' mean squared error of estimated speed as value of evaluation fitness
Kal_train.mdl	Simulink models of induction motor drive and EKF speed estimation	Chromosome [G_d, Q_d, R_d]	'yout' error of estimated speed

Notes: The chromosome matrix [G_d, Q_d, R_d] is described in Section 9.5.

Step 1 Programming GA Program 'Start_GA.m'
The following MATLAB® program 'Start_GA.m' is first written:

```
function x1=Start_GA()
clc
clear all
warning off all
%Give_VF_table_GA;
X0=rand(1,12);
lb=ones(1,12)*0.00001;
lb=lb';
ub=ones(1,12)*100;
ub=ub';
options = gaoptimset(@ga);
options = gaoptimset(options,'MutationFcn',@mutationadaptfeasible);
options = gaoptimset(options,'PopulationSize',100);
options = gaoptimset(options,'Generations',20);
options = gaoptimset(options, 'EliteCount',2);
options = gaoptimset(options, 'CrossoverFraction',0.8);
options = gaoptimset(options,'PlotFcns',{@gaplotbestf },'Display',
          'iter');
[xx,fval,exitflag]=ga(@Call_kal,12,[],[],[],[],lb, ub,[], options)
assignin('base','fval',fval);
x1=xx;
assignin('base','x1',x1);
[A,B]=size(x1);
Error=sum(x1.^2)/A;
assignin('base','Error',Error);
G=diag(x1(1:5));
Q=diag(x1(6:10));
R=diag(x1(11:12));
assignin('base','G',G);
assignin('base','Q',Q);
assignin('base','R',R);
end
```

The programming information is described in **Step 3** of Section 4.5.2.

Step 2 Programming Fitness Evaluation 'Call_kal.m'
The following program 'Call_kal.m' is next written:

```
function mse=Call_kal(xx)
x1=xx;
G=diag(x1(1:5));
Q=diag(x1(6:10));
R=diag(x1(11:12));
assignin('base','G',G);
assignin('base','Q',Q);
assignin('base','R',R);
[tout,xout,yout]=sim('kal_train',1.5);
[A,B]=size(yout);
y=yout;
y(1)=[];
mse=sum(y.^2)/A;
end
```

Step 3 Programming Simulink Model 'kal_train.mdl' of Induction Motor Drive and EKF Speed Estimation
Direct-on-line starting of a 7.5 kW induction motor is employed for the fitness evaluation of GA optimization. The Simulink model 'kal_train.mdl' of induction motor drive and EKF speed estimation is shown in Figure 9.22.

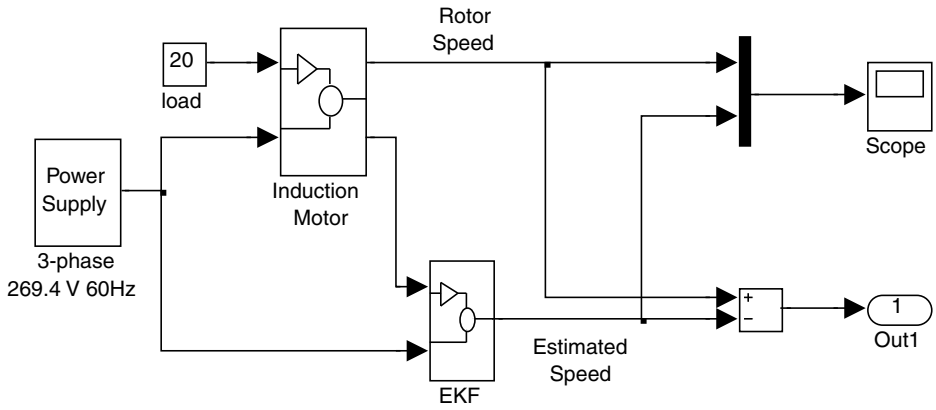


Figure 9.22 Simulink model for fitness evaluation of GA optimization.

The Simulink model 'kal_train.mdl' is called by the program 'Call_kal.m' in **Step 2** and returns error of EKF speed estimation to 'Call_kal.m' for calculating the fitness evaluation (mean squared error of estimated speed). The Simulink model consists of an 'Induction Motor'

block, an 'EKF' block, and a 'Power Supply' block. The 'Induction Motor' block may be a 7.5 kW voltage-input model (described in Section 3.4 and programmed in Section 6.8.1) or a 7.5 kW discrete-state model (described in 3.5) of induction motor. The 'EKF' block is described in Figure 9.1 and its MATLAB® program is listed in Appendix F. The 'Power Supply' block is a dq-axis voltage source with output V_{ds}, V_{qs}, which is programmed in Section 6.8.1 and shown in Figure 6.39.

The parameters of the Simulink model are set as follows.

Simulation type: Variable-step
Max-step size = 0.0002 s
Simulation time = 1.5 s

Step 4 Running the GA Programs to Optimize EKF Speed Estimation
Upon entering command 'Start_GA' in the MATLAB® window, the GA optimizing process starts. The optimizing process may be time-consuming, depending on the computer speed and the following parameters in the programs.

1. the simulation time in 'sim' function of the program 'Call_kal.m';
2. 'PopulationSize'; and
3. 'Generations' in the program 'Start_GA.m'.

After the program 'Start_GA.m' has stopped, the optimum matrices $[G_d, Q_d, R_d]$ of EKF speed estimation are stored in MATLAB® workspace. With the optimum matrices $[G_d, Q_d, R_d]$, the Simulink model 'kal_train.mdl' built in **Step 3** is run, and the simulation results are shown in Figure 9.23.

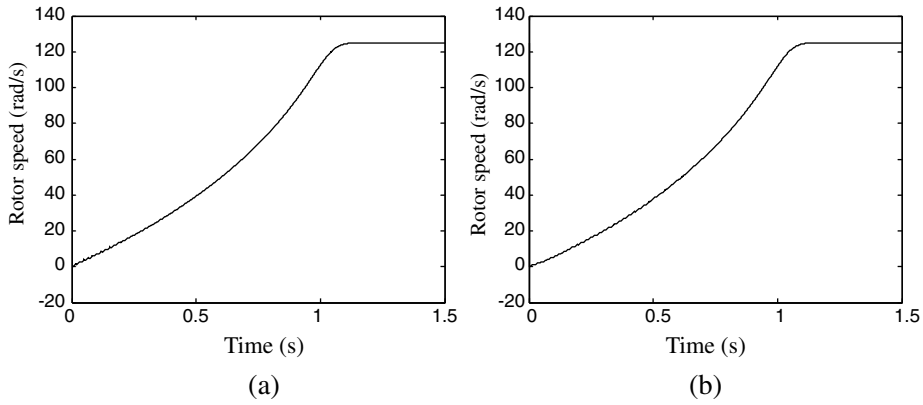


Figure 9.23 Rotor speed response of direct-on-line starting: (a) rotor speed of motor (b) estimated speed by the optimized EKF.

9.8.3 *Programming Example 3: GA-based EKF Sensorless Voltage-Frequency Controlled Drive*

The Simulink model of a GA-based EKF sensorless voltage-frequency controlled drive is shown in Figure 9.24.

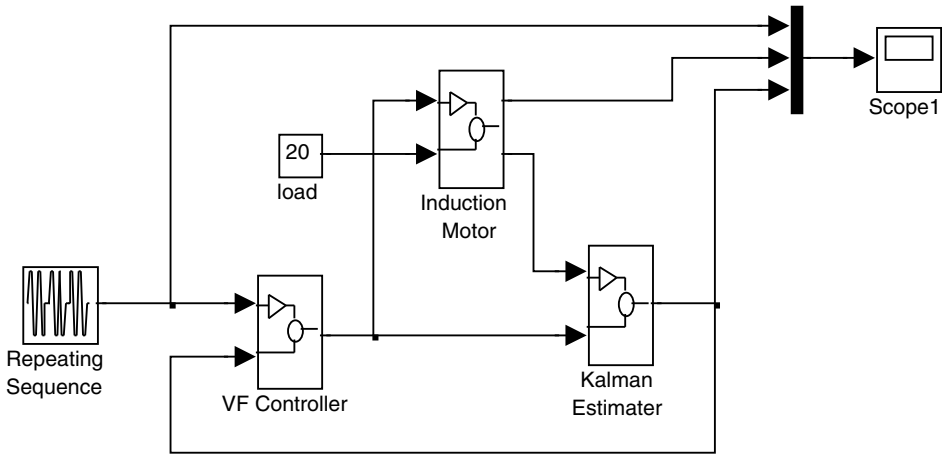


Figure 9.24 Sensorless voltage-frequency controlled drive with GA-based EKF.

The model consists of the Simulink model of voltage-frequency controlled drive built in Section 9.8.1 and an embedded optimized EKF speed estimation built in Section 9.8.2.

The parameters of the 'Repeating Sequence' block and the 'load' block in Figure 9.24 are described in **Step 3** in Section 9.8.1, and the speed response of the GA-based sensorless voltage-frequency controlled drive is shown in Figure 9.25.

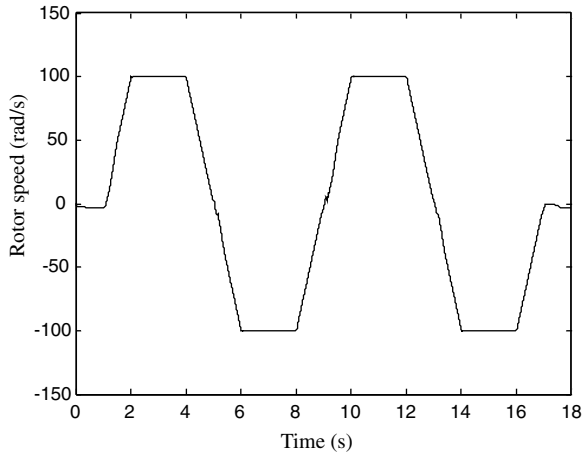


Figure 9.25 Speed response of the GA-based EKF sensorless voltage-frequency controlled drive.

9.8.4 *Programming Example 4: GA-based EKF Sensorless FOC Induction Motor Drive*

The Simulink model of a GA-based EKF sensorless FOC induction motor drive is shown in Figure 9.26.

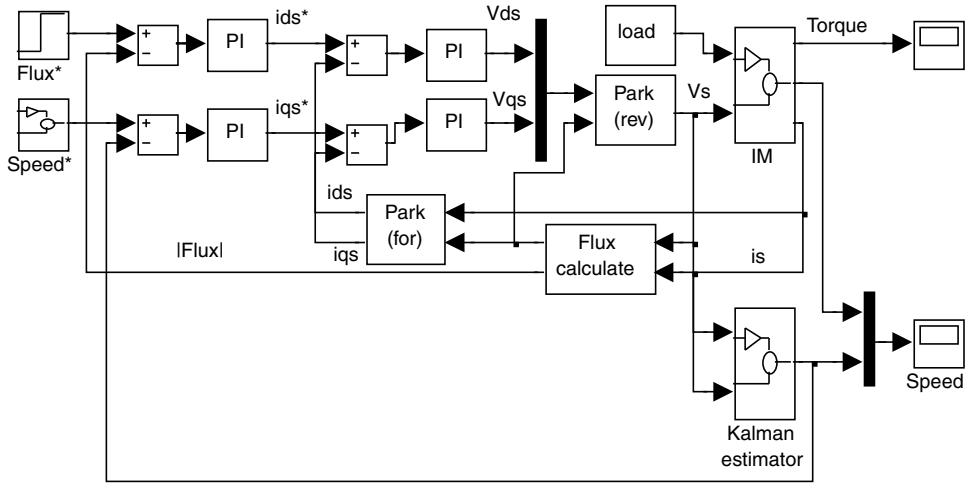


Figure 9.26 Simulink model of a sensorless FOC induction motor drive with GA-based EKF.

The sensorless FOC drive employs a 0.147 kW induction motor ('Motor 3' of Appendix B). Replacing the 7.5 kW induction motor by the 0.147 kW induction motor and repeating the GA optimizing program in Section 9.8.2, we obtain the optimized matrices $[G_d, Q_d, R_d]$ in EKF speed estimation for the 0.147 kW induction motor drive.

The Simulink model of FOC induction motor drive built in Section 9.7 and shown in Figure 9.10 is employed for constructing the sensorless FOC induction motor drive, the speed estimated by the optimized EKF being fed back to the speed PI controller, as shown in Figure 9.26.

The parameters of the four PI controllers are set as follows.

Parameters of the current i_{ds} PI controllers	$K_p = 20$	$K_I = 1$
Parameters of the current i_{qs} PI controllers	$K_p = 20$	$K_I = 1$
Parameters of the flux PI controller:	$K_p = 100$	$K_I = 1$
Parameters of the flux PI controller:	$K_p = 1$	$K_I = 100$

The speed commands are created by a 'Repeating Sequence' block with following parameters.

Time values = [0 0.1 0.25 0.25 1 1.1 2]
Output values = [0 0 180 180 180 20 20]

The 'Load' block outputs a constant load of 0.5 N m and the parameter of the block is set as follows:

Constant = 0.5

The simulated rotor speed response of the sensorless FOC induction motor drive with GA-based EKF is shown in Figure 9.27.

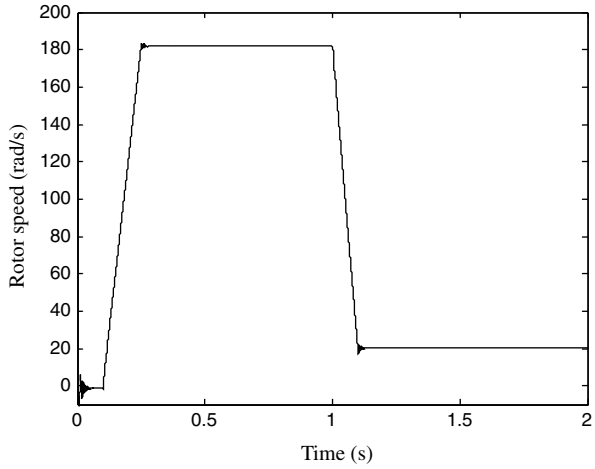


Figure 9.27 Rotor speed response of the sensorless FOC induction motor drive with GA-based EKF.

9.9 Summary

This chapter has presented a novel method to achieve good performance of an EKF for speed estimation of an induction motor drive. Based on a real-coded GA, the optimization procedure enables the noise covariance and weight matrices, on which the EKF performance critically depends, to be properly selected. Simulation studies on different induction motor drives have confirmed the efficacy of the approach.

Real-coded GA is found to be a powerful technique for optimizing the EKF algorithm as applied to the three different controllers and two different induction motors. Computer simulation results have demonstrated that the GA-EKF has good noise rejection and its performance is less sensitive to the machine parameter variations.

Possible future developments of the extended Kalman filter are (1) to find a relationship between the motor parameters and the initial range of the matrices G, Q, and R of EKF for the GA optimization, and (2) to implement the GA-EKF on a DSP board.

References

Bolognani, S., Oboe, R., and Zigliotto, M. (1999) Sensorless full-digital PMSM Drive with EKF estimation of speed and rotor position. *IEEE Transactions on Industrial Electronics*, **46**(1), 184–191.

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley Publishing Company, Reading, MA.

Henneberger, G., Brunsbach, B.J., and Klepsch, Th. (1996) Field-oriented control of synchronous and asynchronous drives without mechanical sensors using a Kalman filter, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura, and K. Matsuse), IEEE Press, pp. 207–214.

Kim, F Y.R., Sul, S.K., and Park, M.H. (1996) Speed sensorless vector control of induction motor using extended Kalman filter, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura, and K. Matsuse), IEEE Press, New Jersey, pp. 215–223.

Lewis, F.L. (1992) *Applied Optimal Control and Estimation*, Prentice-Hall, Inc., New Jersey.

Manes, C., Parasiliti, F., and Tursini, M. (1994) A comparative study of rotor flux estimation in induction motors with a nonlinear observer and the extended Kalman filter. IECON 94 Conference, pp. 2149–2154.

Muhlenbein, H. and Schlierkamp-Voosen, D. (1993) Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, **1**(1), 25–49.

Salvatore, L., Stasi, S., and Tarchioni, L. (1993) A new EKF-based algorithm for flux estimation in induction machines. *IEEE Transactions on Industrial Electronics*, **40**(5), 496–504.

Shi, K.L., Chan, T.F., and Wong, Y.K. (May (1998)) Modelling and simulation of direct self control system. IASTED International Conference: Modelling and Simulation, Pittsburgh, USA.

Shi, K.L., Chan, T.F., Wong, Y.K., and Ho, S.L.(February (2002)) Speed estimation of an induction motor drive using an optimized extended Kalman filter. *IEEE Transactions on Industrial Electronics*, **49**(1), 124–134.

Texas Instruments Incorporated (July 1997) Sensorless Control with Kalman Filter on TMS320 Fixed-Point DSP, Literature Number: BPRA057. Texas Instruments Europe.

The MathWorks, Inc. (2008) SimPowerSystems™ 5 User's Guide.

Wright, A.H. (1991) Genetic algorithms for real parameter optimization, in *Foundations of Genetic Algorithms* (ed J.E. Rawlins), Morgan Kaufmann, San Mateo, CA, pp. 205–218.

Xu, X. and Novotny, D.W. (1991) Implementation of direct stator flux orientation control on a versatile DSP based system. *IEEE Transactions on Industry Applications.*, **27**, 694–700.

10

Optimized Random PWM Strategies Based on Genetic Algorithms

10.1 Introduction

Many new pulse width modulation (PWM) techniques have been developed in order to give inverters a wider linear modulation range, lower switching loss, wider spread of energy over the harmonic spectrum, and reduced total harmonic distortion (THD) (Zhou and Wang, 2002). These new techniques may be categorized as (1) selected harmonic elimination method, (2) random PWM (RPWM) method, and (3) genetic algorithm (GA) optimization techniques. The selected harmonic elimination method suppresses the chosen harmonics by controlling the switching angles (Chiasson *et al.*, 2004). On the other hand, the random PWM inverter achieves very small magnitude of switching harmonics compared with a standard PWM inverter by spreading the harmonic energy over a wide frequency range. Two major advantages of the RPWM over the standard PWM are (Bech *et al.*, 1999): (1) the whistling acoustic noise emitted by standard PWM is converted into less annoying broad-band noise, (2) the size of filter components required for the inverter to comply with standards for conducted electromagnetic interference (EMI) is reduced. However random PWM techniques cannot significantly improve the total harmonic distortions (THD). To overcome this problem, genetic algorithm (GA) optimization techniques have recently been proposed (Shi and Li, 2003; Ozpineci, Tolbert and Chiasson, 2004) to reduce the THD of PWM waveforms.

In this chapter, a random carrier-frequency PWM, a random pulse-position PWM, a random pulse-width PWM, and a hybrid random pulse-position and random pulse-width PWM will be optimized by genetic algorithm. Compared with standard PWM and random PWM inverters, the GA-optimized random PWM inverters have the same switching loss, but they have a wider linear modulation range, lower value of THD (and hence less copper loss), and smaller harmonic amplitudes (which require a smaller filter size). The harmonic energy of the

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

GA-optimized random PWM is spread over a wide range, and the energy the fundamental component is also enhanced. In practical applications, these advantages are obtained without any extra hardware cost and programming complexity. The optimized random PWM strategy may be implemented by adding only arrays including the optimized carrier series or pulse series with some instructions included in the DSP program. A single-phase inverter is employed for this optimization study. The validity of the proposed methods has been verified by simulation and experimental studies on a DSP-based voltage-controlled inverter. As steady-state performance is being considered, the GA-optimized random PWM inverter proposed may be used as an uninterruptable power supply (UPS) or used to drive a single-phase induction motor for low performance applications, such as pumps, fans and mixers.

10.2 PWM Performance Evaluation

In a single-phase two-arm bridge PWM inverter, a triangular-carrier wave is employed for comparison with sinusoidal modulation waves to generate signals to control the semiconductor switching devices as shown in Figure 10.1.

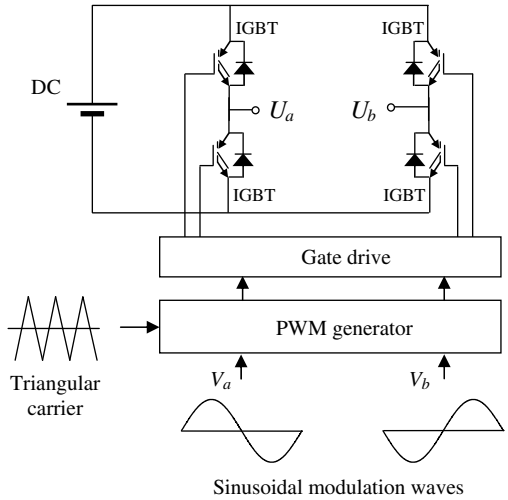


Figure 10.1 A single-phase two-arm bridge PWM inverter.

For convenience, assume that the amplitude of the triangular-carrier wave V_c shown in Figure 10.2 is 1 V. Then, V_c may be expressed mathematically as

$$\begin{aligned} V_c &= \frac{4}{T_c}t - 4n + 3 \text{ (V)} \qquad \text{when } (n-1)T_c < t < nT_c - \frac{T_c}{2} \\ V_c &= -\frac{4}{T_c}t + 4n - 1 \text{ (V)} \quad \text{when } nT_c - \frac{T_c}{2} < t < nT_c \end{aligned} \tag{10.1}$$

where $n = 1, 2, \ldots, N$, and T_c is the carrier period.

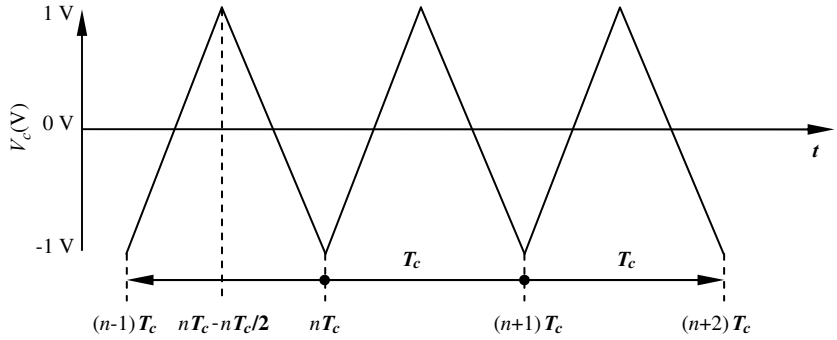


Figure 10.2 Triangular carrier wave.

The sinusoidal modulation waves in the single-phase PWM inverter are known as reference signals or modulating signals, which may be expressed as

$$\begin{aligned} V_a &= V \sin(\omega t) \\ V_b &= -V \sin(\omega t) \end{aligned} \tag{10.2}$$

where ω is the frequency and V is the amplitude of the sinusoidal waveform.

For a carrier frequency of 1 kHz, a reference sinusoidal frequency of 50 Hz, and a modulation index of 0.8, the carrier wave and the modulating waves are as shown in Figure 10.3. The points of intersection between the carrier and modulating waves generate the control signals for the PWM inverter.

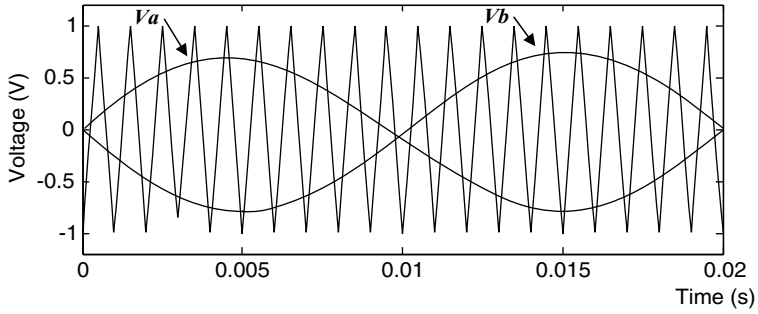


Figure 10.3 Two sinusoidal waves and triangular carrier wave.

When the value of the reference sinusoidal wave is larger than value of the triangular-carrier wave, the PWM output is in the high state; otherwise it is in the low state. In this chapter, the single-phase triangular-carrier PWM is referred to as the standard PWM.

For a DC source of 1 V, the outputs of the PWM inverter are U_a, U_b, and, U_{ab}, as shown in Figure 10.4.

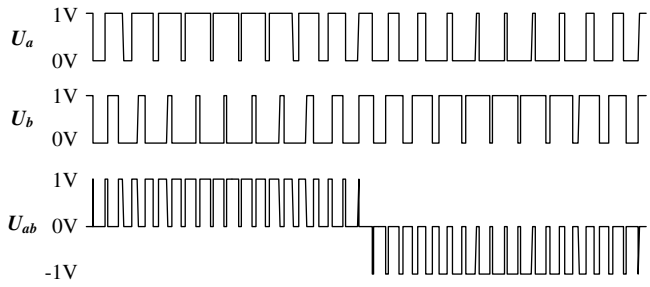


Figure 10.4 Output waveforms of PWM inverter (The pulse frequency of U_{ab} is about two times of U_a and U_b).

10.2.1 Fourier Analysis of PWM Waveform

The PWM-inverter output waveform U_{ab} may be expressed by a trigonometric form of Fourier series (Lathi, 2005):

$$U_{ab}(t) = A_0 + \sum_{i=1}^{N} [A_i \cos(i\omega_0 t) + B_i \sin(i\omega_0 t)] \text{ (V)} \tag{10.3}$$

where $A_0 = \frac{1}{T_0}\int_{T_0} U_{ab}(t)dt$, $A_i = \frac{2}{T_0}\int_{T_0} U_{ab}(t)\cos n\omega_0 t dt$, $B_i = \frac{2}{T_0}\int_{T_0} U_{ab}(t)\sin n\omega_0 t dt$, ω_0 is the fundamental frequency, and $T_0 = \frac{2\pi}{\omega_0}$.

The compact trigonometric form of the Fourier series is

$$U_{ab}(t) = C_0 + \sum_{i=1}^{N} C_i \cos(i\omega_0 t + \theta_i) \text{ (V)} \tag{10.4}$$

where $C_0 = A_0$, $C_i = \sqrt{A_i^2 + B_i^2}$, and $\theta_i = \tan^{-1}\left(\frac{-B_i}{A_i}\right)$.

The PWM output waveform may be evaluated by Fourier coefficients as follows.

a. Amplitude of the fundamental wave of PWM-inverter output:

$$U_{\_F} = C_1 \quad \text{(V)} \tag{10.5}$$

Amplitude of the maximum harmonic:

$$U_{\text{max}\_H} = \max(C_{i=2,\ldots N}) \quad \text{(V)} \tag{10.6}$$

Total harmonic distortion (THD):

$$THD = \frac{1}{C_1}\sqrt{\sum_{i=2}^{N} C_i^2} \tag{10.7}$$

Power of PWM signal in time domain is defined as

$$P\_PWM = \frac{\int_0^{T_0} U(t)^2 dt}{T_0} \tag{10.8}$$

where $U(t)$ is the PWM-inverter output waveform in the time domain, T_0 is the period, and P_PWM is power of the PWM signal.

b. Power of a discrete PWM signal is defined as (Lathi, 2005)

$$P\_PWM = \frac{\sum_{i=0}^{N-1} U_i^2}{N}$$

where U_i is the ith sample of PWM output waveform in the time domain, and N is the total sample number.

According to Parseval's theorem (Lathi, 2005), the sum of the power of all discrete FFT components equals the power of the continuous signal in the time domain, that is,

$$P\_PWM = C_0^2 + \frac{1}{2}\sum_{i=1}^{N-1} C_i^2. \tag{10.9}$$

The power of the fundamental component of PWM output waveform is defined as

$$P\_F = \frac{1}{2}C_1^2. \tag{10.10}$$

The power of all the harmonic components of PWM output waveform is defined as

$$P\_H = \frac{1}{2}\sum_{i=2}^{N-1} C_i^2. \tag{10.11}$$

10.2.2 Harmonic Evaluation of Typical Waveforms

Example 10.1 Square Wave

A square wave and its Fourier series are shown in Figure 10.5.

The period of the square wave T_0 is 0.02 s and the fundamental frequency $\omega_0 = 2\pi/T_0 = 100\pi$ rad. The square wave may be evaluated as follows.

The fundamental components are calculated from Equation (10.3):

$$A_1 = \frac{2}{T_0}\left[\int_0^{0.01} 1 \times \cos\omega_0 t dt + \int_{0.01}^{0.02} (-1) \times \cos\omega_0 t dt\right] = 0\ \text{V}$$

$$B_1 = \frac{2}{T_0}\left[\int_0^{0.01} 1 \times \sin\omega_0 t dt + \int_{0.01}^{0.02} (-1) \times \sin\omega_0 t dt\right] = \frac{4}{\pi}\ \text{V}.$$

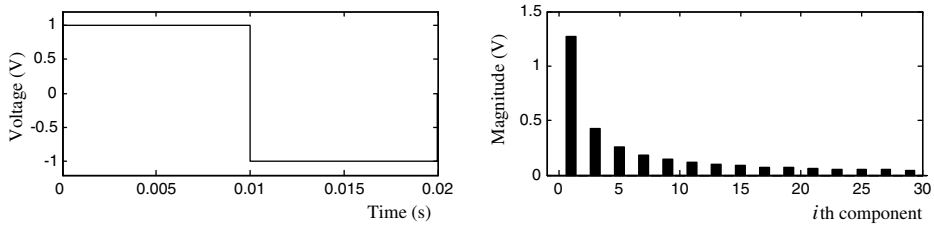


Figure 10.5 A square wave and its Fourier series.

The amplitude of fundamental component is then obtained from Equation (10.5):

$$U\_F = C_1 = \sqrt{A_1^2 + B_1^2} = \frac{4}{\pi}\ \text{V}$$

The power of signal of the square wave is obtained from Equation (10.8):

$$P\_PWM = \frac{\int_0^{T_0} U(t)^2 dt}{T_0} = \frac{1}{0.02}\left(\int_0^{0.01} 1^2 \times dt\right) + \frac{1}{0.02}\left(\int_{0.01}^{0.02} (-1)^2 \times dt\right) = 1.$$

The power of the fundamental component of PWM output is obtained from Equation (10.10):

$$P\_F = \frac{1}{2}C_1^2 = \frac{1}{2} \times \left(\frac{4}{\pi}\right)^2 = 0.811.$$

The power of all the harmonic components of PWM output is obtained from

$$P\_H = P\_PWM - P\_F = 1 - 0.811 = 0.189.$$

Because the square wave is an odd function, the harmonic components have the following properties:

1. $A_i = 0$, for all i
2. $C_i = 0$, when i is even, which implies that there are no even harmonic components in the square wave.
3. $C_i = \frac{2}{i \times \pi}$, when i is odd.

The 3rd and 5th harmonics are large in the square wave. Elimination of the harmonics of low frequency will require a large filter. PWM techniques are therefore employed in order to reduce the amplitudes of the low-frequency harmonics.

Example 10.2 Four-Pulse PWM Wave

A four-pulse waveform, U_{ab}, is produced by the triangular-carrier PWM inverter as shown in Figure 10.1 with following parameters.

Frequency of carrier wave = 100 Hz
Amplitude of carrier wave = 1 V
Frequency of reference sinusoidal wave = 50 Hz
Amplitude of reference sinusoidal wave = 0.8 V (Modulation index = 0.8)
DC source = 1 V.

Using a computer simulation program, we obtain the four-pulse PWM wave and its Fourier series as shown in Figure 10.6.

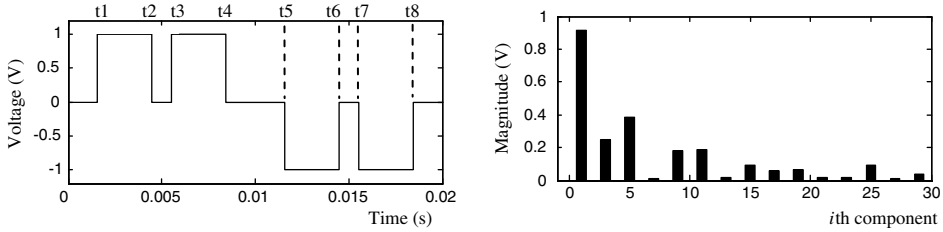


Figure 10.6 A four-pulse PWM waveform and its Fourier series.

The pulse width of the four-pulse PWM wave is 0.00291s and the pulse positions are as follows:

$$\mathrm{t\,1} = 0.00156\,\mathrm{s}, \mathrm{t\,2} = 0.00447\,\mathrm{s}, \mathrm{t\,3} = 0.00553\,\mathrm{s}, \mathrm{t\,4} = 0.00844\,\mathrm{s},$$

$$\mathrm{t\,5} = 0.01156\,\mathrm{s}, \mathrm{t\,6} = 0.01447\,\mathrm{s}, \mathrm{t\,7} = 0.01553\,\mathrm{s}, \mathrm{t\,8} = 0.01844\,\mathrm{s}.$$

The four-pulse PWM wave is an odd function and its fundamental component is calculated as follows.

The fundamental components are calculated from Equation (10.3):

$$\mathrm{A}_1 = 0;$$

$$B_1 = \frac{2}{T_0}\left[\int_{t1}^{t2} 1 \times \sin\omega_0 t dt + \int_{t3}^{t4} 1 \times \sin\omega_0 t dt + \int_{t5}^{t6} (-1) \times \sin\omega_0 t dt + \int_{t7}^{t8} (-1) \times \sin\omega_0 t dt\right] = 0.912\,\mathrm{V}$$

where $T_0 = 0.02$ s and $\omega_0 = 100\pi$ rad.

Amplitude of fundamental component is obtained from Equation (10.5):

$$U\_F = C_1 = \sqrt{A_1^2 + B_1^2} = 0.912\ \text{V}$$

Power of signal of the wave is obtained from Equation (10.8):

$$P\_PWM = \frac{\int_0^{T_0} U(t)^2 dt}{T_0} = 0.582.$$

Power of the fundamental component of the wave is obtained from Equation (10.10):

$$P\_F = \frac{1}{2} C_1^2 = \frac{1}{2} \times (0.912)^2 = 0.416.$$

Power of all the harmonic components of the wave is

$$P\_H = P\_PWM - P\_F = 0.582 - 0.416 = 0.166.$$

Because the four-pulse PWM wave is an odd function, the harmonic components have the following properties:

1. $A_i = 0$, for all i
2. $C_i = 0$ when i is an even number, which implies that there are no even harmonic components in the four-pulse PWM wave.

Comparing with the square wave in Example 10.1, the amplitude of the 3rd harmonic has been reduced in the four-pulse PWM wave.

Example 10.3 Four-Pulse Wave with Moved Positions

When the pulses in Figure 10.6 are moved to certain positions, the energy of the harmonics may be spread over a wide frequency range. The harmonic amplitudes are reduced, and the fundamental amplitude is increased as shown in Figure 10.7.

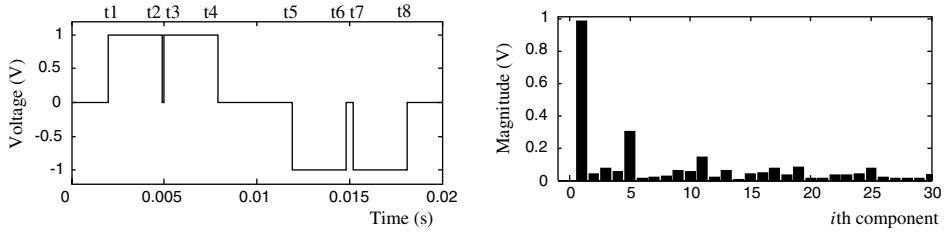


Figure 10.7 A four-pulse waveform and its Fourier series.

The pulse width of the four-pulse wave is also 0.00291s and the pulse positions are as follows:

$$\mathrm{t}\,1 = 0.00199\ \mathrm{s};\ \mathrm{t}\,2 = 0.0049\ \mathrm{s};\ \mathrm{t}\,3 = 0.005\ \mathrm{s};\ \ \mathrm{t}\,4 = 0.00791\ \mathrm{s};$$
$$\mathrm{t}\,5 = 0.0119\ \mathrm{s};\ \mathrm{t}\,6 = 0.01481\ \mathrm{s};\ \mathrm{t}\,7 = 0.0152\ \mathrm{s};\ \mathrm{t}\,8 = 0.01811\ \mathrm{s}.$$

The four-pulse wave is neither even nor odd and its fundamental component is calculated as follows.

Fundamental components are calculated from Equation (10.3):

$$A_1 = \frac{2}{T_0}\left[\int_{t1}^{t2} 1 \times \cos\omega_0 t dt + \int_{t3}^{t4} 1 \times \cos\omega_0 t dt + \int_{t5}^{t6} (-1) \times \cos\omega_0 t dt + \int_{t7}^{t8} (-1) \times \cos\omega_0 t dt\right] = 0.0073\ \mathrm{V}$$

$$B_1 = \frac{2}{T_0}\left[\int_{t1}^{t2} 1 \times \sin\omega_0 t dt + \int_{t3}^{t4} 1 \times \sin\omega_0 t dt + \int_{t5}^{t6} (-1) \times \sin\omega_0 t dt + \int_{t7}^{t8} (-1) \times \sin\omega_0 t dt\right] = 0.9879\ \mathrm{V}$$

where $T_0 = 0.02$ s and $\omega_0 = 100\pi$ rad.

The amplitude of the fundamental component is obtained from Equation (10.5):

$$U\_F = C_1 = \sqrt{A_1^2 + B_1^2} = 0.988\ \mathrm{V}.$$

The power of the wave is obtained from Equation (10.8):

$$P\_PWM = \frac{\int_0^{T_0} U(t)^2 dt}{T_0} = 0.582.$$

The four-pulse wave with moved pulse positions has the same power of signal as the PWM wave in Example 10.2 because of equal pulse widths.

The power of the fundamental component of the wave is obtained from Equation (10.10):

$$P\_F = \frac{1}{2}C_1^2 = \frac{1}{2} \times (0.988)^2 = 0.488.$$

The power of the all harmonic components of the wave is

$$P\_H = P\_PWM - P\_F = 0.582 - 0.488 = 0.094.$$

The 5th harmonic is prominent in the four-pulse wave. To reduce the lower frequency harmonics, PWM with a higher carrier frequency should be employed.

Example 10.4 Standard PWM Output with a 5-kHz Triangular Carrier Wave

The standard PWM inverter (Figure 10.1) is simulated with the following parameters:

Frequency of carrier wave = 5 kHz
Amplitude of carrier wave = 1 V
Frequency of reference sinusoidal wave = 50 Hz
Amplitude of reference sinusoidal wave = 0.8 V (Modulation index = 0.8)
DC source = 1 V.

The spectrum of the output wave U_{ab} of the standard PWM inverter is shown in Figure 10.8 and several prominent harmonics are present. As shown in Figure 10.4, the pulse frequency of U_{ab} is about two times of the triangular-carrier frequency, hence the prominent harmonic has a frequency of 10 kHz.

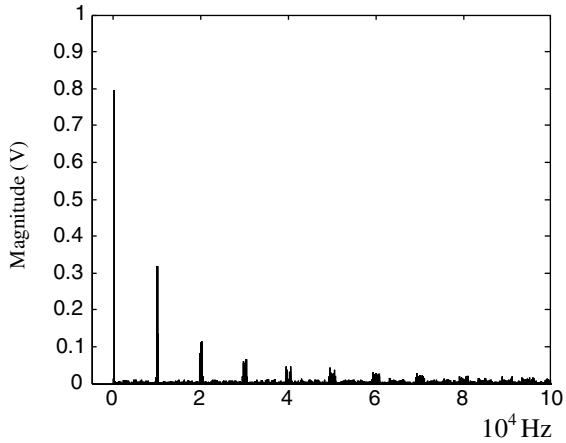


Figure 10.8 Spectrum of the standard PWM output U_{ab}.

The evaluation results for various PWM waveforms are summarized in Table 10.1.

Comparing the results for Examples 10.2 and 10.3 in Table 10.1, we may conclude that when pulse positions are changed,

1. the THD is reduced;
2. the energy of the harmonics is spread;
3. power and amplitude of the harmonic components are reduced;
4. power and amplitude of fundamental component are enhanced;

Table 10.1 Evaluation results of various PWM waveforms.

Performance of PWM	Example 10.1	Example 10.2	Example 10.3	Example 10.4
THD Equation (10.7)	0.483	0.631	0.438	0.774
Power of signal Equations (10.8) and (10.9)	1	0.582	0.582	0.508
Power of fundamental component Equation (10.10)	0.811	0.416	0.488	0.318
Power of all harmonic components Equation (10.11)	0.189	0.166	0.094	0.190
Amplitude of fundamental component Equation (10.5)	1.273 V	0.912 V	0.988 V	0.797 V
Amplitude of max harmonic component Equation (10.6)	0.424 V	0.385 V	0.301 V	0.318 V at 10 kHz

5. the power of signal (summation of the power of fundamental component and the power of all harmonic components) is unchanged.

10.3 Random PWM Methods

To improve the performance of the PWM inverter by using traditional mathematical methods, very complex calculation is involved. Hence, random PWM methods optimized by genetic algorithm (GA) are proposed.

Various random pulse-width modulation techniques for dc-ac inverters have been reported and summarized in (Trzynadlowski *et al.*, 1994). The three basic random PWM strategies are:

1. random carrier-frequency modulation (Habetler and Divan, 1991; Boys and Handley, 1992; Pedersen and Blaabjerg, 1992), in which the carrier frequencies are randomly varied;
2. random pulse-position modulation (Kirlin, Kwok and Trzynadlowski, 1993; Kirlin *et al.*, 1994), in which the switching pulses are randomly placed in individual switching intervals;
3. random switching or random pulse-width modulation (Legowski and Trzynadlowski, 1989; Legowski and Trzynadlowski, 1990; Tse *et al.*, 2000), in which the pulse width is randomly varied relative to the output-pulse width of the standard PWM.

10.3.1 Random Carrier-Frequency PWM

Random carrier-frequency modulation may be regarded as a special frequency shift keying (FSK) technique (Habetler and Divan, 1991). The instantaneous carrier half period can be

described in terms of the half switching period T_n, as follows:

$$f_i = \frac{1}{T_n}$$
$$T_n = n(t_i) \tag{10.12}$$

where n = 1, 2,. . ., N, and T_n is the time period determined by the random function $n(t_i)$.

A triangular carrier waveform with random frequency is shown in Figure 10.9.

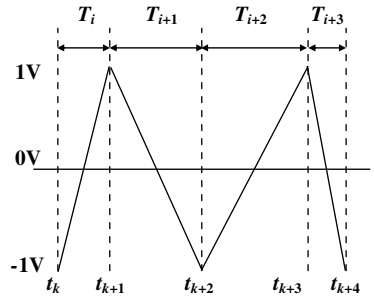


Figure 10.9 Triangular carrier waveform with random frequency.

In computer simulation, pseudo Gaussian 'white' noise is used as the random function $n(t_i)$, while in DSP-based PWM inverter implementation, the random function is generated by a pseudo-random function generator. When the DC supply is 1 V, the mean random-carrier frequency is 5 kHz and its amplitude is 1 V, frequency of sinusoidal wave is 50 Hz and amplitude is 0.8 V (modulation index equals 0.8), and FFT is based on 4096 samples, the spectrum of output U_{ab} of the random carrier-frequency PWM inverter is as shown in Figure 10.10.

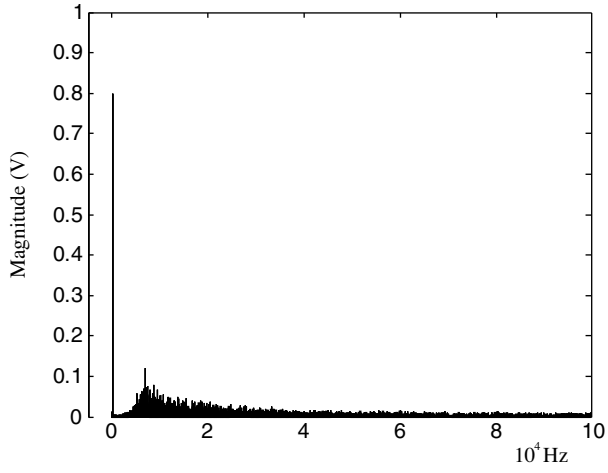


Figure 10.10 Spectrum of output U_{ab} of random carrier-frequency PWM inverter.

10.3.2 Random Pulse-Position PWM

In random pulse-position PWM, harmonic components of the output waveform are suppressed by moving the PWM positions randomly in a pulse interval, while keeping all pulse widths unchanged.

The random pulse position PWM may be expressed as follows.

$$P_{center}(n) = P_{PWM\_center}(n) \pm R(n) \tag{10.13}$$

where $n = 1, 2, \ldots, N$; $P_{\_center}$ and P_{PWM_center} are center positions of nth pulse of the random PWM and the standard PWM, and $R(n)$ the is nth random number.

An example of random pulse-position PWM waveform is shown in Figure 10.11 together with a standard PWM for comparison.

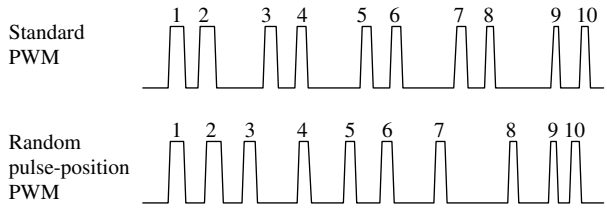


Figure 10.11 Comparison between random pulse-position PWM and standard PWM.

A random pulse-position PWM is simulated and evaluated by the following steps.

Step 1 Obtain a standard PWM pulse series by using a 5 kHz triangular-carrier wave of amplitude 1 V and a 50 Hz modulating sinusoidal waveform of amplitude as 0.8 V, giving a modulation index of 0.8.

Step 2 Move the pulse positions of the standard PWM output randomly in each half cycle.

Step 3 Calculate the FFT of random pulse-position PWM output based on 4096 samples. When DC supply is 1 V, the spectrum of output U_{ab} of the random pulse-position PWM inverter is shown in Figure 10.12.

10.3.3 Random Pulse-Width PWM

Random pulse-width PWM may be expressed as follows:

$$W_{rand}(n) = W_{PWM}(n) \pm R(n) \tag{10.14}$$

where $n = 1, 2, \ldots, N$; W_{PWM} and W_{rand} are widths of the nth pulse of standard PWM and random pulse-width PWM, and $R(n)$ is the nth random number.

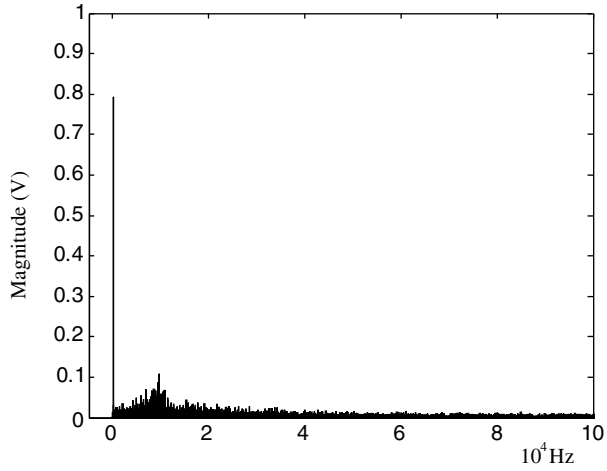


Figure 10.12 Typical spectrum of random pulse-position PWM inverter output.

An example of random pulse-width PWM is shown in Figure 10.13 together with a standard PWM for comparison.

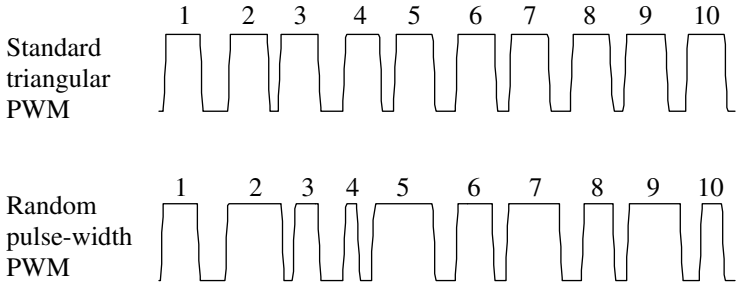


Figure 10.13 Comparison between random pulse-width PWM and standard PWM.

When DC supply is 1 V, carrier frequency is 5 kHz and amplitude is 1 V, frequency of sine wave is 50 Hz and amplitude is 0.8 V (modulation index equals 0.8), and FFT is based on 4096 samples, the spectrum the random pulse-width PWM inverter output U_{ab} is shown in Figure 10.14.

10.3.4 Hybrid Random Pulse-Position and Pulse-Width PWM

Both the pulse position and pulse width are randomly changed in a hybrid random pulse-position and pulse-width PWM.

When DC supply is 1 V, carrier frequency is 5 kHz and amplitude is 1 V, frequency of sine wave is 50 Hz and amplitude is 0.8 V (modulation index equals 0.8), and FFT is based on 4096

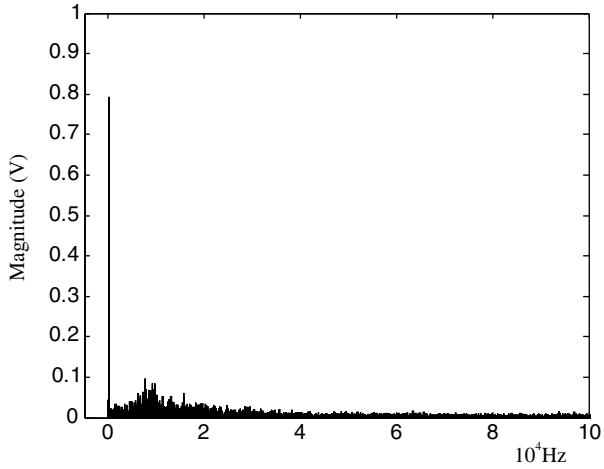


Figure 10.14 Typical spectrum of random pulse-width PWM inverter output.

samples, the spectrum of output U_{ab} of the hybrid random pulse-position and random pulse-width PWM inverter is shown in Figure 10.15.

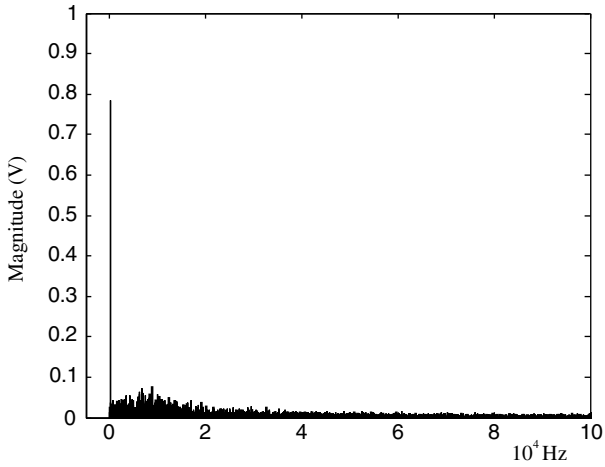


Figure 10.15 Typical spectrum of hybrid random pulse-position and random pulse-width PWM inverter output.

10.3.5 Harmonic Evaluation Results

The standard and random PWM output waveforms are generated using the following conditions:

DC supply = 1 V
Frequency of carrier wave = 5 kHz

Amplitude of carrier wave = 1 V
Frequency of sinusoidal wave = 50 Hz
Amplitude of sinusoidal wave = 0.8 V (Modulation index = 0.8)
Sample number of FFT = 4096.

The output waveforms are then evaluated by Equations (10.5)–(10.11) and the results are summarized in Table 10.2.

Table 10.2 Evaluation results of standard and random PWM output waveforms.

Evaluation results and equations	Standard PWM	Random carrier frequency PWM	Random pulse-position PWM	Random pulse-width PWM	Hybrid random pulse-position and pulse-width PWM
THD Equation (10.7)	0.7739	0.7721	0.7820	0.7840	0.7997
Signal power Equation (10.8)	0.5078	0.5093	0.5061	0.5078	0.5044
Power of fundamental Equation (10.10)	0.3176	0.3191	0.3141	0.3142	0.3076
Power of all harmonics Equation (10.11)	0.1902	0.1902	0.1920	0.1931	0.1967
Amplitude of fundamental Equation (10.5)	0.7970 V	0.7988 V	0.7925 V	0.7927 V	0.7843 V
Amplitude of max harmonic Equation (10.6)	0.3182 V	0.1219 V	0.1096 V	0.0956 V	0.0757 V

Comparing the results in Tables 10.1 and 10.2, we observe that the harmonic amplitudes of various random PWM methods are greatly reduced. However, other evaluated values of the various random PWM methods are not improved compared with the standard PWM inverter. Hence, genetic algorithm is proposed to improve the PWM output waveform.

10.4 Optimized Random PWM Based on Genetic Algorithm

Genetic algorithms have been applied to optimize electrical drive systems recently (Zhang *et al.*, 2001; Shi *et al.*, 2002; Shi and Li, 2003). In Chapter 9, the application of real-coded genetic algorithm (GA) to an extended Kalman filter based sensorless induction motor drive is described. In this chapter, the same technique will be employed for optimizing various PWM strategies. The THD of the PWM output waveform is selected as the fitness value of chromosomes, hence the fittest individual in GA optimization will have the best PWM waveform evaluation result. A flowchart for optimizing a random PWM inverter waveform using GA is shown in Figure 10.16. The simulation results for different PWM strategies are listed in Table 10.3.

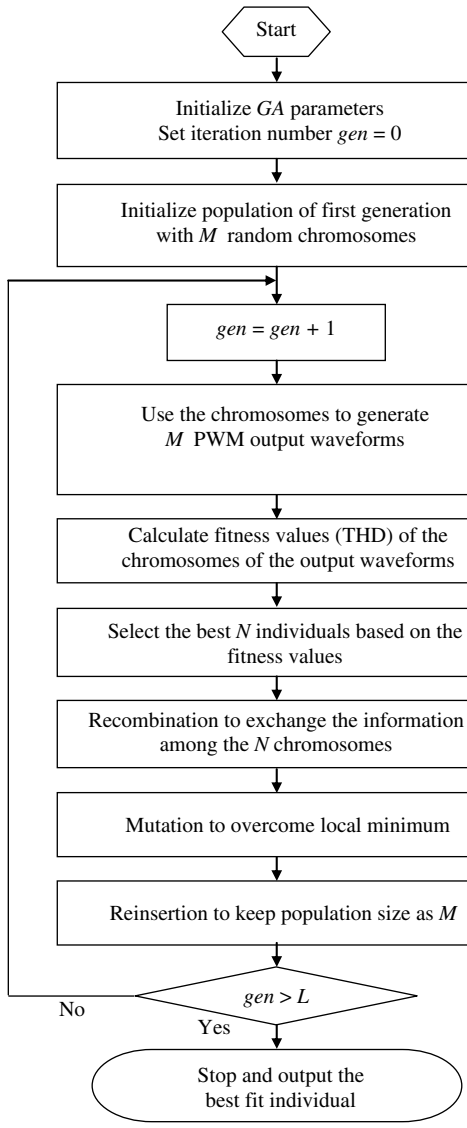


Figure 10.16 Flowchart for optimizing random PWM inverter output waveform using GA. (Reproduced by permission of K.L. Shi and H. Li, "Optimized PWM strategy based on genetic algorithms," *IEEE Transaction on Industrial Electronics*, **52**(5), 2005: 1458–1461. © 2005 IEEE.)

10.4.1 GA-Optimized Random Carrier-Frequency PWM

The GA program with following parameters is used to optimize a random carrier-frequency PWM.

1. DC supply = 1 V
2. Mean of the carrier frequencies: 5 kHz

Table 10.3 Comparison of various PWM evaluation results.

Simulation results	Calculation equations	Standard PWM	GA carrier frequency	GA pulse-position	GA pulse-width	GA pulse-position - width
THD	Equation (10.7)	0.7739	0.7478	0.6420	0.6489	0.6085
Power of signal	Equation (10.8)	0.5078	0.5152	0.5078	0.5078	0.5098
Power of fundamental	Equation (10.10)	0.3176	0.3304	0.3596	0.3573	0.3719
Power of all harmonics	Equation (10.11)	0.1902	0.1848	0.1482	0.1505	0.1377
Amplitude of fundamental	Equation (10.5)	0.7970 V	0.8129 V	0.8481 V	0.8453 V	0.8625 V
Amplitude of max harmonic	Equation (10.6)	0.3182 V	0.0877 V	0.1165 V	0.1102 V	0.1423 V

3. Amplitude of carrier wave = 1 V
4. Frequency of sinusoidal wave = 50 Hz
5. Amplitude of sinusoidal wave = 0.8 V (modulation index: 0.8)
6. Total pulse number in 0.02 s: equals total pulse number yielded by the standard PWM
7. Chromosome: time period series of triangular-carrier wave
8. Fitness value of chromosome: THD of PWM inverter output
9. Population size: 200
10. Maximum number of generations: 200
11. Sample number of FFT = 4096

By proper scaling, the total pulse number of the GA-optimized random PWM equals exactly the total pulse number yielded by the standard PWM in a fundamental cycle (0.02 s). Hence, the total switching loss in the GA-optimized random PWM is same as that in the standard PWM inverter. Computer simulation shows that the fitness value THD has decreased to 0.7478 at the 200th generation, as shown in Figure 10.17.

The spectrum of output U_{ab} of the optimized random-carrier-frequency PWM inverter is shown in Figure 10.18. The real-coded GA method has spread the harmonic energy over a wide frequency range.

Comparing with the standard PWM, the THD of the GA-optimized random-carrier-frequency PWM is reduced by about 3.4 %, the power of fundamental component is increased by 4 %, the power of all harmonic components is reduced by 2.8 %, the amplitude of the maximum harmonic component is reduced by about 72.4 %, and the amplitude of the fundamental component is increased by about 2 %.

10.4.2 GA-Optimized Random-Pulse-Position PWM

The GA program with following parameters is used to optimize a random pulse-position PWM inverter.

1. DC supply = 1 V
2. Mean of the carrier frequencies: 5 kHz
3. Amplitude of carrier wave = 1 V

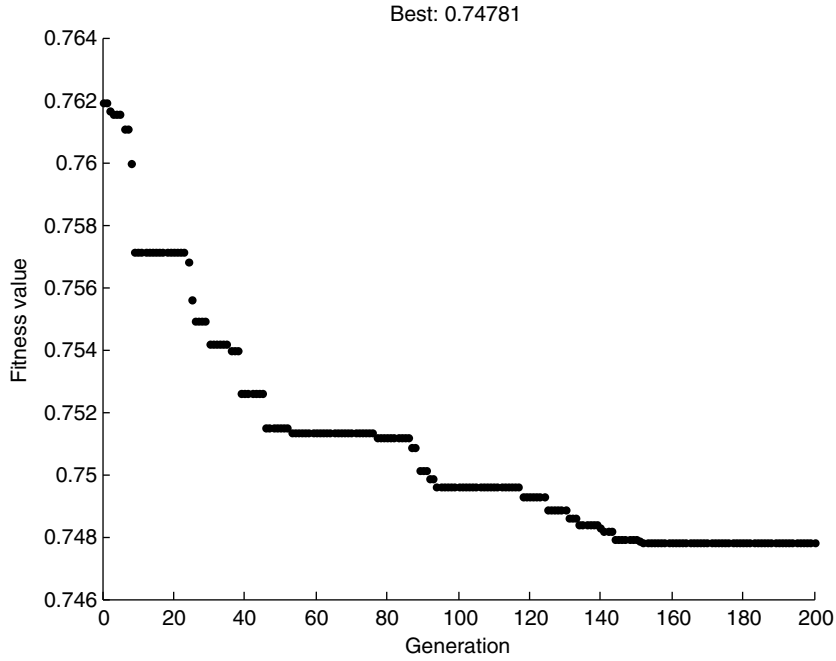


Figure 10.17 Convergence process of optimizing a random carrier-frequency PWM inverter.

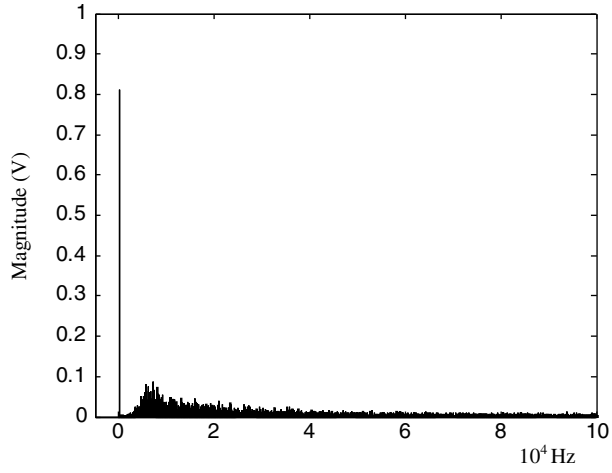


Figure 10.18 Spectrum of GA-optimized random-carrier-frequency PWM inverter output.

4. Frequency of sinusoidal wave = 50 Hz
5. Amplitude of sinusoidal wave = 0.8 V (Modulation index: 0.8)
6. Total pulse number in 0.02 s: equals total pulse number yielded by the standard PWM
7. Chromosome: time series of pulse position

8. Fitness value of chromosome: THD of PWM inverter output
9. Population size: 100
10. Maximum number of generations: 50
11. Initial population: Pulse-position series produced by the standard PWM inverter
12. Sample number of FFT = 4096

As shown in Figure 10.19, the fitness value THD has decreased to 0.6420 at the 50th generation.

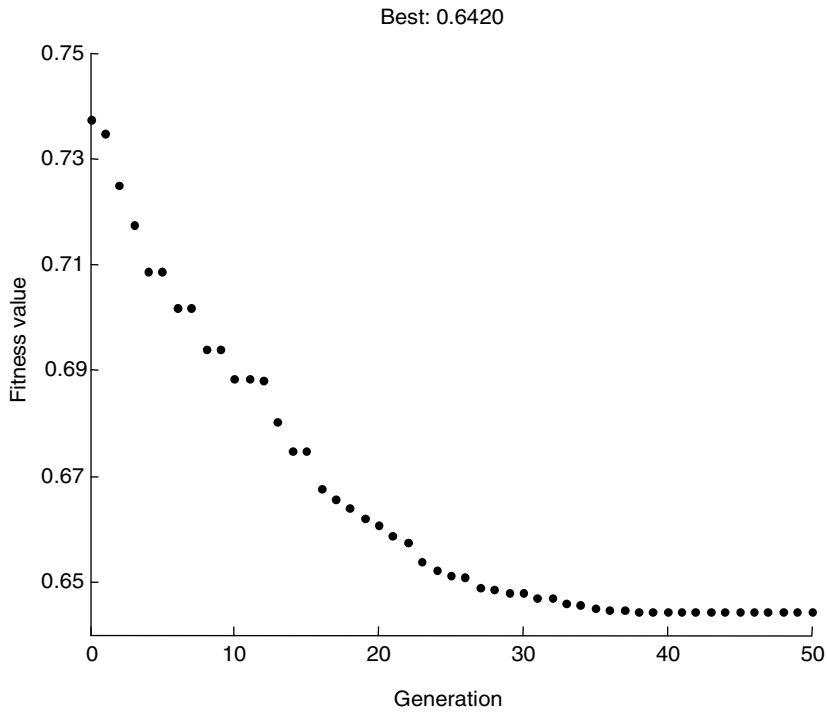


Figure 10.19 Convergence process of optimizing a random pulse-position PWM inverter.

The spectrum of output U_{ab} of the optimized random-pulse-position PWM inverter is shown in Figure 10.20.

10.4.3 GA-Optimized Random-Pulse-Width PWM

The GA program with following parameters may yield satisfactory results to optimize pulse-width PWM.

1. DC supply = 1 V
2. Mean of the carrier frequencies: 5 kHz
3. Amplitude of carrier wave = 1 V
4. Frequency of sinusoidal wave = 50 Hz

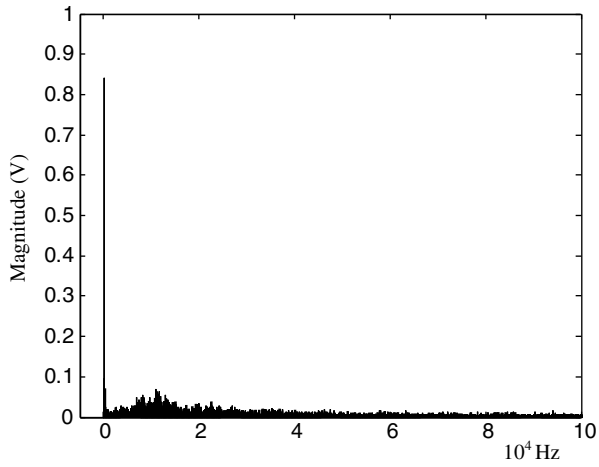


Figure 10.20 Spectrum of GA-optimized random-pulse-position PWM inverter output.

5. Amplitude of sinusoidal wave = 0.8 V (Modulation index = 0.8)
6. Total pulse number in 0.02 s: equals total pulse number yielded by the standard PWM
7. Chromosome: time series of pulse width
8. Fitness value of chromosome: THD of PWM inverter output
9. Population size: 200
10. Maximum number of generations: 50
11. Initial population: pulse-width series produced by standard PWM inverter
12. Sample number of FFT = 4096

As shown in Figure 10.21, the fitness value THD has decreased to 0.64895 at the 50th generation.

The spectrum of output U_{ab} of the optimized random-pulse-width PWM inverter is shown in Figure 10.22.

10.4.4 GA-Optimized Hybrid Random Pulse-Position and Pulse-Width PWM

The GA program with following parameters is used to optimize a hybrid random pulse-position and pulse-width PWM.

1. DC supply = 1 V
2. Mean of the carrier frequencies: 5 kHz
3. Amplitude of carrier wave = 1 V
4. Frequency of sinusoidal wave = 50 Hz
5. Amplitude of sinusoidal wave = 0.8 V (Modulation index = 0.8)
6. Total pulse number in 0.02 s: equals total pulse number yielded by the standard PWM
7. Chromosome: time series of pulse position and pulse width
8. Fitness value of chromosome: THD of PWM inverter output
9. Population size: 200

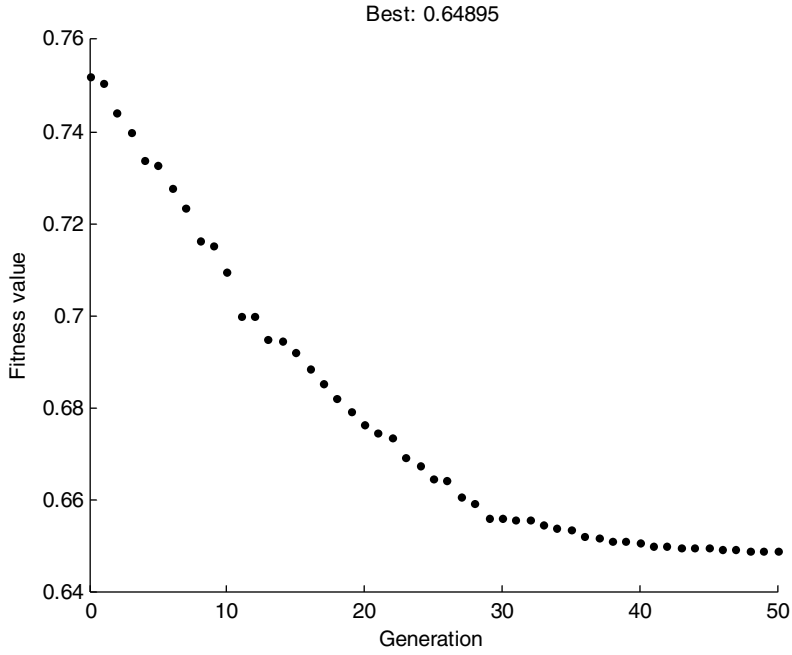


Figure 10.21 Convergence process of optimizing a random pulse-width PWM inverter.

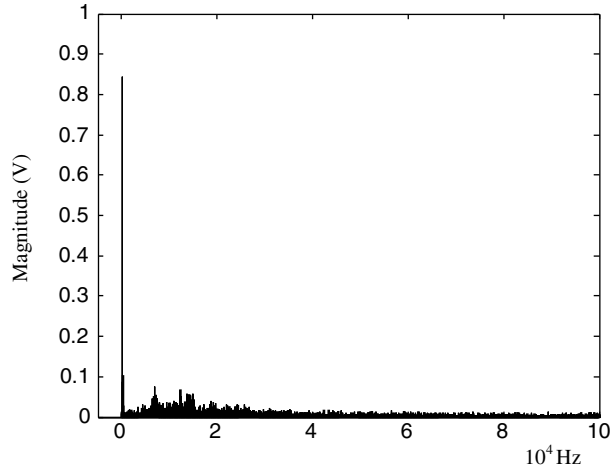


Figure 10.22 Spectrum of GA-optimized random-pulse-width PWM inverter output.

10. Maximum number of generations: 50
11. Initial population: pulse-position and pulse-width series produced by standard PWM inverter
12. Sample number of FFT = 4096

As shown in Figure 10.23, the fitness value THD has decreased to 0.60853 at the 50th generation.

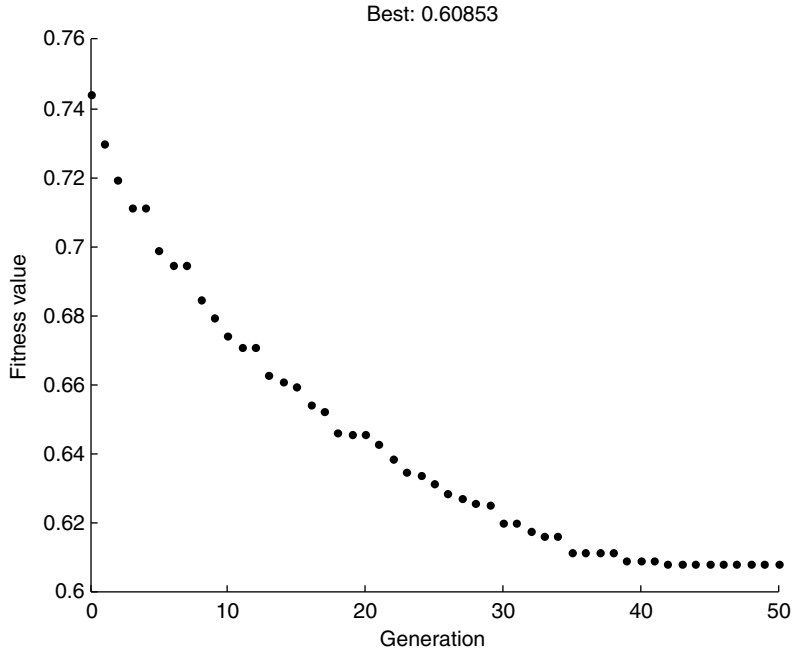


Figure 10.23 Convergence process of optimizing a hybrid random pulse-position and pulse-width PWM inverter.

The spectrum of output U_{ab} of the optimized hybrid random pulse-position and pulse-width PWM inverter is shown in Figure 10.24.

10.4.5 Evaluation of Various GA-Optimized Random PWM Inverters

Computer evaluation results of various GA-optimized random PWM methods are listed in Table 10.3 for comparison.

In order to compare various PWM methods, we define a parameter $\Delta Eval$ as follows:

$$\Delta Eval = \frac{(Evaluation\_GA - Evaluation\_S)}{Evaluation\_S} \times 100\%. \tag{10.15}$$

where *Evaluation_S* is evaluation value of the standard PWM and *Evaluation_GA* is evaluation value of the GA-optimized random PWM. The computed values of $\Delta Eval$ for different GA-optimized random inverters are summarized in Table 10.4.

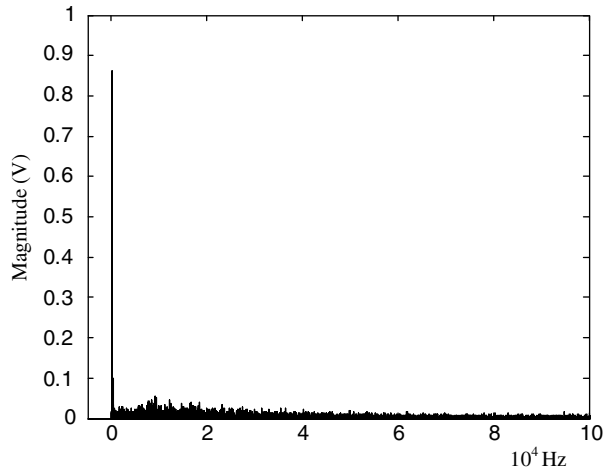


Figure 10.24 Spectrum of GA-optimized hybrid random pulse-position and pulse-width PWM inverter output.

Table 10.4 $\Delta Eval$ (%) of GA-optimized random PWM inverters.

Simulation results	GA random carrier-frequency PWM	GA random pulse-position PWM	GA random pulse-width PWM	GA Hybrid pulse-position and pulse-width PWM
THD %	−3.4 %	−17.04 %	−16.2 %	−21.4 %
Power of signal of PWM output	1.5 %	0.0 %	0.0 %	0.4 %
Power of fundamental component	4.0 %	13.22 %	12.50 %	17.1 %
Power of all harmonic components	−2.8 %	−22.08 %	−20.9 %	−27.6 %
Amplitude of fundamental	2.0 %	6.41 %	6.1 %	8.2 %
Amplitude of max harmonic	−72.4 %	−63.4 %	−65.4 %	−55.3 %

10.4.6 Switching Loss of GA-Optimized Random Single-Phase PWM Inverters

Switching energy losses of semiconductor devices are caused by charging and discharging output capacitance at turn-on and turn-off periods, which may be expressed as (Kazimierczuk, 2008),

$$E_{sw} = N_p \times C_o \times V_{dc}^2 \tag{10.16}$$

where C_o is the output capacitance, V_{dc} is the DC voltage, N_p is number of pulses, and E_{sw} is the switching energy loss.

Hence, the total switching loss in a fundamental cycle increases in proportion to the total pulse number in the cycle. Since the same pulse number is used, the total switching losses in various GA-optimized random PWM inverters are same as the total switching loss in the standard PWM inverter.

10.4.7 Linear Modulation Range of GA-Optimized Random Single-Phase PWM Inverters

In the linear modulation range, that is, modulation index is less than or equals 1, the average value of pulse width of the standard PWM inverter varies directly with the amplitude of the sinusoidal reference signal. Since the total number of pulses and average width of pulses are unchanged, the GA-optimized random PWM inverters retain the features of linear modulation. To demonstrate the linear modulation capability of the GA-optimized random PWM inverters, the GA-optimized random-pulse-position series at a modulation index of 0.8 (obtained in Section 10.3.2) is applied to the PWM inverter with various modulation indices and the harmonic evaluation results are listed in Table 10.5.

Table 10.5 Evaluation values of the GA-optimized random-pulse-position PWM.

Modulation index	THD	Power of signal	Power of fundamental	Power of all harmonics	Amplitude of fundamental	Amplitude of max harmonic
1	0.5036	0.6377	0.5085	0.1292	1.0083	0.1081
0.9	0.5699	0.5713	0.4311	0.1402	0.9285	0.0797
0.8	0.6420	0.5078	0.3596	0.1482	0.8481	0.1165
0.6	0.8479	0.3818	0.2221	0.1597	0.6665	0.2082
0.4	1.1828	0.2529	0.1054	0.1475	0.4592	0.2280
0.2	1.9573	0.1250	0.0259	0.0991	0.2275	0.1507
0.1	3.4643	0.0566	0.0044	0.0523	0.0933	0.0713

Evaluation values of the standard PWM are listed in Table 10.6 for comparison.

Table 10.6 Evaluation values of the standard PWM.

Modulation index	THD	Power of signal	Power of fundamental	Power of all harmonics	Amplitude of fundamental	Amplitude of max harmonic
1	0.5235	0.6377	0.5005	0.1372	1.0005	0.2104
0.9	0.6509	0.5713	0.4013	0.1700	0.8958	0.2605
0.8	0.7739	0.5078	0.3176	0.1902	0.7970	0.3182
0.6	1.0664	0.3818	0.1787	0.2031	0.5978	0.3724
0.4	1.4791	0.2529	0.0793	0.1736	0.3984	0.3255
0.2	2.3360	0.1250	0.0194	0.1056	0.1968	0.1881
0.1	3.5540	0.0566	0.0042	0.0525	0.0912	0.0920

Since the total pulse width remains unchanged, the power of the output signal of the GA-optimized random PWM is same as the standard PWM. The values of $\Delta Eval$ computed by Equation (10.15) are listed in Table 10.7.

Table 10.7 $\Delta Eval$ (%) for GA-optimized random PWM at different modulation indices.

Modulation index	THD	Power of signal	Power of fundamental	Power of all harmonics	Amplitude of fundamental	Amplitude of max harmonic
1	−3.80	0	1.56	−6.05	0.78	−48.62
0.9	−12.44	0	7.40	−17.65	3.65	−69.40
0.8	−17.04	0	13.22	−22.08	6.41	−63.39
0.6	−20.49	0	24.29	−21.37	11.49	−44.09
0.4	−20.03	0	32.91	−15.03	15.26	−29.95
0.2	−16.21	0	33.51	−6.16	15.60	−19.88
0.1	−2.52	0	4.76	−0.38	2.30	−22.50

The GA-optimized random-pulse-position PWM inverter has very good linear modulation capacity with lower THD and good evaluation values when the modulation index is between 0.2 and 0.9. When the modulation index equals 0.1 and 1, the GA-optimized random-pulse-position PWM is also superior to the standard PWM as observed from Table 10.7.

10.4.8 Implementation of GA-Optimized Random Single-Phase PWM Inverter

10.4.8.1 GA-Optimized Random-Carrier-Frequency PWM Inverter

The GA-optimized random-carrier-frequency series will be repeatedly applied to control the PWM inverter to replace the standard triangular carrier generator in order to minimize the total harmonic distortion.

10.4.8.2 GA-Optimized Random Pulse-Position and Pulse-Width PWM Inverters

The other three GA-optimized random PWM inverters are (1) random-pulse-position PWM inverter, (2) random-pulse-width PWM inverter, and (3) hybrid random pulse-position and pulse-width PWM inverter. The states of U_{ab} of the inverter must be translated into the switching states of the semiconductor switches, and this process is repeated in every fundamental cycle. For the inverter shown in Figure 10.25, the transformation relationship is as listed in Table 10.8.

10.4.9 Limitations of Reference Sinusoidal Frequency of GA-Optimized Random PWM Inverters

Because the optimization process is based on a fixed sinusoidal reference frequency, the optimum performance of the GA-optimized random PWM inverter is obtained for that

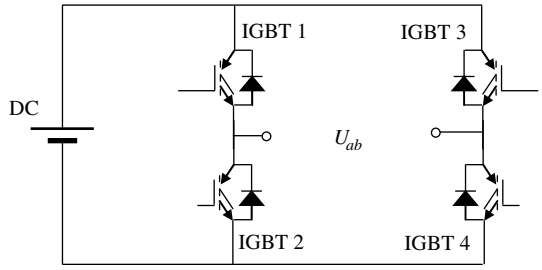


Figure 10.25 Semiconductor devices and U_{ab} pulse sequence.

Table 10.8 State of semiconductor switches according to the states of U_{ab}.

U_{ab}	IGBT 1	IGBT 2	IGBT 3	IGBT 4
−1	Off	On	On	Off
1	On	Off	Off	On
0	On	Off	On	Off
0	Off	On	Off	On

particular frequency. When the GA-optimized random PWM is used at a different reference frequency, the carrier frequency should be modified as follows:

$$f_{\_new\_carrier} = \frac{f_{\_GA\_carrier} \times f_{\_new\_reference}}{f_{\_GA\_reference}} \tag{10.17}$$

where $f_{\_new\_carrier}$ is the new carrier frequency, $f_{\_new\_reference}$ is the new reference frequency, $f_{\_GA\_carrier}$ is the carrier frequency when PWM inverter is optimized, and $f_{\_GA\_reference}$ is the reference sinusoidal frequency when PWM inverter is optimized.

An alternative solution is to construct a look-up table which stores the GA-optimized random switching series at various reference sinusoidal frequencies for the variable-frequency inverter. It is also possible to model the data in the look-up table by using a regression function or a neural network in order to maintain a lower THD.

10.5 MATLAB®/Simulink Programming Examples

Three MATLAB®/Simulink programming examples are presented in this section. (1) Single-phase sinusoidal PWM; (2) Evaluation of a four-pulse wave; (3) Random carrier-frequency PWM.

10.5.1 Programming Example 1: A Single-Phase Sinusoidal PWM

In this programming example, the reference sine wave has a frequency of 50 Hz and an amplitude of 0.8 V, while the triangular carrier wave has a frequency of 5 kHz and an amplitude of 1 V. The single-phase sinusoidal PWM may be modeled and simulated by the following steps.

Step 1 Build a Simulink model of the triangular carrier PWM 'single_phase_inverter.mdl' as shown in Figure 10.26.

The PWM model consists of a 'Sine Wave' block, two 'Relational Operator' blocks, a 'Repeating Sequence' block, and an 'Out1' block. The 'Relational Operator' blocks simulate the modulation operation, the 'Repeating Sequence' block simulates the triangular-carrier wave, and the 'Out1' block outputs U_{ab} for evaluation. The parameters of these blocks are as listed in Table 10.9.

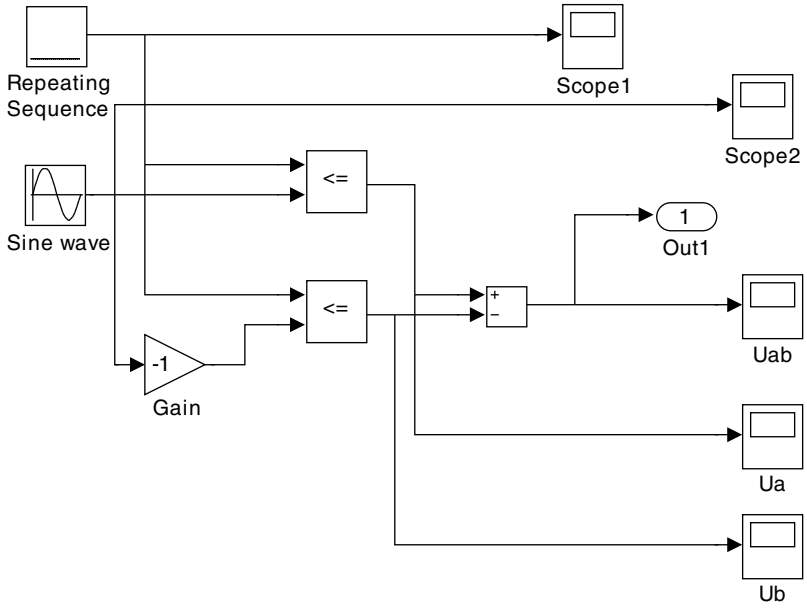


Figure 10.26 Simulink model of triangular carrier PWM.

Table 10.9 Parameters of Simulink model 'single_phase_inverter.mdl'.

Name	Function	Parameter
Sine Wave	Generate sinusoidal reference wave	Amplitude = 0.8 Frequency = 2*pi*50 Phase = 0
Repeating Sequence	Generate 5 kHz triangular carrier wave	Time values: T_V created by the MATLAB® program Output values: o_V created by the MATLAB® program
Simulation parameter	Manage simulation process	Solver options: Fixed-step Fixed-step size:1/204800 Simulation time: 0–0.2 s

Step 2 Write the following MATLAB® program (with filename 'Evaluation.m' on the book companion website) for waveform evaluation.

```
function y=Evaluation(x)                % Evaluate x
N=1024*4;
H_pwm=fft(x,N);                         % FFT calculation
Harmonica = H_pwm(1:N/2);               % Cut half of FFT
ss=abs(Harmonica);                      % Calculate magnitude of FFT
                                          components
s_V=ss/(N/2);                           % Calculate real amplitude of output
                                          components
assignin('base','s_V',s_V);             % Assign array 's_V' into workspace
s_THD=0;                                % Calculate THD of waveform
for i=3:N/2,
s_THD=s_THD + s_V(i)^2;
end
THD=sqrt(s_THD)/s_V(2)
V_Fundamental=s_V(2)                    % Obtain amplitude of fundamental
                                          waveform
s_Vh=s_V;
s_Vh(2)=[ ];                            % Cut off fundamental component
Max_Harmonic =max(s_Vh)                 % Calculate amplitude of max
                                          harmonic component
P_Fundamental=s_V(2)^2/2                % Calculate power of fundamental
                                          component
DC=s_V(1)/2                             % Obtain amplitude of DC component
s_Vfft=s_V;
Pt_FFT=DC^2 + sum(s_Vfft.^2)/2          % Total power of FFT components
Pt_PWM=sum(x.^2)/4098                   % Total power of standard PWM signal
s_Vfft(1)=[ ];                          % Cut off DC component
s_Vfft(1)=[ ];                          % Cut off fundamental component
P_all_harmonics = sum(s_Vfft.^2)/2      % Calculate power of all harmonic
                                          components
tx=(0:1:(N/2-1))*50;                    % Set x-axis as Hz
bar(tx,s_V)                             % Plot spectrum of PWM output
axis([-5000 50000 0 1]);                % Set x-axis and y-axis ranges
end
```

Step 3 Enter the following code to create the two arrays T_V and o_V in the MATLAB® workspace. Run the Simulink model 'single_phase_inverter.mdl', and run the evaluation program written in **Step 2.** The two arrays (T_V and o_V) are used by the 'Repeating Sequence' block to simulate the triangular carrier wave with a frequency of 5 kHz and an amplitude of 1 V.

```
NN=200; % Switch frequency 5kHz, 200 points during 0.02 seconds
for I=1:NN/2                            % Create +1 and -1V voltage series
  o_V(2*I)=1;
  o_V(2*I-1)=-1;
end
o_V(NN+1)=-1;
T_V=(0:0.02/NN:0.02);                   % Create time series of carrier wave
assignin('base','T_V',T_V);             % Save to workspace
assignin('base','o_V',o_V);             % Save to workspace
```

Then enter the following MATLAB® commands to evaluate the PWM output:

```
[tout,xout,yout]=sim                    % Run Simulink model 'single_phase_
  ('single_phase_inverter',0.02);         inverter.mdl'
Evaluation(yout);                       % Call MATLAB program 'Evaluation.m'
```

The evaluation results are shown on the computer screen as follows and a spectrum of the PWM output wave is shown in Figure 10.8.

```
THD= 0.7739
V_Fundamental= 0.7970
Max_Harmonic= 0.3182
P_Fundamental= 0.3176
DC= 0
Pt_FFT= 0.5078
Pt_PWM= 0.5076
P_all_harmonics= 0.1902
```

The evaluation results are as listed in the column 'U_{ab} of standard 5 kHz PWM' in Table 10.1.

Step 4 Enter the command 'plot(tout, yout)' to plot the single-phase sine PWM output waveform U_{ab} with 5 kHz carrier frequency, as shown in Figure 10.27.

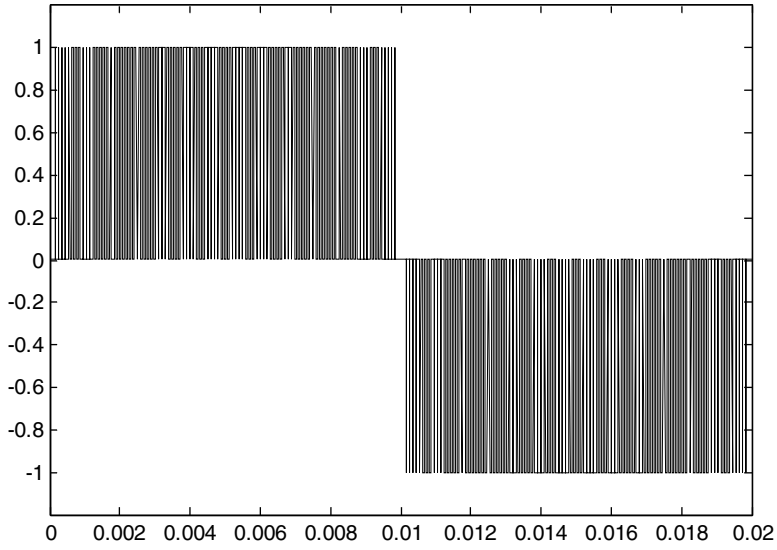


Figure 10.27 Waveform U_{ab} of sine PWM output with 5 kHz carrier frequency.

10.5.2 Programming Example 2: Evaluation of a Four-Pulse Wave

A four-pulse wave is modeled and simulated by the following steps.

Step 1 Build a Simulink model 'Four_pulse.mdl' as shown in Figure 10.28.

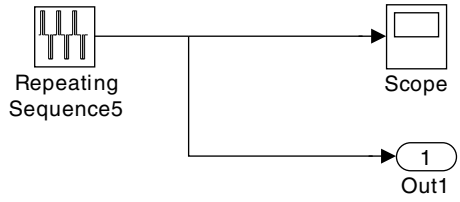


Figure 10.28 Simulink model of the four-pulse wave.

Step 2 Input the following vectors into the 'Repeating Sequence' block.

```
Time Values =
[0,0.00199,0.00199,0.0049,0.0049,0.005,0.005,0.00791,0.00791,0.0119,
0.0119,0.01481,0.01481,0.0152,0.0152,0.01811,0.01811,0.02]

Output Values =
[0,0,1,1,0,0,1,1,0,0,-1,-1,0,0,-1,-1,0,0]
```

Step 3 Enter the following MATLAB® commands (or run the program 'Run_Exp2.m' on the book companion website).

```
[tout,xout,yout]=sim      % Call Simulink model named as 'Four_pulse.mdl'
  ('Four_pulse',0.02);
Evaluation(yout);         % Call MATLAB program 'Evaluation.m'
bar(s_V);                 % Plot bar graph
axis([-1 30 0 1.1]);      % Set x-axis and y-axis ranges
```

The evaluation results are as follows and a spectrum of the PWM output shown in Figure 10.6 is also obtained.

```
THD = 0.4376
V_Fundamental = 0.9882
Max_Harmonic = 0.3007
P_Fundamental = 0.4883
DC = 2.4414e-004
Pt_FFT = 0.5818
Pt_PWM = 0.5815
P_all_harmonics = 0.0935
```

10.5.3 Programming Example 3: Random Carrier-Frequency PWM

A random carrier-frequency PWM may be modeled and simulated by the following steps.

Step 1 Open the Simulink model 'single_phase_inverter.mdl' in Programming Example 1 and save it as 'triangle.mdl'.

Step 2 Replace the elements of the arrays T_V in the 'Repeating Sequence' block in the Simulink model with random numbers by entering following MATLAB® code:

```
clear;
clc;
NN=200;         % Switch frequency 5kHz, 200 points during 0.02 seconds
for I=1:NN/2          % Create +1 and -1V voltage series
    o_V(2*I)=1;
    o_V(2*I-1)=-1;
end
o_V(NN+1)=-1;
T_V(1) =0; % Create time series of carrier wave with random number
for i=2:NN+1,
T_V(i)=T_V(i-1)+0.00002+rand(1)*0.00016;
  % rand(1) yields a pseudorandom number range as -1~ +1
End
assignin('base','T_V',T_V);     % Save to workspace
assignin('base','o_V',o_V);     '% Save to workspace
```

Step 3 Enter the following MATLAB® commands to simulate the random carrier-frequency PWM and evaluate the waveforms by the evaluation program written in **Step 2** in Programming Example 1.

```
[tout,xout,yout]=              % Call Simulink model named as 'triangle.mdl'
  sim('triangle',0.02);
Evaluation(yout);              % Call MATLAB program 'Evaluation.m'
```

After running the above MATLAB® program and the evaluation program, a possible evaluation result is obtained as follows:

```
THD= 0.7708
V_Fundamental= 0.8009
Max_Harmonic= 0.0813
P_Fundamental= 0.3207
DC= 0.0034
Pt_FFT= 0.5112
Pt_PWM= 0.5110
P_all_harmonics= 0.1905
```

A spectrum of the PWM output similar to that in Figure 10.10 will also be obtained.

Because the carrier frequency is randomly produced, the evaluation results may vary from one simulation to another.

Step 4 Enter the command 'plot(tout, yout)' to plot the random carrier-frequency PWM output U_{ab}, as shown in Figure 10.29.

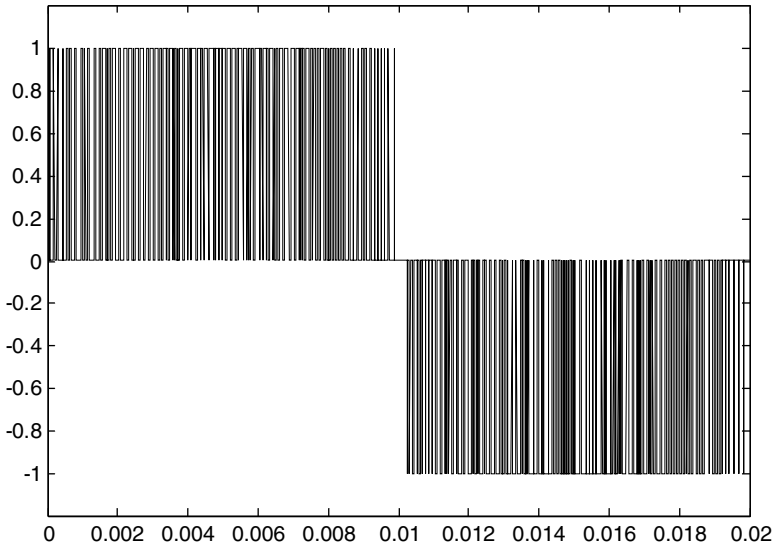


Figure 10.29 Waveform U_{ab} of the random carrier-frequency PWM.

10.6 Experiments on Various PWM Strategies

The GA-optimized single-phase random-carrier-frequency PWM inverter is implemented and compared with the standard PWM inverter by the hardware setup in Figure 10.30. The experimental facility consists of a TMS320F2812 DSP board, an IRAMX16UP60A inverter module, a digital oscilloscope, a PC host computer, and a DC power supply. The DSP facilitates the implementation of various PWM strategies in real-time.

10.6.1 Implementation of PWM Methods Using DSP

DSP TMS320F2812 is able to generate two types of PWM waveform, *viz*. symmetric waveform and asymmetric waveform. The symmetric PWM has less harmonic distortion (Texas Instruments Incorporated, 2003) than the asymmetric PWM, but the latter gives twice the resolution. In this chapter, the symmetric PWM is chosen for implementing the various PWM methods. The main steps (Texas Instruments Inc., 2003) to operate PWM in TMS320F2812 DSP chip are:

a. Initialize a 16-bit 'Timer Period Register' and a 16-bit 'Full Compare Register', respectively.

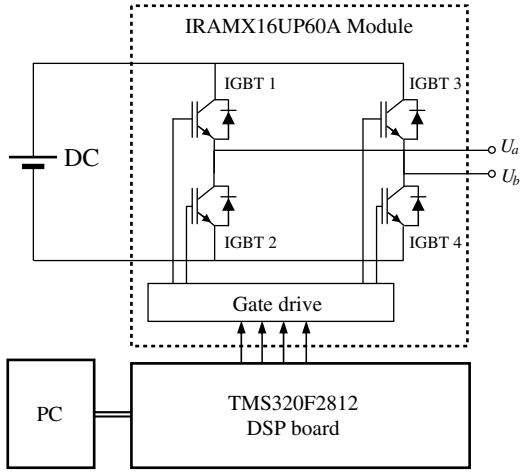


Figure 10.30 PWM inverter experimental system. (Reproduced by permission of K.L. Shi and H. Li, "Optimized PWM strategy based on genetic algorithms," *IEEE Transaction on Industrial Electronics*, **52**(5), 2005: 1458–1461. © 2005 IEEE.)

b. Load the PWM period into the 'Timer Period Register' and the sinusoidal reference value into the 'Full Compare Register', respectively.
c. Compare the output of a 'General Purpose Timer' with the 'Timer Period Register'. When they are equal, two pairs of pulse signals are generated from a 'Full Compare Unit' and a software interrupt is triggered.
d. Upon receiving the software interrupt signal, the program updates the value in the 'Timer Period Register' and the value in the 'Full Compare Register' according to the desired carrier frequency requested by different PWM methods.
e. Repeat from step (*b*) to step (*d*).

Three PWM methods are implemented by the DSP-based inverter. The operating strategies are described as follows:

A. Standard PWM method, which applies a constant carrier frequency;
B. Random carrier-frequency PWM, which employs a random carrier frequency generated by a pseudorandom function that depends on a special seed sequence; and
C. GA-optimized random-carrier-frequency PWM method, in which the random-carrier-frequency series is stored in a DSP, the optimization having been performed optimized off-line by GA.

It should be noted that the above implementation of the GA-optimized PWM method does not incur extra hardware cost and programming complexity compared with other PWM techniques.

10.6.2 Experimental Results

In order to compare the performance of three PWM methods, the normalized output voltage of PWM inverter and the corresponding harmonic spectrum at different switching frequencies are displayed on a digital oscilloscope as shown in Figures 10.31–10.33.

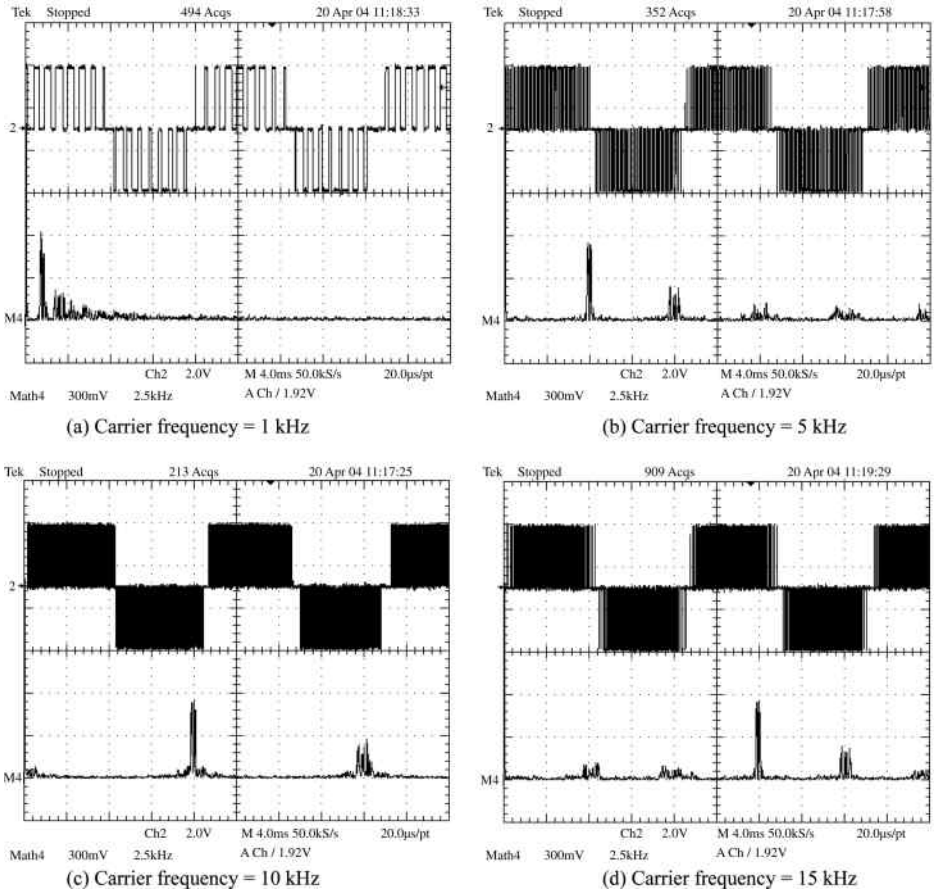


Figure 10.31 Output waveforms of standard PWM inverter. Upper trace: Voltage U_{ab} (Time scale: 4 ms/div; voltage scale: 2 V/div) Lower trace: Frequency spectrum (Frequency scale: 2.5 kHz/div; voltage scale: 300 mV/div). (a) Carrier frequency = 1 kHz (b) Carrier frequency = 5 kHz (c) Carrier frequency = 10 kHz (d) Carrier frequency = 15 kHz. (Reproduced by permission of K.L. Shi and H. Li, "Optimized PWM strategy based on genetic algorithms," *IEEE Transaction on Industrial Electronics*, **52**(5), 2005: 1458–1461. © 2005 IEEE.)

10.6.2.1 Standard PWM

When the carrier frequency is constant, the DSP-controlled inverter outputs standard PWM waves. Figure 10.31 shows the output voltage U_{ab} when the carrier frequency is 1 kHz, 5 kHz, 10 kHz, and 15 kHz, respectively.

10.6.2.2 Random Carrier-Frequency PWM

When the carrier frequency is randomly changed, the DSP generates a random carrier-frequency PWM. Figure 10.32 shows the output voltage U_{ab} when the mean frequency of the carrier is 1 kHz, 5 kHz, 10 kHz, and 15 kHz, respectively.

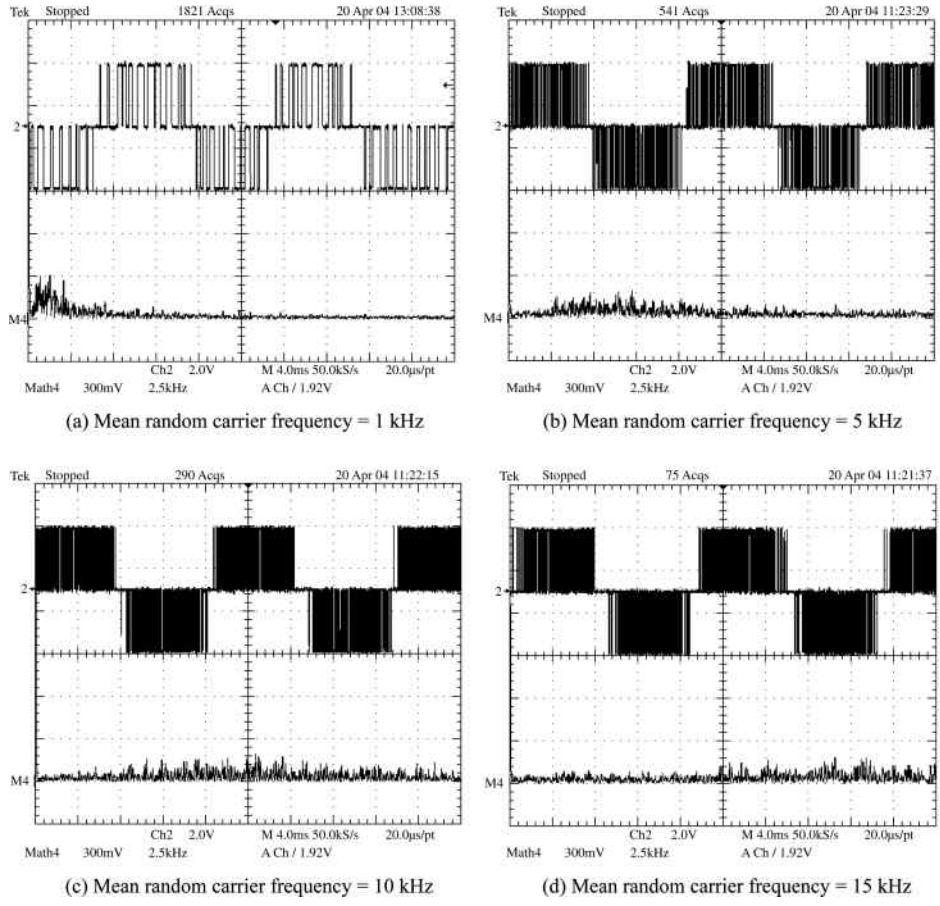


Figure 10.32 Output waveforms of random PWM inverter. Upper trace: Voltage U_{ab} (Time scale: 4 ms/div; voltage scale: 2 V/div) *Lower trace: Frequency spectrum (Frequency scale: 2.5 kHz/div; voltage scale: 300 mV/div)* (a) Mean random carrier frequency = 1 kHz (b) Mean random carrier frequency = 5 kHz (c) Mean random carrier frequency = 10 kHz (d) Mean random carrier frequency = 15 kHz. (Reproduced by permission of K.L. Shi and H. Li, "Optimized PWM strategy based on genetic algorithms," *IEEE Transaction on Industrial Electronics*, **52**(5), 2005: 1458–1461. © 2005 IEEE.)

10.6.2.3 GA-Optimized Random-Carrier-Frequency PWM

The GA-optimized random-carrier-frequency series is obtained off-line by a MATLAB® program and then copied to the DSP PWM program to replace conventional carrier operation. When the optimized program is loaded into the experimental DSP device, the inverter gives optimized performance as presented in Figure 10.33.

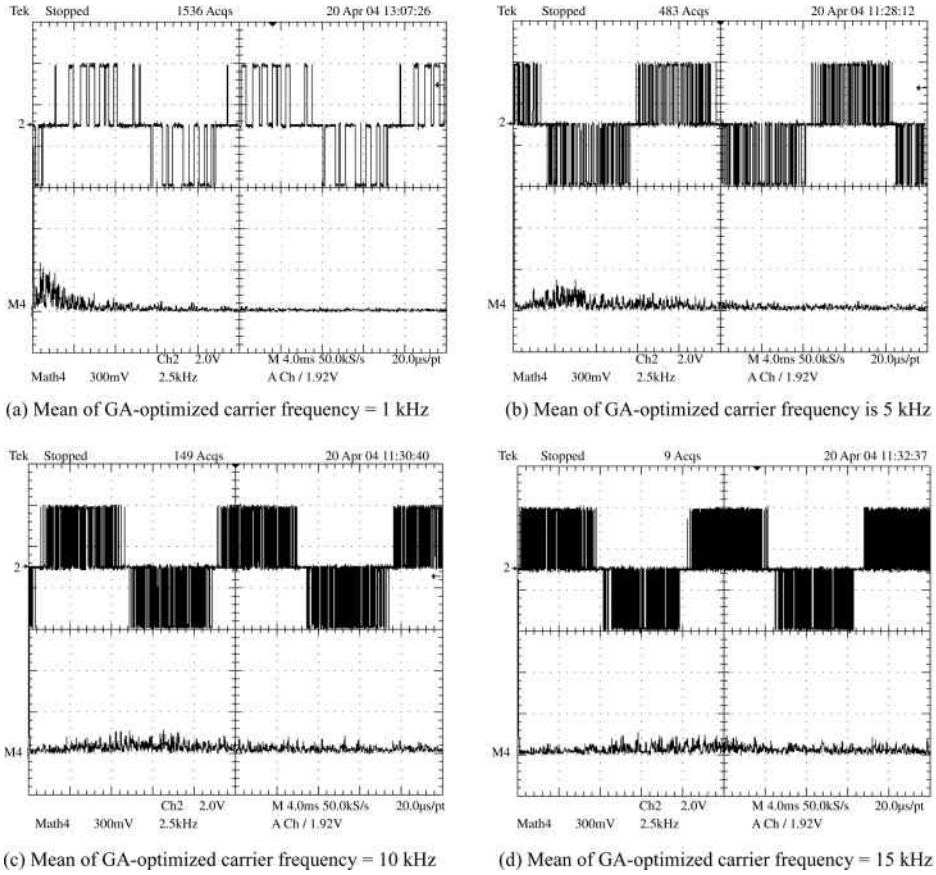


Figure 10.33 Output waveforms of GA-optimized random PWM inverter Upper trace: Voltage U_{ab} (Time scale: 4 ms/div; voltage scale: 2 V/div). Lower trace: Frequency spectrum (Frequency scale: 2.5 kHz/div; voltage scale: 300 mV/div). (a) Mean of GA-optimized carrier frequency = 1 kHz (b) Mean of GA-optimized carrier frequency is 5 kHz (c) Mean of GA-optimized carrier frequency = 10 kHz (d) Mean of GA-optimized carrier frequency = 15 kHz. (Reproduced by permission of K.L. Shi and H. Li, "Optimized PWM strategy based on genetic algorithms," *IEEE Transaction on Industrial Electronics*, **52**(5), 2005: 1458–1461. © 2005 IEEE.)

10.7 Summary

In this chapter, real-coded genetic algorithm has been employed to optimize a single-phase inverter by using four random PWM methods. They are (1) GA-optimized random-carrier-frequency PWM, (2) GA-optimized random-pulse-position PWM, (3) GA-optimized random-pulse-width PWM, and (4) GA-optimized hybrid random pulse-position and pulse-width PWM. Simulation studies have demonstrated that the GA-optimized random PWM is superior to conventional PWM techniques when harmonic energy spread and THD are considered. DSP-based inverter experiments confirm the feasibility of implementation of GA-optimized random-carrier-frequency PWM. The present investigation has demonstrated the capability of the GA-optimized random PWM technique in improving the power quality of inverters. Further work would be to extend the technique to three-phase PWM inverters that find wider applications in induction motor drives.

References

Bech, M.M., Pedersen, J.K., Blaabjerg, F., and Trzynadlowski, A.M. (1999) A methodology for true comparison of analytical and measured frequency domain spectra in random pwm converters. *IEEE Transactions on Power Electronics*, **14**(3), 578–586.

Boys, J.T. and Handley, P.G. (1992) Spread spectrum carrier: low noise modulation technique for PWM inverter drives. *Proc IEE, Part B*, **139**(3), 252–260.

Chiasson, J.N., Tolbert, L.M., McKenzie, K.J., and Du, Z. (2004) A complete solution to the harmonic elimination problem. *IEEE Transactions on Power Electronics*, **19**(2), 491–499.

Habetler, T.G. and Divan, D.M. (1991) Acoustic noise reduction in sinusoidal PWM drives using a randomly modulated carrier. *IEEE Transactions on Power Electronics*, **6**(3), 356–363.

Kazimierczuk, M.K. (2008) *Pulse-width DC-DC Power Converters*, John Wiley & Sons, Inc., New York, NY.

Kirlin, R.L., Kwok, S., and Trzynadlowski, A.M. (1993) Power spectra of a PWM inverter with randomized pulse position. Proc. PESC'93, pp. 1041–1047.

Kirlin, R.L., Kwok, S., Legowski, S., and Trzynadlowski, A.M. (1994) Power spectra of a PWM inverter with randomized pulse position. *IEEE Transactions on Power Electronics*, **9**(5), 463–472.

Legowski, S. and Trzynadlowski, A.M. (1989) Hypersonic MOSFET-based power inverter with random pulse width modulation. Conference Record 1989 IEEE-IAS Annual Meeting, pp. 901–903.

Legowski, S. and Trzynadlowski, A.M. (1990) Power-MOSFET, hypersonic inverter with high quality output current. Proc. APEC'90, pp. 3–7.

Lathi, B.P. (2005) *Linear System and Signals*, Second edition, Oxford University Press, Oxford, pp. 247–634.

Ozpineci, B., Tolbert, L.M., and Chiasson, J.N. (June 2004), Harmonic optimization of multilevel converters using genetic algorithms. Proceedings of Power Electronics Specialists Conference (PESC 04) and vol. 5, pp. 3911–3916.

Pedersen, J.K. and Blaabjerg, F. (1992) Implementation and test of a digital quasi-random modulated SFAVM PWM in a high performance drive system. Proceedings of IECON'92 pp. 256–270.

Shi, K.L., Chan, T.F., Wong, Y.K., and Ho, S.L. (2002) Speed estimation of an induction motor drive using an optimized extended Kalman filter. *IEEE Transactions on Industrial Electronics*, **49**(1), 124–134.

Shi, K.L. and Li, H. (November 2003) An optimized PWM method using genetic algorithms. Proceedings of IECON'03 – 29th Annual Conference of the IEEE Industrial Electronics Society Roanoke, Virginia, USA, pp. 7–11.

Texas Instruments Incorporated (November 2003) TMS320F28x DSP Event Manager (EV) Reference Guide, Literature Number: SPRU065B.

Trzynadlowski, A.M., Blaabjerg, F., Pedersen, J.K. *et al.* (1994) Random pulse width modulation techniques for converter-fed drive systems – a review. *IEEE Transactions on Industry Applications*, **30**(5), 1166–1175.

Tse, K.K., Chung, H., Hui, S.Y.R., and So, H.C. (2000) A comparative investigation on the use of random modulation schemes for dc/dc converters. *Transactions on Industrial Electronics*, **47**, 245–252.

Zhang, J., Chung, H.S.-H., Lo, W.-L. *et al.* (2001) Implementation of a decoupled optimization technique for design of carrier regulators using genetic algorithms. *IEEE Transactions on Power Electronics*, **16**(6), 752–763.

Zhou, K. and Wang, D. (2002) Relationship between space-vector modulation and three-phase carrier-based PWM: a comprehensive analysis. *IEEE Transactions on Industrial Electronics*, **49**(1), 186–196.

11

Experimental Investigations

11.1 Introduction

This chapter discusses how the algorithms of induction motor intelligent control are realized physically by hardware. The inverter, the controller, and the induction motor are essential components for implementing the control algorithms. Different control algorithms, however, may require different hardware configurations. For example, the open-loop V/Hz controller can be implemented using analog components, and the vector controller can be implemented using digital devices such as a DSP (Ben-Brahim and Kawamura, 1992; Kubota, Matsuse and Nakano, 1993; Xu and Novotny, 1996) or a PC (Zhong, Messinger and Rashad, 1991). The fuzzy control algorithm can be implemented using a fuzzy microcontroller or a DSP device with a fuzzy-logic program; the neural-network algorithm can be implemented with neural network devices; the expert system algorithm may be implemented by a DSP device with an expert-system control program. It should be mentioned that the implementation of the expert-system control algorithm requires a high-precision speed sensor, such as the expensive Gurley's Model R158 Rotary Incremental Encoder.

Induction motor controllers with analog components have several drawbacks. Aging and temperature rise bring about component variations and regular system adjustments are required. As the parts count increases the reliability of the system decreases. Analog components also raise tolerance issues and upgrades are difficult to achieve as the design is hardwired. Digital systems offer improvements over analog designs. Drift is eliminated since most functions are performed digitally. Upgrades can easily be made in software and parts count is also reduced since most calculation functions can be handled on a single chip. Two fixed-point-DSP motor controllers, namely Analog Devices Inc.'s ADMC331 and Texas Instruments's TMS320F240, are available commercially. Each system integrates a digital signal processor chip with the peripherals of a micro-controller and hence is suitable for induction motor control applications. The fixed-point DSP is preferred for two reasons. Firstly, its cost is much less than that of floating point DSPs. Secondly, a dynamic range of 16 bits and the capability of executing 26 MIPS (million instructions per second) are sufficient for most motor controllers. ADMC331 and TMS320F240 have comparable technical specifications and performance levels, such as the executing capability, memory size, and micro-controller

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

peripherals, but the former is less expensive. Hence, a Development Tool Kit ADMC331-ADVEVALKIT (delivered by Analog Devices Inc.) is chosen for the experimental studies. The kit is complete with hardware (ADMC331 processor board and ADMC connector board), software (assembler, linker and debugger), and serial cable for connection to a PC.

With the ADMC331 as the core element, a DSP-based hardware system is configured to implement some of the intelligent control algorithms. The system consists of a power module, a DSP-processor, 3-phase current sensors, an encoder, an induction motor, a host PC, a data-acquisition card and a data-acquisition PC. Based on the special hardware configuration and some Assembly and C++ referencing programs and manuals delivered by Analog Devices Inc. Amirix Inc., and Advantech Co., the following program files in Assembly language and C++ language have been developed for the experimental studies: (*a*) DSP main program and subroutines for induction motor run-up experiment, (*b*) DSP main program, routines, and subroutines of fuzzy/PI controller, (*c*) DSP FOC main program for the GA-EKF experiment, (*d*) a data-acquisition C++ program for the DSP code debug on a PC.

Four experiments have been performed: (*1*) determination of the electrical parameters of the 147-W induction motor used in the drive system, (*2*) an induction motor run-up experiment to verify the induction motor model, PWM model, encoder model, and decoder model built in Chapter 3, (*3*) a DSP based experiment on the fuzzy/PI two-stage controller to verify the control algorithm proposed in Chapter 6, and (*4*) an experiment to verify the GA-EKF speed estimation algorithm proposed in Chapter 9. Section 11.8 presents four DSP Programming examples on a floating-point digital signal processor (DSP) TMS320F28335 delivered by Texas Instruments Inc.

11.2 Experimental Hardware Design for Induction Motor Control

The experimental hardware of the intelligent control is configured by the following components and is shown in Figure 11.1, while photographs of the experimental system are shown in Figure G.1 and G.2 of Appendix G.

11.2.1 Hardware Description

11.2.1.1 DSP ADMC331 Processor Board

Analog Devices Inc.'s ADMC331 is a low cost, single-chip DSP-based controller, which is suitable for implementing an induction motor drive. The ADMC331 integrates a 26 MIPS, fixed-point DSP core with a complete set of motor control peripherals. The DSP core of the ADMC331 is the ADSP-2171 with ADSP-2100 based architecture, which is completely code compatible with the ADSP-2100 DSP family and combines three computational units, data-address generators and a program sequencer. The computational units comprise an ALU, a multiplier/accumulator (MAC) and a barrel shifter (comprising the shifter array, the OR/PASS logic, the exponent detector, and the exponent compare logic). The ADSP-2171 has instructions for bit manipulation, multiplication (X squared), biased rounding and global interrupt masking. In addition, two flexible, double-buffered, bidirectional, and synchronous serial ports are included in the ADMC331.

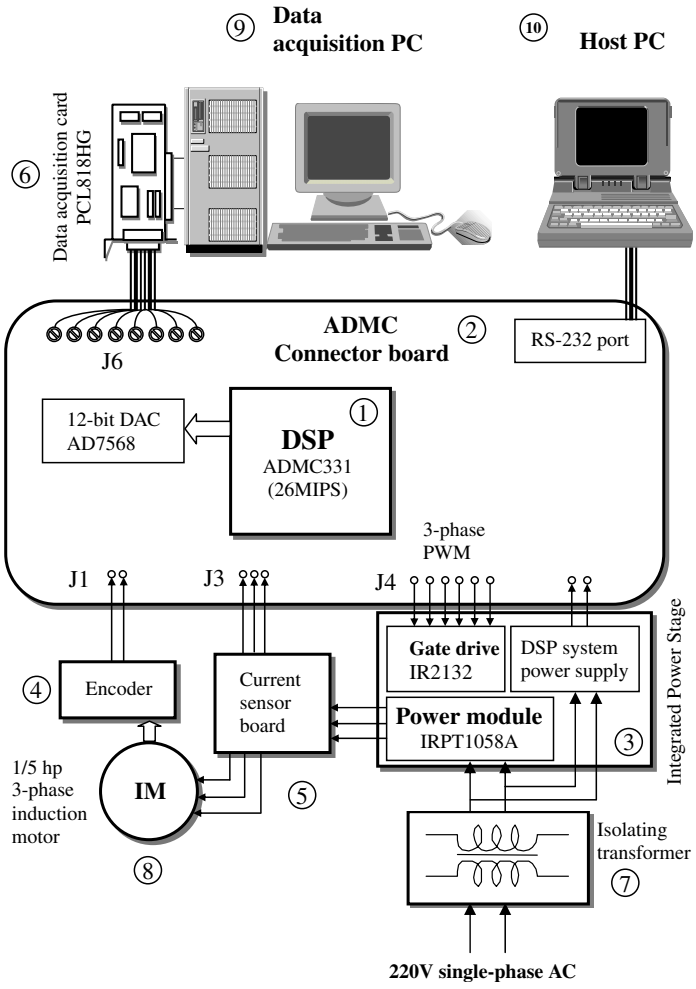


Figure 11.1 Hardware configuration for the experiments on intelligent control of induction motor. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

	Component	Manufacturer
(1)	ADMC331 Processor Board	Analog Devices Inc.
(2)	ADMC Connector Board	Analog Devices Inc.
(3)	Integrated Power Stage (IRPT1058A)	International Rectifier Inc.
(4)	Encoder (GBZ02)	China Sichuan Opto-electronic Co.
(5)	Current Sensor Board (3I411A)	China WB Automation Institute.
(6)	Data-Acquisition Card (PCL818HG)	Taiwan Advantech Co. Ltd.
(7)	Isolating Transformer	China Chengdu Transformer Co.
(8)	AC Induction Motor (Model 295)	USA Bodine Electric Co.
(9)	Data-Acquisition PC (PII350)	Intel Inc.
(10)	Host PC (ThinkPad 600E PII 300)	IBM Co.

The ADMC331 provides 2K × 24-bit program memory RAM, 2K × 24-bit program memory ROM and 1K × 16-bit data-memory RAM. The program and data-memory RAM can be bootloaded through the serial port from a serial ROM (SROM), E2PROM, asynchronous (UART) connection or synchronous connection. The program memory ROM includes a monitor that adds software-debugging features through the serial port. In addition, a number of pre-programmed mathematical and motor control functions are included in the program memory ROM. The motor control peripherals of the ADMC331 include a 16-bit center-based PWM generation unit that can be used to produce high accuracy PWM signals with minimal processor overhead and seven analog input channels. The device also contains two auxiliary 8-bit PWM channels, a 16-bit watchdog timer and it has expansion capability through the serial ports and 24-bit digital I/O ports.

11.2.1.2 ADMC Connector Board

The ADMC connector board is a part of the development tool kit ADMC331-ADVEVALKIT produced by Analog Devices Inc. The connector board offers the following functions:

- Connector (J1) accepts speed signals from the encoder.
- Connector (J3) provides analog inputs that are fed to the ADC interface circuitry of the ADMC331 processor chip. It can receive the signals from the current sensors.
- Connector (J4) brings the six PWM output signals from the DSP ADMC331 to International Rectifier PowIRtrain module.
- An eight-channel, 12-bit serial Digital to Analog Converter (DAC AD7568 chip) interfaces to the DSP-based motor controller through serial port SPORT0 on the ADMC331 chip. The eight analog outputs are brought to the data-acquisition PC through connector (J6).

11.2.1.3 Integrated Power Stage

The integrated power stage adopts the IRPT1058C PowIRtrain of International Rectifier Inc., which provides the complete conversion function for a 0.75 hp (0.56 kW) induction motor controller with variable frequency and variable voltage. The PowIRtrain combines a power module IRPT1058A with a Driver-Plus Board IRPT1058D. The power module IRPT1058A contains a single-phase input bridge rectifier and a 3-phase IGBT inverter. Figure 11.2 shows the block diagram of the power module.

The specifications of IRPT1058A are as follows.

- Input Power: Voltage 220 V, single-phase, 50/60 Hz
- Output Power: Voltage 0–230 V (defined by external PWM control)
- Pulse deadtime: 0.8 µs
- Minimum input pulse width: 1 µs
- DC link voltage: 230 V
- Isolation voltage: 2500 V rms
- DC bus filter capacitor: 2 × 680 µF, 400 V

The Driver-Plus Board IRPT1058D contains DC link capacitors, capacitor soft charge function, gate driver IR2132 for insulated gate bipolar transistors (IGBT), DC bus voltage

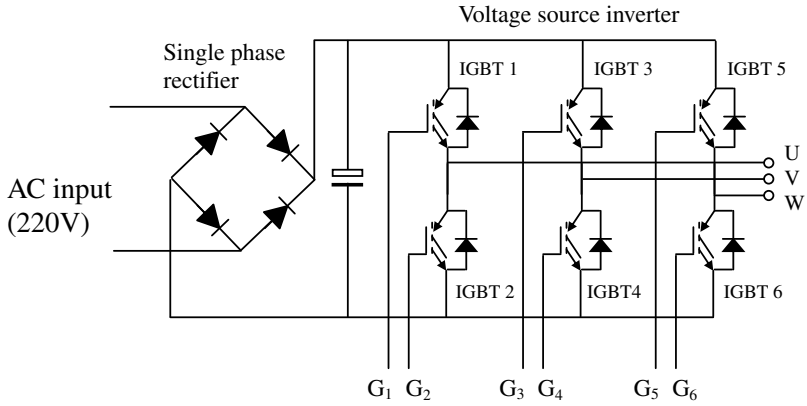


Figure 11.2 Power module IRPT1058A.

and current feedback circuit, protection circuitry and system power supply. Pulse-Width Modulated (PWM) signals from the DSP ADMC331 are input to the gate drive of the inverter IGBT switches to produce a 3-phase voltage of variable magnitude and frequency. The system power supply offers the user 5 and 15 V to power the DSP controller.

11.2.1.4 Encoder

An encoder manufactured by China Sichuan Opto-electronic Co. is used for the motor speed measurement. The encoder is an optoelectronic feedback device that uses a patterned optical mask and a LED light source and transistor photosensor pair. As the motor shaft rotates, the light source either passes through the disk, or is blocked by the disk. The two-emitter/detector pairs produce two digital output waveforms which are 90 degrees out of phase with each other. The various components making up the encoder are shown in Figure 11.3.

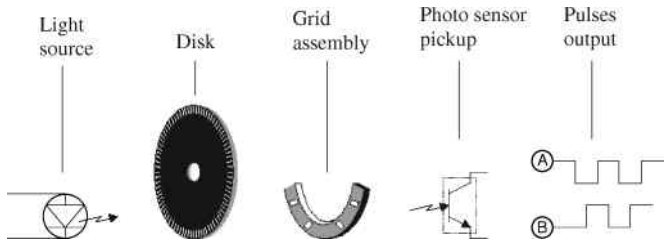


Figure 11.3 Various components making up the encoder.

Because the ADMC331 does not have an on-chip encoder interface, the encoder is interfaced to the AMDC331 using programmable I/O (PIO) lines. A DSP software, ENCODER.DSP is used to calculate the shaft speed.

Encoder features:
Model GBZ02

- 200 pulses/revolution
- Rise time <100 ns
- Fall time <100 ns
- Output voltage $V_H = 2.5$ V, $V_L = 0$ V
- Single $+5$ V DC supply.

11.2.1.5 Current Sensor Board

The current sensor board is used to produce three analog voltages proportional to the motor phase currents. These analog voltages are scaled for input to the ADC on the connector (J3) of the ADMC331 DSP. This board is used to sense currents up to 10 A and provides a voltage output suitable for sampling with an analog to digital converter (ADC). The current sensor board is useful in motion control and power supply designs. It has the following features:

- Two phase current sensing based on closed-loop Hall effect current transducer.
- Single $+5$ V DC supply.
- Working current: 42 mA.
- Low pass filtering on outputs.

11.2.1.6 PC Data-Acquisition Card

Analog to digital (A/D) conversion changes analog voltage or current levels into digital information. The conversion is necessary for the computer to process or store the signals. PCL-818HG of Advantech Co. is a low-cost high-performance data-acquisition board. It is also called a PC-based data-acquisition board. PCL818HG is configured on an ISA slot of PC motherboard and is used to acquire the information of voltages, currents, and rotor speed from the induction motor drive. Specifications of the PCL818HG board are listed as follows.

• A/D conversion time:	8 µs
• Maximum data throughput:	100 kHz for input range ± 10 V and ± 5 V 35 kHz for input range ± 1 V, ± 0.5 V, and ± 0.1 V
• Accuracy:	0.01–0.04 %
• Channels:	16 single-ended or 8 differential analog inputs
• Resolution:	12 bits A/D converter
• Input range selection:	Software controlled
• Data transfer:	Interrupt (IRQ) in Chapter 1 or DMA in Chapter 3
• Input impedance:	10 MΩ
• Input overvoltage:	± 30 V DC max.

11.2.1.7 Isolation Transformer

An isolation transformer (Single-phase 220 V 300 W) manufactured by China Chengdu Transformer Company is used to provide the following benefits:

- Improved safety.
- Reduced EMI emissions onto the power lines. As the controller is modified by adjusting parameters, power quality issues may arise.

11.2.1.8 AC Induction Motor

The motor is a Bodine Electric Company (USA) model 295, 147-W (1/5 HP), 230-V induction motor. The motor is star or 'Y' connected with no access to the neutral point. The motor ratings are as follows:

• Rated voltage, per phase:	230 V AC
• Rated current, per phase:	2 A
• Starting current:	4.5 A
• Stator resistance per phase:	14.6 Ω
• Pole number:	4
• Net weight:	30 pounds

11.2.1.9 Data-acquisition PC

A Pentium II 350 PC is used for the data-acquisition PC in the experiments of the DSP-based induction motor drive. It performs the following functions:

- Monitoring the drive system by acquiring data from the ADMC connector board, through the data-acquisition card PCL818HG mounted on an ISA port of the personal computer.
- Storing, plotting, and analyzing the acquired data.

11.2.1.10 Host PC

A notebook computer (IBM ThinkPad 600E PII 300) is used as a host PC in the experiments of the DSP-based induction motor drive. It is used for:

- Loading an executable program to the DSP ADMC311 via a RS232 port on the host computer.
- Starting and debugging the program for driving the induction motor.

11.2.1.11 Power Budget of DSP Controller

Table 11.1 summarizes the power budget for the electronics in the experimental system. Note that this budget does not include the drive current required by the induction motor. The motor power requirements (147 W) are well within the PowIRtrain's 560 W rating.

Table 11.1 Power budget for the electronics in the experimental system.

Component	Voltage	Current(max)
ADMC connector board, ADMC331 processor board	5VDC	200 mA
PowIRtrain	5VDC	25 mA
	15VDC	25 mA
Encoder	5VDC	25 mA
Current sensor board	5VDC	50 mA
	Total	325 mA

11.3 Software Development Method

Figure 11.4 shows the software development process for creating an application to run on the ADMC331. It comprises the following steps:

1. A source code in ADSP-21xx assembly language is created by an editor that produces plain text files [.DSP].
2. The assembly language code is translated by the ADSP-2100 Family Assembler into object code.
3. The ADSP-21xx Linker generates an executable program [.exe] by linking together separately-assembled modules.
4. The executable program is run at debug state on ADMC331 board by using the software 'ADMC331 Motion Control Debugger' of Analog Devices Inc.
5. Modify the source program and repeat steps (2) to (4) until the desired output is obtained.

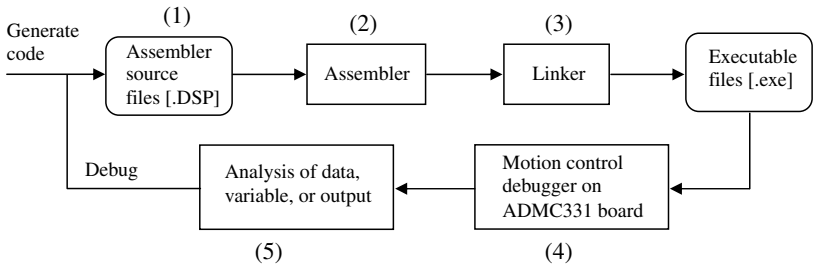


Figure 11.4 Software development procedure on ADMC331.

Each DSP program is organized into a number of source code files (with .DSP extension) each handling a separate function. Each DSP source code file (except the main program) has an accompanying header file (.H extension) which is used to declare functions and variables located within its associated .DSP file. A program structure using multiple files and header files results in minimal coupling between blocks.

- A 'program' is one that can run independently. Its source file has the extension .DSP or .C, for example, main program MAIN.DSP or ACQUIRE.C.

- A 'routine' may be called by a 'program'. It has the extension .DSP, for example, CAPTURE.DSP.
- A 'subroutine' is not an independent DSP program, but only a program segment located in a 'program' or a 'routine' and called only by a 'program' or a 'routine'. In ADSP-21xx assembly language structure, a 'subroutine' begins from a label and ends with 'RTS'.

In program debugging, one of the following three methods may be employed to monitor the data, variable, and input/output of the developed program:

1. Multimeter: Certain steady-state values in the running program may be observed at the DAC output ports on the ADMC connector board by a digital multimeter.
2. Reading the data memory on the DSP: ADMC331 DSP has a 1k x 16-bit data memory (RAM with address form 0x3C00 to 0x3FFF) which can store successive data values to aid in debugging the code and tuning the controller parameters. The stored data values may then be examined by displaying them with Motion Control Debugger's plot memory function, or by dumping them to a file on the PC using a dump memory command. From a memory dump file, the data can be further analyzed. Due to limitation of the data memory size on the DSP, the number of captured data is less than 1000.
3. Acquiring data by PCL818HG: A Visual C++ program is designed to implement the data acquisition from the DAC to the PC. The eight-way signals can be acquired at a sampling rate up to 100 kHz.

11.4 Experiment 1: Determination of Motor Parameters

Tests were performed to determine the electrical parameters of the 147-W induction motor manufactured by Bodine Electric Company. These parameters are required in the simulation studies as well as in tuning of the PI controllers. The details are given in Appendix H.

11.5 Experiment 2: Induction Motor Run Up

In order to verify the voltage-input model, sinusoidal PWM model, encoder model, and decoder model built in Chapter 3, the ADMC331 pulse-width-modulator (PWM switching frequency of 10 kHz) block is used to generate a 3-phase, 60-Hz supply to run the motor up to its base speed of 1800 r/min. The encoder GBZ02 is used to measure the angular speed of the motor, while the data-acquisition board PCL818HG is used to acquire signals of the angular speed into the data-acquisition PC from the DAC port on the ADMC connector board.

The basic programs designed for this experiment are: (1) main program MAIN.DSP, (2) routine PWM331.DSP (ADSP-21xx assembly language), to produce three PWM signals, (2) routine DAC.DSP, to write the reference phase voltages to the AD7568 digital to analog converter (DAC), (3) routine ENCO.DSP, to detect pulses from the encoder, and (4) PC program ACQUIRE.C (C++ language), to acquire the signals of voltage and angular speed from the DAC, as well as the encoder output signal to the data-acquisition PC for observation and analysis.

File Name	Description
1. MAIN.DSP:	DSP source code for main program (listed in Appendix I).
2. pwm331.dsp:	pulse width modulator initialization and use routines.
3. pwm331.h:	header file containing definitions for PWM331.DSP.
4. ENCODER.DSP:	calculate shaft speed by reading the signal from an encoder.
5. DAC.DSP:	writing to the AD7568 digital to analog converter (DAC) on the ADMC connector board. The DAC is a 12-bit, 8-channel device, and is accessed using the SPORT0 serial port on the DSP.
6. DAC.H:	header file containing definitions for DAC.DSP.
7. build.bat:	batch file to convert the DSP source code into an executable file, **main.exe** which can be downloaded and run on the DSP.
8. ACQUIRE.C:	C++ source code for acquiring phase voltage signals and encoder output signal to the PC. This is built into an executable file **acquire.exe** to run on the data-acquisition PC with the data-acquisition card PCL818HG.
9. ACQUIRE.H:	header file containing definitions for ACQUIRE.C.

11.5.1 Program Design

11.5.1.1 PWM Algorithm

The algorithm produces three PWM signals whose fundamental outputs may be described in normalized variables as:

$$V_{refA} = \kappa \sin(\theta) \tag{11.1}$$

$$V_{refB} = \kappa \sin(\theta + 2\pi/3) \tag{11.2}$$

$$V_{refC} = \kappa \sin(\theta + 4\pi/3) \tag{11.3}$$

where $\kappa \in [0,1]$ is the amplitude and frequency scale factor. The angle θ is calculated incrementally for each PWM cycle as:

$$\begin{aligned} \theta(n) &= \theta(n-1) + 2\pi\kappa f_{\max} T_s \\ &= \theta(n-1) + \kappa\Delta \end{aligned} \tag{11.4}$$

where T_s is the PWM switching period and f_{max} is the maximum fundamental frequency at $\kappa = 1$. For an assumed maximum frequency f_{max} of 100 Hz and a PWM switching frequency of 10 kHz (i.e., $T_s = 0.0001$s), $\Delta = 0.0628$ rad.

A suitable sine approximating function (Analog Devices, 1992) is given by:

$$\sin(x) = 3.140625x + 0.020264x^2 - 5.325196x^3 + 0.544678x^4 + 1.800293x^5 \tag{11.5}$$

The sine function is implemented by calling a subroutine ADMC_SIN in ADMC331 ROM. Having calculated the sine functions and scaled by κ to produce the reference voltages, the

PWM on-times may be calculated as:

$$T_A = \frac{T_s}{2} + \frac{T_s}{2} V_{refA}. \tag{11.6}$$

Similar equations can be written for the other two phases.

11.5.1.2 Estimating the Actual Stator Frame Voltages

In order to estimate the actual 3-phase voltages (V_{ac_a}, V_{ac_b}, V_{ac_c}) applied to the motor, the VDC_HANDLING subroutine in MAIN.DSP uses the DC bus voltage value to perform the following calculations:

$$V_{ac\_a} = V_a \cdot \frac{V_{dc\_measured}}{V_{dc\_\max}} \tag{11.7}$$

$$V_{ac\_b} = V_b \cdot \frac{V_{dc\_measured}}{V_{dc\_\max}} \tag{11.8}$$

$$V_{ac\_c} = V_c \cdot \frac{V_{dc\_measured}}{V_{dc\_\max}}. \tag{11.9}$$

V_a, V_b, and V_c are the requested voltages, but if the DC bus voltage is less than the maximum value, such as when the motor is heavily loaded, the applied voltages will be reduced.

11.5.1.3 Digital to Analog Converter

The routine DAC.DSP writes digital values via the AR register to the appropriate location in a buffer. The resulting polarity is then correct at the DAC screw terminals on the ADMC connector board. Table 11.2 gives some examples of digital values used as parameters to the DAC subroutines and the resulting analog voltages produced.

When a stator phase-voltage is 0 V in an algorithm and its digital value in the DSP program is defined as 0x0000 or 0xFFFF, the analog voltage output from the DAC is 2.5 V, according to Table 11.2. When the stator phase-voltage magnitude is 188 V in the algorithm and its digital value in the DSP program is defined as 0x3FFF, the analog voltage output from the DAC is 3.75 V.

Table 11.2 DAC output voltages for various digital inputs.

Digital Value	Analog Voltage
0x8000	0 V
0xBFFF	1.25 V
0xFFFF	2.5 V
0x0000	2.5 V
0x3FFF	3.75 V
0x7FFF	5 V

11.5.1.4 Encoder Functions

The routine ENCO.DSP is used to detect pulses from the encoder, and a programmable timer on ADMC331 enables the motor speed to be determined. The file contains three subroutines: (1) Init_encoder subroutine: it is used to initialize the encoder subsystem; (2) Timer_isr (interrupt service routine) subroutine: on each timer interrupt, the counter variable is incremented. ALU saturation mode is enabled so that the counter value will only go as high as 0x7FFF. (3) Pio_isr subroutine handles changes detected on the encoder PIO line with a rising edge interrupt.

11.5.1.5 Data Acquisition

A C++ code file ACQUIRE.C is written to acquire stator voltages and encoder signals via the PCL818HG card to the data-acquisition PC using the direct memory access (DMA) method. DMA is a fast data-transfer method by allowing external devices to transfer data directly to the PC memory without involving the interrupt commands of the system CPU. An executable program ACQUIRE.EXE is produced from the C++ code ACQUIRE.C and is run in the data-acquisition PC.

11.5.2 Program Debug

The program debug procedure comprises the following steps:

1. Disconnect the inverter PowIRtrain from the DSP ADMC331.
2. Produce an executable program **main.exe** using the batch file BUILD.BAT.
3. Invoke the 'Motion Control Debugger' program on the host PC.
4. Download the **main.exe** program from the host PC to DSP.
5. Click the run button on the host PC.
6. Observe and analyze the data, variable, and input/output values, by a digital multimeter, by reading the data memory of ADMC331, or on the data-acquisition PC with the data-acquisition card PCL818HG.
7. Modify the source code of assembly language and iteratively repeat steps (2) to (7), until satisfactory voltage waveforms are obtained.
8. Connect the inverter PowIRtrain to the DSP ADMC331 and replace the induction motor by a larger resistor load (three 5K resistors with 'Y' connection). The inverter output terminals are connected to the voltage sensor (model 3V411A), so that the output line voltages, V_{ab} and V_{ac} can be transferred to the data-acquisition PC via the data-acquisition card PCL818HG.
9. Download the **main.exe** program from the host PC to DSP.
10. Run the **acquire.exe** program on the data-acquisition PC.
11. Click the run button on the host PC and click the start button on the data-acquisition PC.
12. Observe the inverter output line voltages on the data-acquisition PC. When the output of the inverter is satisfactory, program debug is complete and the PowIRtrain may be connected to drive the induction motor.

11.5.2.1 Results of Debug At Step (6)

Figure 11.5 shows the voltage waveforms V_a, V_b, and V_c (requested three phase voltages) which are directly acquired by the data-acquisition card PCL818HG via the DAC on the ADMC connector board.

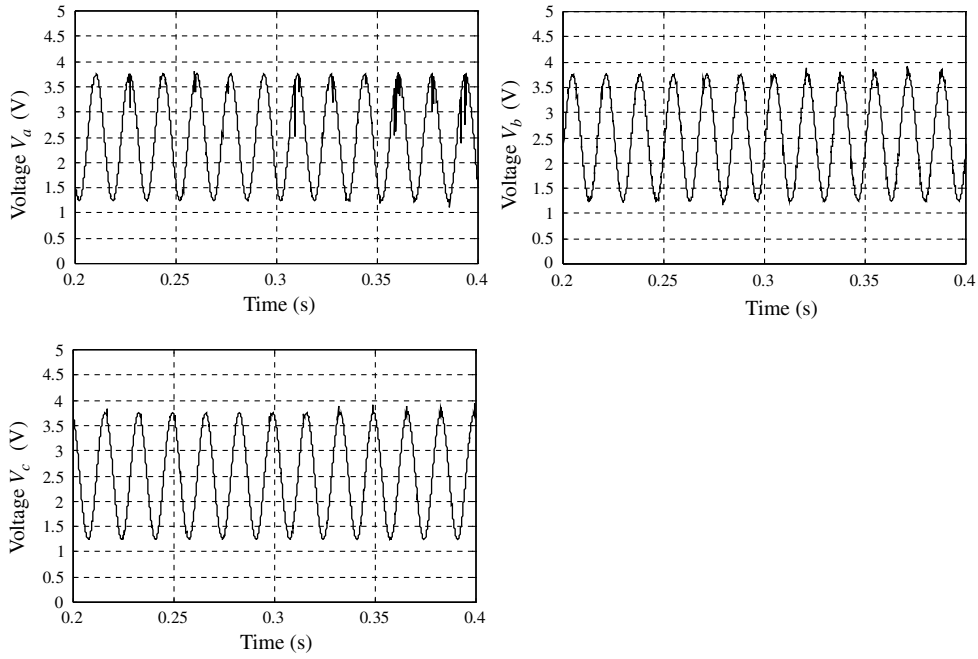


Figure 11.5 Requested three phase voltages V_a, V_b, and V_c output directly from DAC.

In Figure 11.5, the requested phase-voltage magnitudes from DAC are 1.25 V (= 3.75 V − 2.5 V). From Table 11.2, the requested phase-voltage magnitude should be 188 V in the control algorithm and its digital value in the DSP program should be defined as 0x3FFF.

Figure 11.6 shows the output waveforms V_{ah}, V_{al}, V_{bh}, and V_{ch} (three high-side PWM signals and a low-side PWM signal) which are directly acquired by the data-acquisition card PCL818HG from terminals J4 on the ADMC connector board at a sampling rate of 35 kHz. Because the signal values of PWM output are smaller (about 50 mV), they have to be amplified by the PCL818HG card before A/D transformation. With an amplification factor of 10, the maximum sampling rate of the PCL818HG card is limited to 35 kHz. If the four signals V_{ah}, V_{al}, V_{bh}, and V_{cb}, are to be acquired simultaneously, the sampling rate of each signal is 8.75 kHz (35/4 kHz), which is much lower than the minimum of 20 kHz (for 10 kHz PWM switching frequency). Hence, the acquired signals in Figures 11.6 and 11.7 have larger distortions and very narrow switching pulses cannot be shown. Nevertheless, it is confirmed that the three PWM signals produce a balanced three-phase system at the desired output frequency of 60 Hz.

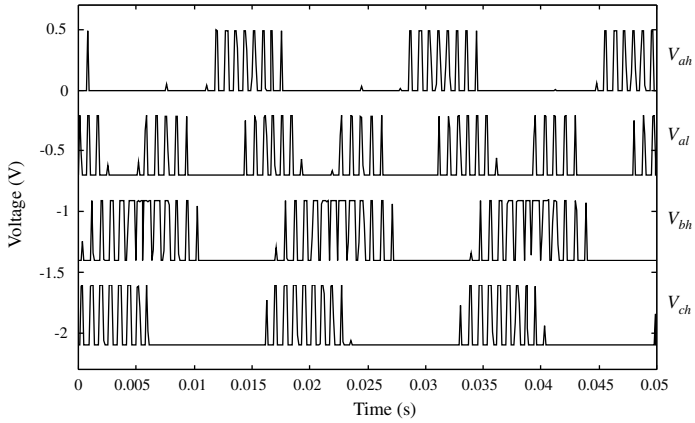


Figure 11.6 Output waveforms V_{ah}, V_{al}, V_{bh}, and V_{ch} (three high-side PWM signals and a low-side PWM signal) acquired at a sampling rate of 8.75 kHz at debug **Step (6)**.

Figure 11.7 shows the waveform of the difference of the two PWM outputs V_{ah} and V_{bh}. The line-voltage modulation process may be clearly seen. Comparing Figure 11.7 with Figure 3.31, the experimental PWM waveform verified that the PWM Simulink model built in Chapter 3 is viable.

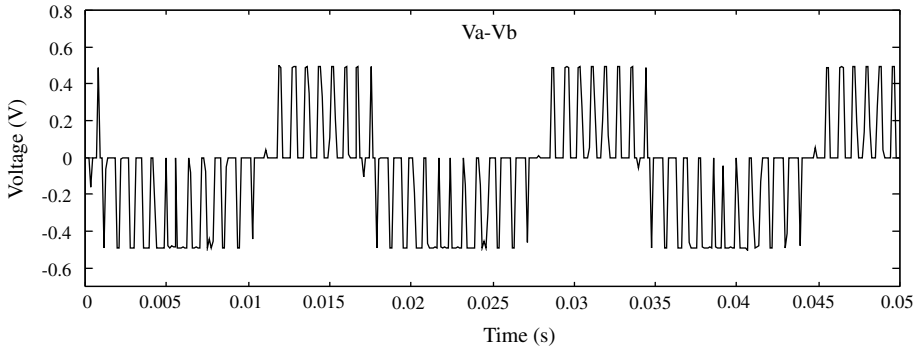


Figure 11.7 PWM output waveform of $V_{ah} - V_{bh}$ acquired at a sampling rate of 8.75 kHz.

11.5.2.2 Results of Debug at Step (11)

At debug **Step (11)**, the inverter PowIRtrain output line voltage is measured by the voltage sensor (model 3V411A) with gain 0.01 and transferred to the data-acquisition PC via the data-acquisition card PCL818HG at 20 kHz sampling rate. Figure 11.8 shows the line voltage V_{ab} at the inverter PowIRtrain output terminals. Because the voltage sensor gain is 0.01, the actual magnitude of the line voltage in Figure 11.8 is 325 V. The waveform suggests that the inverter is operating at the desired output frequency of 60 Hz.

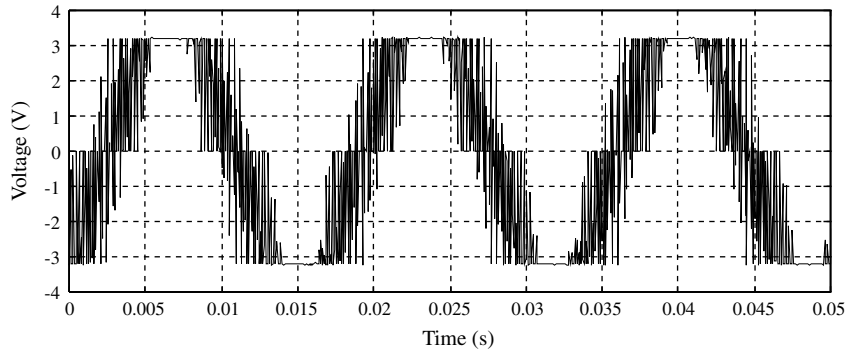


Figure 11.8 Output line voltage V_{ab} of the inverter PowIRtrain acquired at a sampling rate of 20 kHz.

After satisfactory debug results have been obtained, the PWM control program may be used to drive the induction motor.

11.5.3 Experimental Investigations

The experimental investigations start with the following steps:

1. Invoke the 'Motion Control Debugger' program on the host PC.
2. Download the **main.exe** program from the host PC to DSP.
3. Run the **acquire.exe** program on the data-acquisition PC.
4. Click the run button on the host PC.
5. Click the start button on the data-acquisition PC.

When the executable program **main.exe** is downloaded into DSP, it is converted into a binary file and located in ADMC331 program memory at address from 0x0030 to 0x010D (occupying 222 × 12 bit space), while the interrupt vector table is stored at address from 0x0000 to 0x002F. Due to the unavailability of the ADSP-2100 Family Simulators software, program execution time cannot be estimated.

Test Results

The rotor speed is calculated by calling the ENCO.DSP routine which reads the signal from the encoder. Then, the rotor speed signal and signals of 3-phase voltages (V_{ac_a}, V_{ac_b}, V_{ac_c}) are acquired via DAC at a sampling rate of 40 kHz (sampling rate of each signal is 10 kHz) by the **acquire.exe** program and are displayed on the data-acquisition PC. The 3-phase voltages (V_{ac_a}, V_{ac_b}, V_{ac_c}) have been estimated by the VDC_HANDLING subroutine on the DSP as described in Equations (11.7), (11.8), and (11.9).

Figure 11.9 shows the estimated 3-phase voltages which are applied to the induction motor. According to Table 11.2, the analog values of the stator voltages shown in Figure 11.9 have been modified by:

$$V_{estimated} = \frac{(V_{acquire} - 2.5) \times 188}{1.25} (V). \tag{11.10}$$

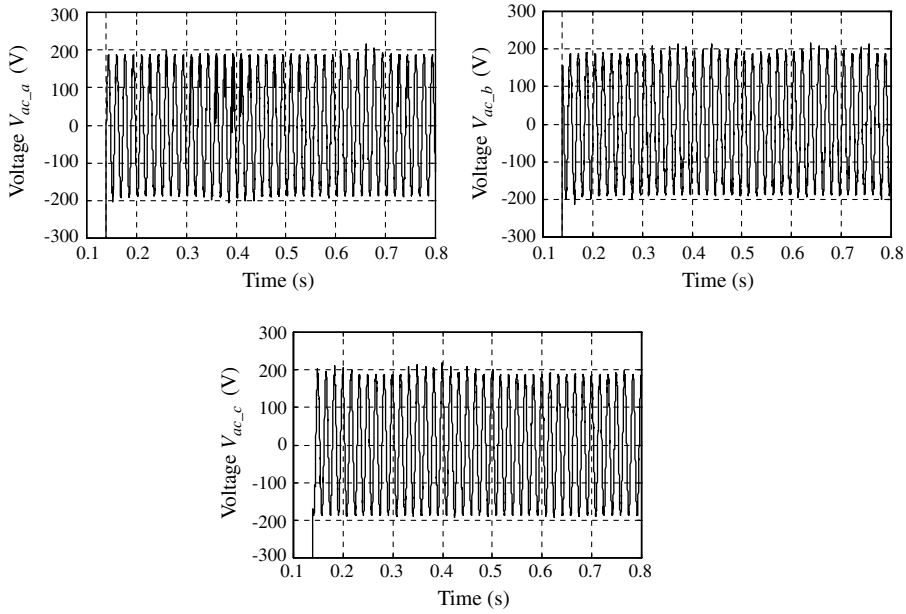


Figure 11.9 Estimated 3-phase voltages (V_{ac_a}, V_{ac_b}, V_{ac_c}).

where $V_{acquire}$ represents the voltage signal acquired by PCL818HG card from DAC on the ADMC connector board, while $V_{estimated}$ represents the estimated value of the stator phase voltage.

Figure 11.10 shows the rotor speed response (time = 0 s ~ 0.8 s) calculated by the routine ENCO.DSP and acquired by the data-acquisition card PCL818HG via the DAC port on the ADMC Connector Board. At steady state, the motor runs at a speed of about 187 rad/s, confirming that the output frequency of the inverter is 60 Hz.

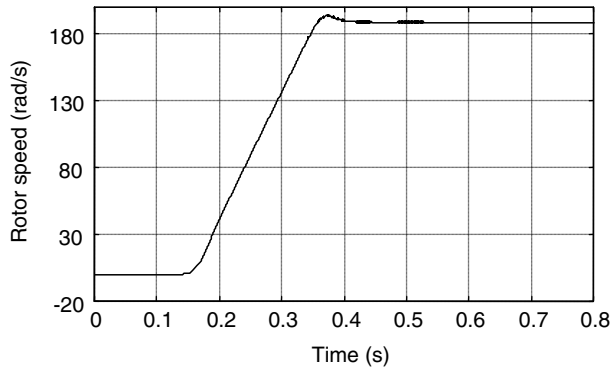


Figure 11.10 Rotor speed response (time = 0 s ~ 0.8 s).

11.5.3.1 Verifying the Encoder and Decoder Models

Figure 11.11 shows the encoder output signal which is acquired at a sampling rate of 10 kHz, while Figure 11.12 shows the rotor speed calculated from the encoder output signal, using the decoder model shown in Figure 3.35 of Chapter 3.

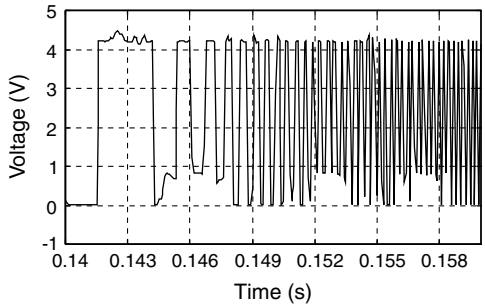


Figure 11.11 Encoder output signal acquired at a sampling rate of 10 kHz.

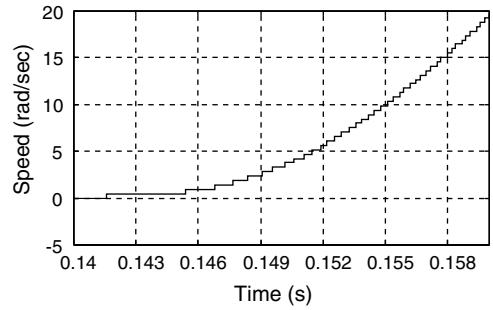


Figure 11.12 Actual speed response calculated by the decoder model.

Figure 11.13 shows the encoder output signal which is acquired at a sampling rate of 20 kHz while Figure 11.14 shows the speed response calculated using the decoder model.

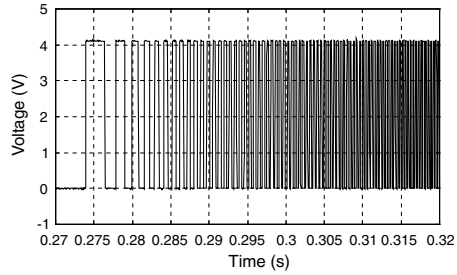


Figure 11.13 Encoder output signal acquired at a sampling rate of 20 kHz.

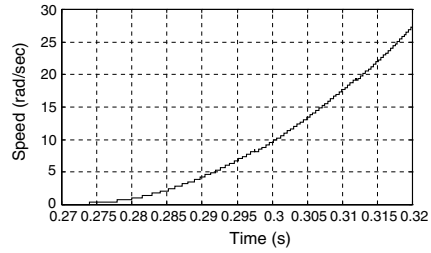


Figure 11.14 Actual speed response calculated by the decoder model.

The simulation models of the PWM inverter, encoder, and decoder (which have been built in Chapter 3) may be compared with the experimental results, for example, Figure 3.34 versus Figures 11.11,11.13 and 3.40 versus Figures 11.12 and 11.14. Although a number of uncertain factors in the experimental system (such as the load, motor parameter changes, calculation tolerance, and noise) have not been accounted for, the simulation results of the encoder and decoder in Chapter 3 are approximately the same as the experimental results. The encoder and decoder models developed in Chapter 3 are thus verified.

11.6 Experiment 3: Implementation of Fuzzy/PI Two-Stage Controller

This experiment is designed to verify the algorithm of the fuzzy/PI two-stage controller which has been described in Section 6.5 in Chapter 6. Source codes of the fuzzy/PI two-stage controller are written by ADSP-21xx assembly language. The hardware system has been described in Figure 11.1 and the software development method has been described in Section 11.3. The contents of the .DSP and .H files of the DSP-based fuzzy/PI controller are summarized as follows.

File Name	Description
MAIN.DSP	Induction machine control main program
FUZZY.DSP, FUZZY.H	Fuzzy frequency controller
CURRENTMC.DSP CURRENTMC.H	Current magnitude PI controller
STCPI.DSP, STCPI.H	Stator-current PI controller
ADC331.DSP, ADC331.H	Analog to digital converter of stator currents
DAC.DSP, DAC.H	Writing to DAC on ADMC connector board
DIVIDE.DSP, DIVIDE.H	Divide function
FPMATH.DSP, FPMATH.H	Floating point math routines
MATH_32B.DSP, MATH_32B.H	32 bit math routines
PWM331.DSP, PWM331.H	PWM functions
ROMUTIL.H	ROM utility definitions
ENCO.DSP, ENCO.H	Encoder functions
SPEEDSET.DSP, SPEEDSET.H	Speed setpoint function
CONST331.H	ADMC331-related constants
CAPTURE.DSP, CAPTURE.H	Capturing variables to data memory of ADMC331
build.bat:	Batch file to build the DSP source code into an executable file main.exe which can be downloaded and run on the DSP.
ACQUIRE.C, ACQUIRE.H:	C++ source codes for acquiring signals to PC

11.6.1 Program Design

11.6.1.1 Control Main Program (MAIN.DSP)

The main program is MAIN.DSP (listed in Appendix J), and the file CONST331.H has constant definitions. The file MAIN.DSP contains the initialization code, the main loop, and top level software architecture. The initialization code makes calls to all of the various subroutines which initialize a given function such as the encoder, DAC, and so on.

Design details of the fuzzy frequency controller, PI current magnitude controller, and stator current controllers are described as follows.

11.6.1.2 Fuzzy Slip Frequency Controller (FUZZY.DSP)

Source code of the fuzzy frequency controller FUZZY.DSP contains (1) subroutine of fuzzification operation of speed command, (2) subroutine of fuzzification operation of error

between speed command and actual speed, (3) subroutine of fuzzy inference operation, INFERE.DSP, and (4) subroutine of defuzzification operation.

Fuzzification Subroutine of Speed Command

The subroutine of fuzzification operation of speed command converts the crisp input values of speed command ω_o^* to fuzzy sets. Fuzzy conversion involves the calculation of triangular degrees of membership and calculation of linguistic values. With reference to Figure 6.32 (Membership function of speed command), the degrees of membership and linguistic values can be calculated.

The six constants of fuzzification operation of speed command in Figure 11.15 are defined in Table 11.3, while Table 11.4 summarizes the fuzzification operations of the speed command.

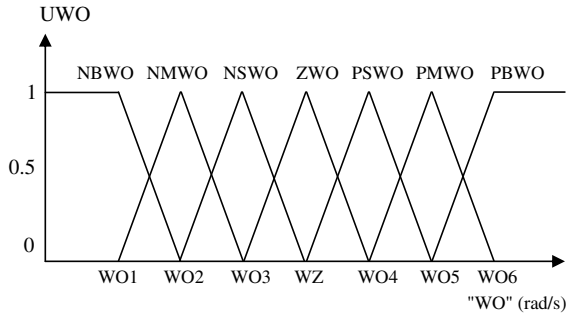


Figure 11.15 Membership function of speed command. NBWO: negative big; NMWO: negative medium; NSWO: negative small; ZWO: zero; PBWO: positive big; PMWO: positive medium; PSWO: positive small.

Table 11.3 Fuzzification operation constants of speed command.

WO1	WO2	WO3	WZ	WO4	WO5	WO6
– 188	– 126	– 63	0	63	126	188

The subroutine segment (written in ADSP-21xx assembly language) to implement the fuzzification operation of speed command is listed as follows.

```
FUZW:                                {subroutine begin}

FUZA:                                {label}
AR = WOC-WO1;                        {AR: ALU result register}
IF GE JUMP FUZB;                     {Program flow controlled by ALU result}
UWO1 = 1;                            { Degrees of membership}
UWO2 = 1;
lingu1= NBWO;                        { Linguistic value}
lingu2= NBWO;
JUMP FUZEND                          {Jump to label FUZEND}
```

Table 11.4 Fuzzification operations of speed command.

Label	Condition	Degrees of membership	Linguistic value
FUZA	WOC *LT* WO1	UWO1 = 1	NBWO
		UWO2 = 1	NBWO
FUZB	WOC *GE* WO1 AND WOC *LT* WO2	UWO1 = (WO2-WOC)/(WO2-WO1)	NBWO
		UWO2 = (WOC-WO1)/(WO2-WO1)	NMWO
FUZC	WOC *GE* WO2 AND WOC *LT* WO3	UWO1 = (WO3-WOC)/(WO3-WO2)	NMWO
		UWO2 = (WOC-WO2)/(WO3-WO2)	NSWO
FUZD	WOC *GE* WO3 AND WOC *LT* WZ	UWO1 = (WZ-WOC)/(WZ-WO3)	NSWO
		UWO2 = (WOC-WO3)/(WZ-WO3)	ZWO
FUZE	WOC *GE* WZ AND WOC *LT* WO4	UWO1 = (WO4-WOC)/(WO4-WZ)	ZWO
		UWO2 = (WOC-WZ)/(WO4-WZ)	PSWO
FUZF	WOC *GE* WO4 AND WOC *LT* WO5	UWO1 = (WO5-WOC)/(WO5-WO4)	PSWO
		UWO2 = (WOC-WO4)/(WO5-WO4)	PMWO
FUZG	WOC *GE* WO5 AND WOC *LT* WO6	UWO1 = (WO6-WOC)/(WO6-WO5)	PMWO
		UWO2 = (WOC-WO5)/(WO6-WO5)	PBWO
FUZH	WOC GE WO6	UWO1 = 1	PBWO
		UWO2 = 1	PBWO

WOC: Speed command; *LT*: Less than (ADSP-2100 assembler expression); *GT*: Greater than (ADSP-2100 assembler expression); *GE*: Greater than or equal (ADSP-2100 assembler expression); /: divide (ADSP-2100 assembler expression); -: subtraction (ADSP-2100 assembler expression).

```
FUZB:
AR = WOC-WO2;
IF GE JUMP FUZC;
UWO1 = (WO2-WOC) / (WO2-WO1);
UWO2 = ( WOC-WO1) / (WO2-WO1)
lingu1= NBWO;
lingu2= NMWO;
JUMP FUZEND

FUZC:
.
.
.
FUZEND:
RTS;                                    { Subroutine ends here }
```

Fuzzification Subroutine of Speed Error

Speed error 'WE' is the difference between rotor speed 'WO' and speed command 'WOC', which can be expressed as:

$$WE = WO - WOC$$

where WO is obtained from the output of the encoder program, ENCO, and WOC is the speed set value.

The subroutine of fuzzification operation of speed error converts the crisp input values of speed error $\Delta\omega$ to fuzzy sets. Table 11.5 defines the five constants of fuzzification operation of speed error and the fuzzification operations of speed error are summarized in Table 11.6.

Table 11.5 Constants of fuzzification operation of speed error.

E1	E2	E3	E4	E5
− 376	− 3	0	3	376

Table 11.6 Fuzzification operations of speed error.

Label	Condition	Degrees of membership	Linguistic value
FUZI	WE LT E1	UWO3 = 1	NDWO
		UWO4 = 1	NDWO
FUZJ	WE GE E1 AND WE LT E2	UWO3 = (E3-WE)/(E3-E1)	NDWO
		UWO4 = 0	ZDWO
FUZK	WE GE E2 AND WE LT E3	UWO3 = (E3-WE)/(E3-E1)	NDWO
		UWO4 = (WE-E2)/(E3-E2)	ZDWO
FUZD	WE GE E3 AND WE LT E4	UWO3 = (W4-WE)/(E4-E3)	ZDWO
		UWO4 = (WE-E3)/(E5-E3)	PDWO
FUZE	WE GE E4 AND WE LT E5	UWO3 = 0	ZWO
		UWO4 = (WE-E3)/(E5-E3)	PDWO
FUZF	WE GE E5	UWO3 = 1	PDWO
		UWO4 = 1	PDWO

Subroutine of Fuzzy Inference Operation

The subroutine of fuzzy inference operation implements the linguistic inference of Equations (6.16) and (6.18), and the degree of membership calculation of Equations (6.19) and (6.20). The details of the fuzzy inference operations are summarized in Table 11.7.

Table 11.7 Fuzzy inference operations.

Variable names	Description
Inputs	
UWO1, UWO2, UWO3, UWO4	Degrees of membership of speed command and speed error
Lingu1, lingu2, lingu3, lingu4,	Linguistic values of speed command and speed error
Outputs	
UC1, UC2	Crisp values of fuzzy linguistic
UD1, UD2	Degrees of membership

Subroutine of Defuzzification Operation

The subroutine of defuzzification operation implements the defuzzification for outputs of the fuzzy inference. The centroid method given by Equation (6.21) is used to implement the defuzzification operation. The details of the defuzzification operations are summarized in Table 11.8.

Table 11.8 Defuzzification operations.

Variable names	Description
Inputs	
UC1, UC2	Crisp values of fuzzy linguistic
UD1, UD2	Degrees of membership
Outputs	
WEC	Numerical value of slip frequency

11.6.1.3 PI Current Magnitude Controller (CURRENTMC.DSP)

When the permissible magnitude of stator current of the induction motor is 2 A, proportional and integral parameters of the PI current magnitude controller are designed, from Equations (6.23)–(6.25), as $K_p = 0.7$ and $K_I = 0.014$. The discrete time difference equations which implement the PI controller are as follows:

$$I(k+1) = I(k) + K_I \cdot e(k+1) \tag{11.11}$$

$$U\{k+1) = K_p \cdot e(k+1) + I(k+1) \tag{11.12}$$

where: $K_I =$ integral constant
$I(k+1) =$ integral of error x integral constant
$I(k) =$ previous integral of error x integral constant
$e(k+1) =$ error for iteration $k+1$
$K_p =$ proportional constant
$U(k+1) =$ controller output

Equations (11.11) and (11.12) are implemented by a routine of PI current magnitude controller, CURRENTMC.DSP.

11.6.1.4 Generate Stator Current Command (Subroutine SINCURRENT)

The slip frequency command WEC of the fuzzy controller output, the current magnitude command U of the output of current magnitude PI controller, and the speed command WO are converted into three stator current commands, IAC, IBC, and ICC by a subroutine

SINCURRENT in the main program MAIN.DSP. Algorithm of the subroutine SINCURRENT is described as follows:

{initialize phase angles}

$$Wa(0) = 0;$$

$$Wb(0) = 2\pi/3;$$

$$Wc(0) = -2\pi/3;$$

{calculate supply frequency}

$$W\_\text{com} = \text{WO} + \text{WEC};$$

{increment phase angles}

$$Wa(n+1) = Wa(n) + W\_\text{com} \times \Delta T;$$

where $\Delta T = 2 \times \pi \times (1/10000)$; (PWM frequency = 10 kHz)

{calculate 3-phase current command}

$$IAC = U\sin(Wa(n+1));$$

$$IBC = U\sin(Wb(n+1));$$

$$ICC = U\sin(Wc(n+1));$$

The sine function is implemented by calling a function ADMC_SIN stored in ADMC331 ROM.

11.6.1.5 Stator Current Controllers (STCPI.DSP)

The design of the current PI controller shown in Figure 6.31 has been described in Equation (6.26)–(6.40). Because the feedback gains in the experimental system is different from the simulation system in Chapter 6, the parameters of the current PI controller must be modified as follows.

The open-loop transfer function, Equation (6.33), of the PI controller and the motor is modified as:

$$G_{openloop} = G_{PI} \times K_1 \times G_{motor} = K_p \times K_1 \times \frac{1}{\sigma L_s s} \tag{11.13}$$

where K_1 is all other gains in the systems.

If a constant K_2 is introduced as follows,

$$K_2 = K_p \times K_1 \times \frac{1}{\sigma L_s} \tag{11.14}$$

then, $G_{openloop} = \dfrac{K_2}{s}$.

Closing the feedback loop with unity gain results in the following closed-loop transfer function:

$$G_{\text{closedloop}} = \frac{K_2}{s + K_2}. \tag{11.15}$$

This is recognized as a single-pole low-pass filter with 3-dB corner frequency at:

$$F_{3dB} = \frac{K_2}{2\pi}. \tag{11.16}$$

Choosing a filter corner frequency (or a controller bandwidth) of 1 kHz results in:

$$K_2 = 2\pi \times 10^3. \tag{11.17}$$

The feedback gain term K_1 is made up of the following terms:

PWM Gain = DC bus voltage	230 V
Current sensor gain	0.825 V/A
ADC gain	0.17041
Scaling	3.4566
Total	111.770

that is, $K_1 = 111.770$

From Equation (11.14):

$$K_p = \frac{K_2}{K_1} \times \sigma L_s. \tag{11.18}$$

From Appendix H, the parameters of the experimental induction motor are: $\sigma = 0.208$, $L_s = 0.3185$ H and $R_s = 14.6\,\Omega$. From Equation (11.18), the proportional constant is

$$\begin{aligned} K_p &= 2\pi \times 10^3 \times 0.208 \times 0.3185/111.770 \\ \text{or} \quad K_p &= 3.724 \end{aligned}. \tag{11.19}$$

From Equation (6.23), the integral constant K_I is

$$K_I = \frac{3.724 \times 14.6}{0.208 \times 0.3185}$$

or

$$K_I = 821. \tag{11.20}$$

11.6.1.6 Analog to Digital Converter (ADC331.DSP)

The file ADC331.DSP contains subroutines used to read analog voltages which represent the three-phase stator currents and the DC bus voltage.

11.6.1.7 Divide Function (DIVIDE.DSP)

The file DIVIDE.DSP contains a function called Division which divides a 32-bit dividend by a 16-bit divisor, and returns a 16-bit quotient.

11.6.1.8 Floating Point Math Routines (FPMATH.DSP)

The routines in the file FPMATH.DSP implement floating point numbers in the program. In the floating point format used here, each number is represented by two 16-bit words, one for the mantissa in 1.15 format (i.e., 1 signed bit, 15 fractional bits), and the other for the exponents in 16.0 format (i.e., 16-bit binary strings). The value of the floating point number is given by:

$$Value = Mantissa \cdot 2^{Exponent}. \tag{11.21}$$

Table 11.9 gives some representations of the floating point number for the speed error between the speed command and actual speed.

Table 11.9 Floating point representation examples.

Speed error	Mantissa (dec)	Mantissa (hex)	Exponent (dec)	Exponent (hex)
− 188	− 0.734375	0xA1FF	8	0x0008
− 126	− 0.984375	0x81FF	7	0x0007
− 1.0	− 1.0	0x8000	0	0x0000
0.25	1.0	0x7FFF	− 2	0xFFFE
126	0.984375	0x7E00	7	0x0007
188	0.734375	0x5E00	8	0x0008

11.6.1.9 Speed Setpoint Function (SPEEDSET.DSP)

This DSP program implements reading of a speed setpoint from a potentiometer on the ADMC connector board to ADMC331 as the speed command of the controller. The program reads the auxiliary ADC input channel and scales the resulting value so that analog input voltages of 0.3–3.3 V produced by the potentiometer result in speed setpoints of 0 to 188 rad/s.

11.6.1.10 Capturing Signals to DSP Memory (CAPTURE.DSP)

The file CAPTURE.DSP contains routines used to aid in debugging the code and tuning the fuzzy controller parameters. They provide a means to capture 100 successive values (floating point number) for each of four variables, which is limited by the 1k 16-bit data memory of the ADMC331. The data values can then be examined by displaying them with Motion Control Debugger's (MCD) plot memory function, or by dumping them to a file using MCDs dump memory command. From a memory dump file, the data can be further analyzed.

11.6.2 Program Debug

11.6.2.1 Debug for Program FUZZY.DSP

1. Disconnect the inverter PowIRtrain from the DSP ADMC331.
2. Open the control loop by disconnecting the encoder input terminal from the DSP and set a static speed value in the program to replace the actual rotor speed.
3. Produce an executable program **main.exe** using the file BUILD.BAT.
4. Invoke the 'Motion Control Debugger' program on the host PC.
5. Download the **main.exe** program from the host PC to DSP.
6. Click the run button on the host PC.
7. Check the speed setpoint (speed command) by a digital multimeter.
8. Observe and analyze the speed error (input of the program FUZZY.DSP) and the slip frequency (output of the program FUZZY.DSP) by reading the data memory of ADMC331.
9. Modify the source code of assembly language and iteratively repeat steps (3) to (9) until satisfactory data values are obtained.

The speed setpoint is read by the program SPEEDSET.DSP from a potentiometer on the ADMC connector board to ADMC331 as a speed command of the controller. At debug **Step (7)**, the speed command value is written to DAC channel 1 and is examined by a digital multimeter. The DAC converts digital values (12 bit fixed point data) of the speed command to analog voltage values. For convenience of observation, the digital values are multiplied by 100 before they are converted to the analog voltage values. According to the DAC algorithm (Table 11.2), the DAC output voltage can be calculated by:

$$V_{out} = \frac{H_{input}}{32767} \times 2.5 + 2.5(V) \tag{11.22}$$

where 32 767 represents the hex number 0x7FFF, H_{input} is the speed command $\times$ 100 (decimal number), and V_{out} is the DAC output voltage.

Table 11.10 shows the analog voltage values of DSP output, which is observed on a digital multimeter.

After the speed command is set and checked, the speed error (input of the program FUZZY.DSP) and the slip frequency (output of the program FUZZY.DSP) are read from the data memory (DM) of ADMC331 (DM address 0x3894). The speed error and slip frequency are floating point numbers in the DSP control program, which has been captured to the

Table 11.10 Speed command observed on a digital multimeter.

Speed command (rad/s)	0	40	80	188
(fixed point)	(0x0000)	(0x0028)	(0x0050)	(0x00BC)
×100 before DAC	0	4000	8000	18800
	(0x0000)	(0x0FA0)	(0x1F40)	(0x4970)
Output value (V)	2.500	2.805	3.110	3.934
Observed value on multimeter (V)	2.50	2.80	3.10	3.90

data memory on ADMC331 by the program CAPTURE.DSP. Table 11.11 shows the slip frequencies captured at various speed setpoints and speed errors during the acceleration and steady-state stages.

Table 11.11 Speed error and the slip frequencies read from the DSP data memory.

Control stages	Stop	Acceleration	Steady	Acceleration	Steady	Acceleration	Steady
Speed setpoint	0	40	40	80	80	188	188
(fixed point)	(0x0000)	(0x0028)	(0x0028)	(0x0050)	(0x0050)	(0x00BC)	(0x00BC)
Speed error	0	40	0	80	0	188	0
(floating point)							
Captured Slip frequency	*0.026*	*18.228*	*0.730*	*18.317*	*2.272*	*18.198*	*5.678*

After the desired slip frequencies have been obtained by the program FUZZY.DSP, the debug for the fuzzy control program is complete.

11.6.2.2 Debug for Other Programs

The stator current control program STCPI.DSP has been tested alone by a demo program delivered by Amirix Inc., while the program ENCO.DSP has been tested in experiment 2 and a test result is shown in Figure 11.10. The debug method and procedure for other programs of the experimental control program are similar to those in Experiment 2 in Section 11.5. After all the programs have been debugged, the encoder is connected to the DSP ADMC331 for performing a closed-loop control experiment.

11.6.3 Performance Tests

11.6.3.1 Performance Test 1

Speed set point = 175 (rad/s).
Seven-way signals (V_a, V_b, V_c, i_a, i_b, i_c, ω_o) are acquired to the data-acquisition PC at same time.
Sampling rate = 40 kHz.
Sampling time = 1.4 s.
The test procedure is similar to that of Experiment 2 in Section 11.5.

Test Results

The induction motor is started by the fuzzy/PI controller from standstill to 175 rad/s. During the time interval 0 s ~ 0.14 s, the 3-phase stator voltage, current, and speed response are acquired by the PCL818HG card to the data-acquisition PC. Figure 11.16 shows the phase-A stator voltage estimated by the VDC_HANDLING subroutine and modified by Equation (11.10), phase-A stator current acquired from the current sensor board 3I411A, and rotor speed response acquired from the output of DAC on the ADMC connector board. In Figure 11.16, it is demonstrated that the stator current magnitude is maintained at about 2 A in the acceleration stage, which is a distinct characteristic of the two-stage fuzzy/PI controller.

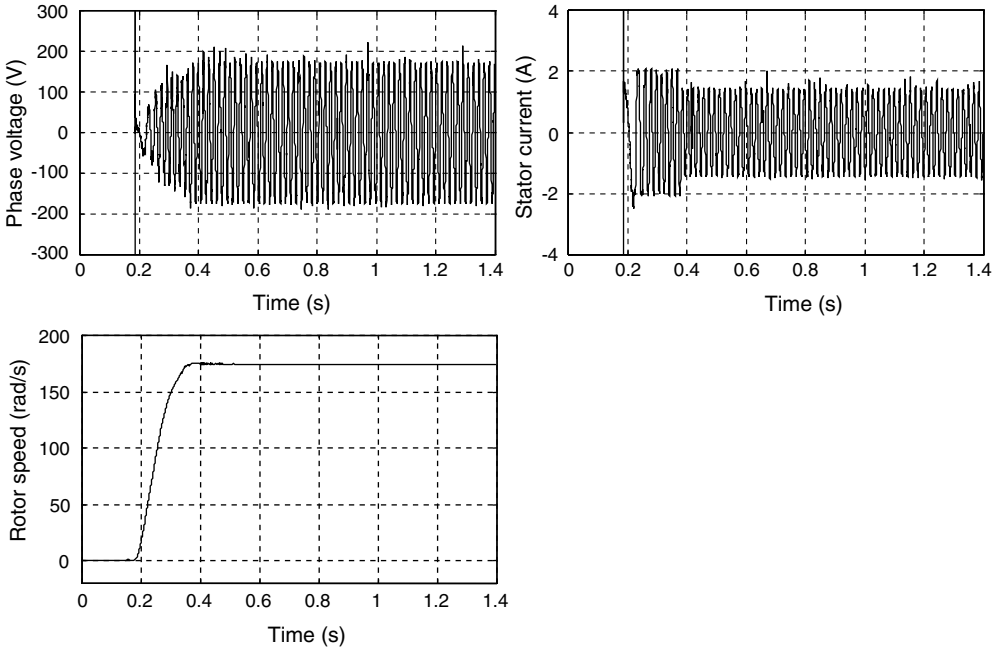


Figure 11.16 Phase-A stator voltage, phase-A stator current, and rotor speed response. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "A novel hybrid fuzzy/PI two-stage controller for an induction motor drive," *IEEE International Electric Machines and Drives Conference* (IEMDC 2001), pp. 415–421, June 17–20, 2001, Cambridge, MA, U.S.A. © 2001 IEEE.)

The 3-phase stator voltage, current, and speed response in the acceleration stage are shown in Figures 11.17–11.19 respectively.

Figure 11.18 shows that 3-phase stator current magnitudes are maintained at about 2 A in the acceleration stage.

Figure 11.19 shows the rotor speed response of the induction motor with the fuzzy/PI controller, the signal being acquired from the output of DAC on the ADMC Connector Board.

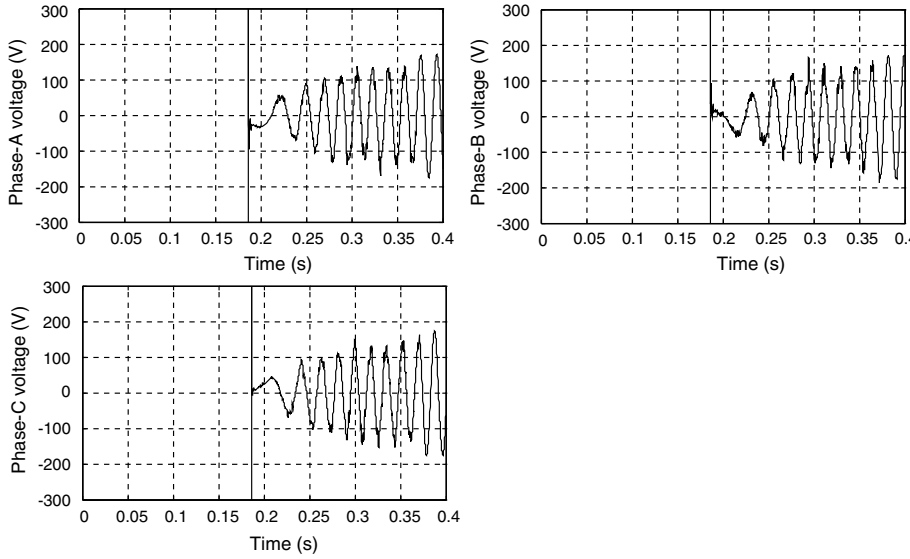


Figure 11.17 Estimated 3-phase voltages applied to the induction motor.

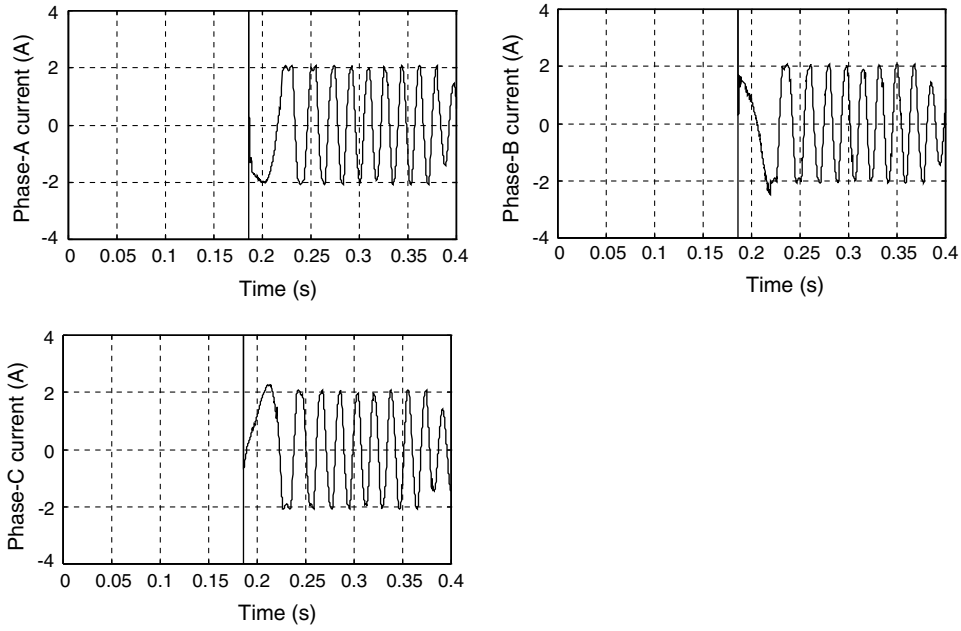


Figure 11.18 Three-phase stator currents in the acceleration stage.

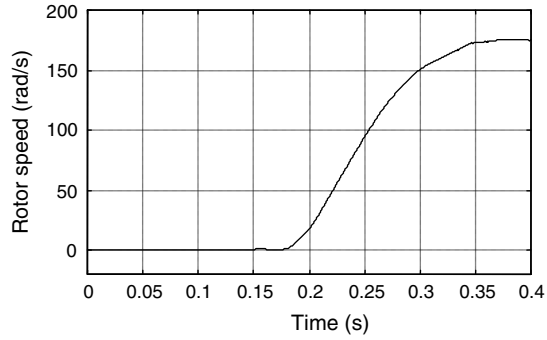


Figure 11.19 Rotor speed response in the acceleration stage.

11.6.3.2 Performance Test 2

Speed set point = 120 rad/s.
Seven-way signals (V_a, V_b, V_c, i_a, i_b, i_c, ω_o) are captured to the data-acquisition PC at same time.
Sampling rate = 40 kHz.
Sampling time = 1.4 s.
The test procedure is similar to that of Experiment 2 in Section 11.5.

Test Results

The induction motor is started by the fuzzy/PI controller from standstill to 120 rad/s. Figure 11.20 shows the 3-phase stator voltage, current, and speed response acquired by the PCL818HG card to the data-acquisition PC.

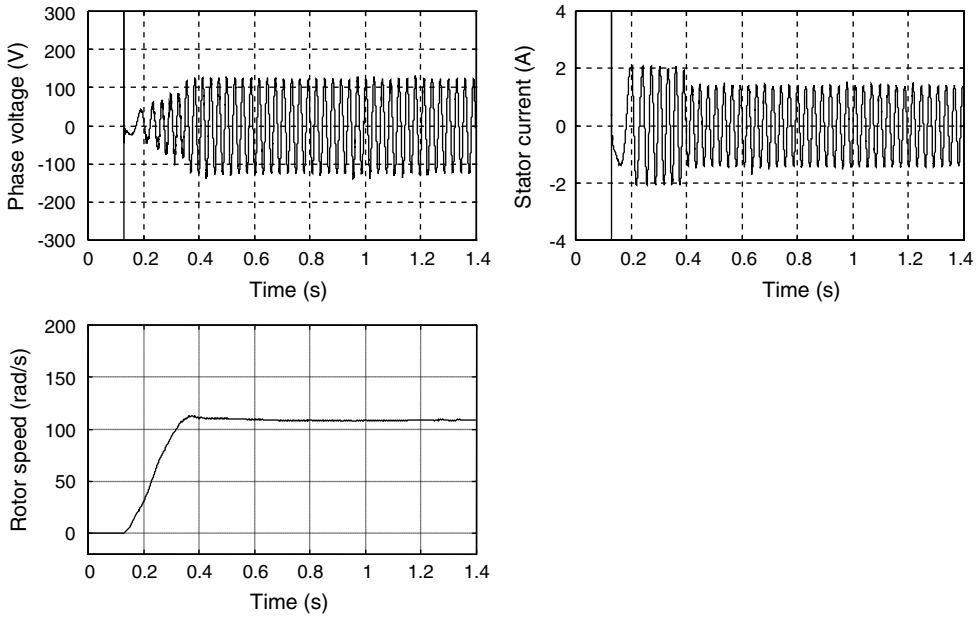


Figure 11.20 Phase-A stator voltage, phase-A stator current, and rotor speed response.

The variations of the 3-phase stator voltages and currents in the acceleration stage are shown in Figure 11.21, while the rotor speed response in the acceleration stage is shown in Figure 11.22.

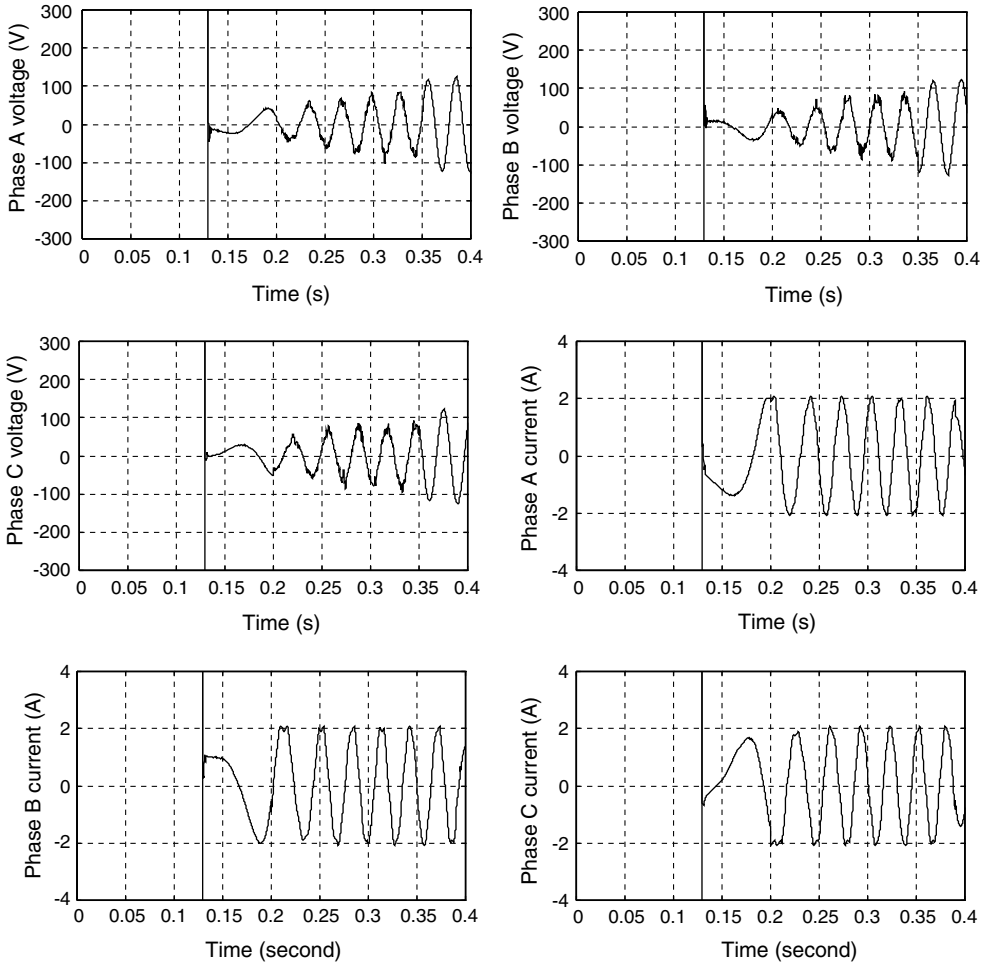


Figure 11.21 Three-phase stator voltages and currents in the acceleration stage.

When real-time data are acquired to the PC memory with the DMA method, the data size is limited to 32 000 by the actual memory assigned by the PC hardware manager. Hence, to observe and record the performance of the fuzzy/PI controlled drive over a longer period, a powerful digital scope is required.

Experiment 3 has basically verified the feasibility of the Fuzzy/PI two-stage controller proposed in Chapter 5. Better performance, however, will necessitate further simulation studies for improving and tuning the Fuzzy/PI control algorithm, for example, by using more advanced

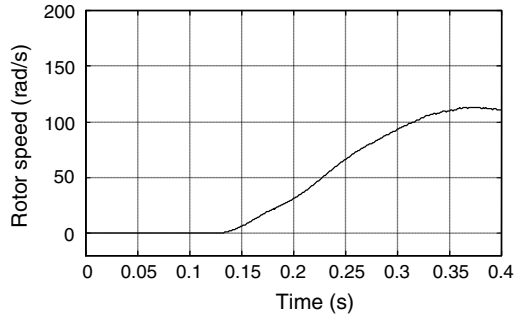


Figure 11.22 Rotor speed response in the acceleration stage.

methods (such as ANN-Fuzzy or GA-Fuzzy) to optimize the membership functions instead of the classical fuzzy method.

11.7 Experiment 4: Speed Estimation Using a GA-Optimized Extended Kalman Filter

This experiment is designed to verify the efficacy of the real-coded genetic algorithm for optimizing the extended Kalman filter which has been proposed and simulated in Chapter 7. Figure 11.23 shows the block diagram of the experimental system of the GA-optimized extended Kalman filter for rotor speed estimation.

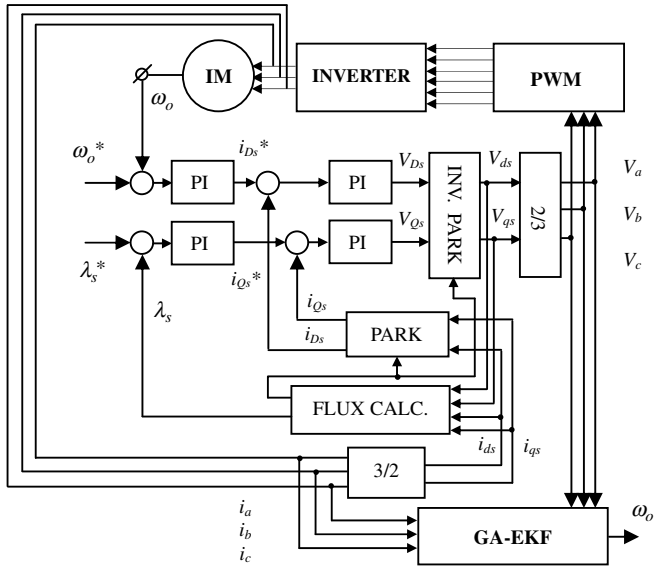


Figure 11.23 Block diagram of experimental EKF drive system.

11.7.1 Program Design

The GA-EKF experimental software includes a DSP software to drive an induction motor using the Field Oriented Control (FOC) technique, a PC C++ software to acquire actual signals of 3-phase stator voltages, currents, and encoder, and a MATLAB® software to optimize EKF by using GA and EKF speed estimation. The basic principle of the FOC technique is to control the torque-producing and flux-producing components of the motor current independently of each another. The FOC algorithm and computer simulation have been described in Section 7.7 of Chapter 7. The C ++ data-acquisition software has been designed in Section 11.5, and the MATLAB® GA software has been designed in Chapter 7. The MATLAB® EKF program is given in Appendix F. Following is the list of program units and the functions that they perform:

File Name	Description
ACIM.DSP	Induction machine controller main program
ADC331.DSP, ADC331.H	ADC routines
CAPTURE.DSP, CAPTURE.H	Debugging functions for capturing variables to memory
CNTRL.DSP, CNTRL.H	Speed, flux, Iqs, and Ids PI controllers
CONST331.H	ADMC331-related constants
DAC.DSP, DAC.H	Debugging functions for writing to DAC on ADMC connector board
DIVIDE.DSP, DIVIDE.H	Divide function
FLUX.H	Field oriented control constants
FPMATH.DSP, FPMATH.H	Floating point math routines
IR_INIT.DSP, IR_INIT.H	Initialization of IR PowIRtrain module
MATH_32B.DSP, MATH_32B.H	32 bit math routines
MODEL.DSP, MODEL.H	Flux estimation
PWM331.DSP, PWM331.H	PWM functions
ROMUTIL.H	ROM utility definitions
ENCO.DSP, ENCO.H	Encoder functions
VECT_TRA.DSP, VECT_TRA.H	Vector transformation functions
build.bat:	Batch file to build the DSP source code into an executable file main.exe which can be downloaded and run on the DSP.
ACQUIRE.C, ACQUIRE.H:	C++ source codes for acquiring signals to PC
GA.M	MATLAB® program to optimize EKF by using GA
EKF.M	MATLAB® program to implement EKF algorithm

11.7.2 GA-EKF Experimental Method

The GA-EKF experiment is divided into the training phase and the verification phase. In the training phase, three-phase voltage and current are acquired as training samples from the experimental system and the actual rotor speed is acquired from the encoder as the target function. After the matrices G, Q, and R have been obtained off-line using the real-coded GA, the performance of the GA-EKF is examined in the verification phase by acquiring new data samples. Figure 11.24 shows the GA training phase procedure and Figure 11.25 shows the verification phase procedure.

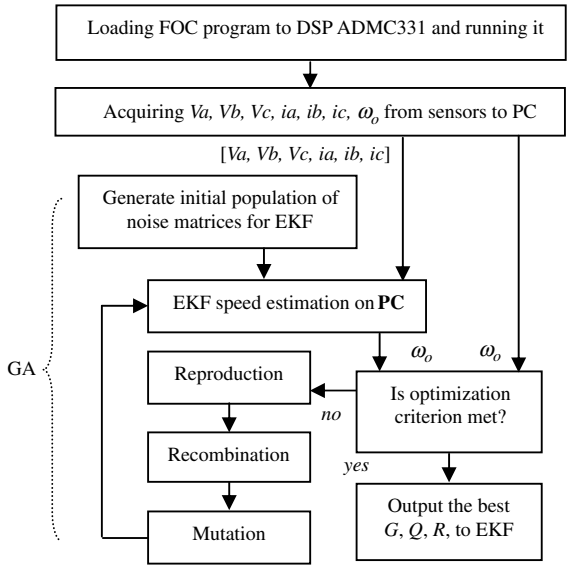


Figure 11.24 Flowchart of GA-EKF program for the training phase. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

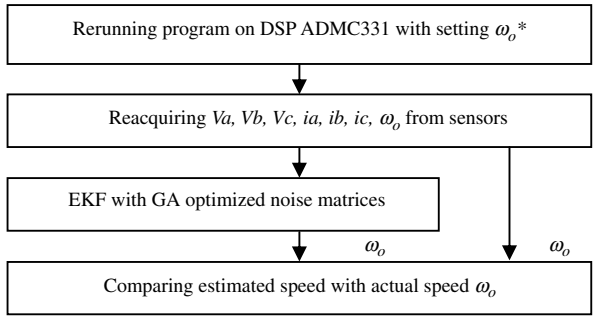


Figure 11.25 GA-EKF speed estimation procedure in the verification phase.

11.7.3 GA-EKF Experiments

The hardware system for the experimental investigations has been described in Figure 11.1 and the following parameters have been chosen for data acquisition via the PCL818HG card:

Sampling rate = 20 kHz for scanning 7-way signals (*Va*, *Vb*, *Vc*, *ia*, *ib*, *ic*, Encoder) Sampling rate of each way signal = 2.857 kHz (=20/7 kHz)
Sampling time = 1.4 s

Total sample number = 28 000
Sample number of each signal = 4000

11.7.3.1 GA EKF in the Training Phase

Figure 11.26 shows the acquired phase voltage and phase current when the induction motor is run up from standstill to 105 rad/s.

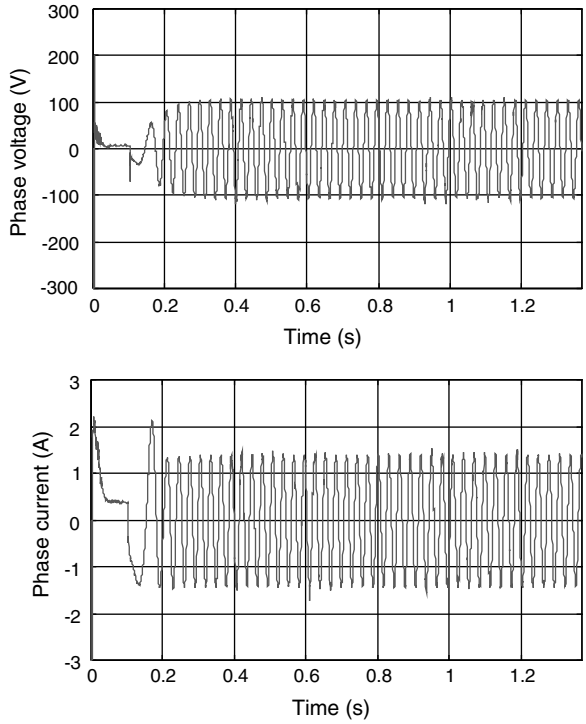


Figure 11.26 Phase voltage and phase current waveforms of the induction motor when run up to 105 rad/s. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

Figure 11.27 shows the actual speed and the speed estimated using EKF with matrices $G = Q = Diag[10^{-3}, 10^{-3}, 10^{-3}, 10^{-3}, 10^{-2}]$ and $R = Diag[10^{-3}, 10^{-3}]$. In this case, the mean squared error of the estimated speed is 48.1632.

The real-coded GA used to optimize the matrices G, Q, and R of the EKF has been described in Chapter 9. In the training stage, the real-coded GA parameters for the EKF experiment are set as follows:

1. Initial population size: 100
2. Maximum number of generations: 20

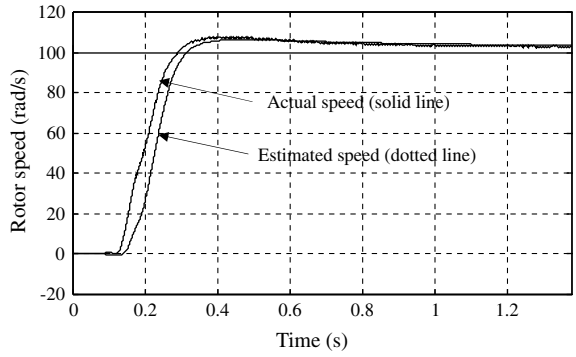


Figure 11.27 Actual speed of rotor and speed estimated using EKF.

3. Probability of crossover: 0.8
4. Mutation probability: 0.01
5. Initial range of real-coded strings: [0.01; 5].
6. Performance measure: the mean squared error between the estimated speed and the actual rotor speed.

Table 11.12 shows the convergence process in EKF training. The GA optimization method has improved the EKF performance by decreasing the mean squared error of estimated speed from 48.1632 to 1.2980. The optimized matrices are found to be: $G = Diag[0.1264, 0.1306, 0.1847, 0.1945, 1.5603]$, $Q = Diag[0.0648, 0.0993, 0.0816, 0.0963, 1.5000]$, and $R = Diag[0.0101,$

Table 11.12 Iteration process of the off-line GA.

Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$	Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$
0	12.6931		
1	4.0216	11	1.6493
2	4.3045	12	1.6320
3	3.1341	13	1.5234
4	3.0235	14	1.6019
5	2.8401	15	1.4820
6	2.5294	16	1.4397
7	2.1983	17	1.3589
8	2.2631	18	1.3740
9	1.9578	19	1.3138
10	1.8653	20	1.2980

s: actual rotor speed; e: estimated speed; n: number of data samples (= 4000); E: mean squared error of estimated speed.

0.0104]. It is apparent that the noise covariance and weight matrices required depend on the motor whose speed is to be estimated.

Figure 11.28 shows the actual speed and the estimated speed by using the optimized EKF. The mean squared error of the estimated speed is 1.2980.

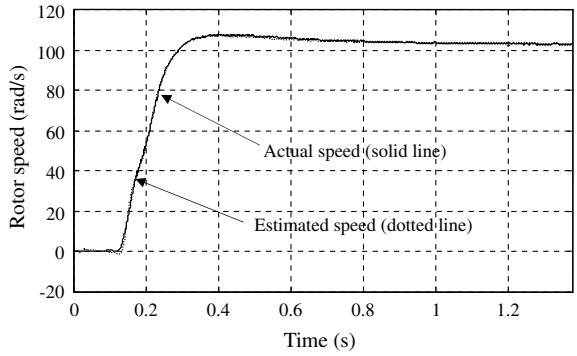


Figure 11.28 Actual speed of rotor and speed estimated using GA-EKF. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

11.7.3.2 GA EKF in Verification Phase

In the verification phase, the induction motor is run up from standstill to 188 rad/s by the same FOC program. Data samples are acquired from the DSP drive system to the PC and the GA-EKF program is run again to check the accuracy of the speed estimation.

Figure 11.29 shows the phase voltage, phase current, and estimated speed of the motor. The mean squared error of the estimated speed is found to be 1.7387, which is very close to that obtained in the training phase.

The experimental results demonstrate that the off-line GA method is efficacious for optimizing an EKF speed estimator. In order to obtain satisfactory covariance and weight matrices by the off-line GA, a short sampling period must be chosen for actual data acquisition. In this experiment, DMA method is used to implement high-speed data acquisition for the 7-way signals (*Va*, *Vb*, *Vc*, *ia*, *ib*, *ic*, and ω). At a sampling rate of 20 kHz, which corresponds to a sampling time interval of 0.00035s (=7/20 000), very satisfactory GA training and EKF verification results have been obtained.

11.7.4 Limitations of GA-EKF

Because the real-coded genetic algorithm begins from a randomly generated initial population of the long real-coded strings, the distribution of values in the initial population may affect the convergence process of the algorithm. The optimization process may need to be repeated several times before the global optimum is obtained. In the simulation studies

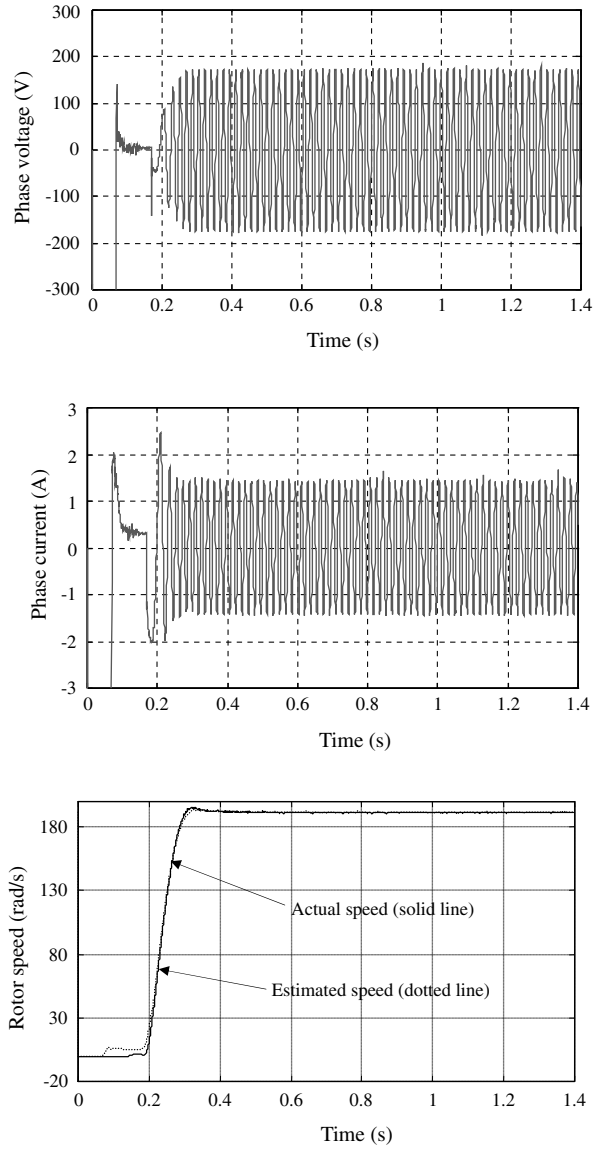


Figure 11.29 Phase voltage, phase current and speed estimated using GA-EKF when the motor is run up to 188 rad/s. (Reproduced by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of an induction motor drive using an optimized extended Kalman filter," *IEEE Transactions on Industrial Electronics*, **49**(1), 2002: 124–133. © 2002 IEEE.)

and the experimental studies, it is also found that an improper initial range of the covariance matrices may result in extremely long time for GA iterations or even GA optimization failure. The initial range is different for various induction motor types, for example, the initial range is [0.0001; 0.1] for the 7.5 kW motor in Chapter 7 and is [0.01; 5] for the 147 W motor.

As an example, consider the case where the initial range of EKF covariance matrices is set as [0.0001; 0.1] for the experimental 147-W motor. The GA iteration process is shown in Table 11.13.

Table 11.13 Iteration process of a GA failure case.

Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$	Generations	$E = \frac{1}{n}\sum_{i=1}^{n}(s_i - e_i)^2$
0	38.2961		
1	31.7557	11	29.9377
2	29.9381	12	29.8982
3	30.2317	13	29.9147
4	30.0430	14	29.9344
5	29.9381	15	29.8982
6	30.0021	16	29.8982
7	29.9376	17	29.8819
8	29.9175	18	29.8392
9	29.8982	19	29.8425
10	29.9378	20	29.8949

s: actual rotor speed; e: estimated speed; n: number of data samples (= 4000); E: mean squared error of estimated speed.

At the 20th iteration step, the mean squared error of the estimated speed is still at a large value of 29.8949 and the corresponding EKF matrices are: $G = Diag$[0.0326, 0.0425, 0.02764, 0.0139, 0.0832], $Q = Diag$[0.0174, 0.0268, 0.0460, 0.0533, 0.09524], and $R = Diag$[0.0230, 0.0147]. Figure 11.30 shows the actual speed and the speed estimated by the sub-optimal EKF.

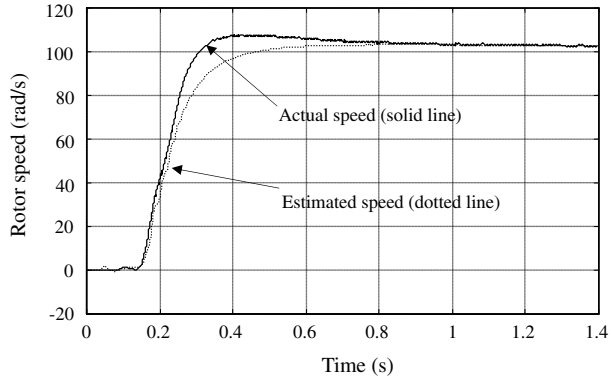


Figure 11.30 Actual speed and the estimated speed using the sub-optimal EKF.

11.8 DSP Programming Examples

TMS320F28335 processor (Texas Instruments Inc., 2009a) is employed for the DSP programming examples. Working in association with MATLAB® software, TMS320F28335 is suitable for implementing an intelligent induction motor control system with the following features:

1. Up to 150 MIPS (million instructions per second) or 6.67-ns instruction cycle time, 32-bit floating-point CPU, up to 18 PWM outputs, 6 event capture inputs, and 16 channels 12-bit ADC (analog-to-digital converter) with 80-ns conversion rate.
2. Standard IEEE 1149.1 JTAG (*Joint Test Action Group*) interface is implemented on TMS320F28335 processor. With JTAG-based emulator hardware, the processor may be connected to a host PC through a USB (*Universal Serial Bus*) interface.
3. Supports RTDX (Real Time Data exchange) technique. RTDX enables real-time, asynchronous exchange of data between the target DSP and the host PC, without stopping the tasks in the target DSP. When a motor control program is running in the DSP, another complex intelligent control program, such as ANN or GA, may be run simultaneously on the host PC to support the motor control program running in the target DSP.
4. Supports Code Composer Studio (CCS) IDE (integrated development environment) and C++ compiler/assembler/linker, DSP/BIOS, and digital motor control software.

In this section, four examples are presented to demonstrate basic DSP programming of an induction motor control system. The four examples are (1) generation of 3-phase sinusoidal PWM pulses for driving a 3-phase inverter; (2) RTDX for exchange of real-time data between the DSP and MATLAB® software; (3) acquisition of analog signals to the DSP by using an ADC; (4) capturing rotor encoder pulses to DSP by CAP programming.

The hardware for the four programming examples includes a host PC, DSP experiment board, and an oscilloscope. The DSP experiment board may be an eZdspF28335 development board (delivered by Spectrum Digital, Inc.) or a TMS320F28335 experimenter kit (delivered by Texas Instruments Inc.). The two boards have an on board JTAG connector that provides interface to a PC via a USB connection. The oscilloscope is used for observing the output waveforms of the DSP board. The hardware configuration is shown in Figure 11.31.

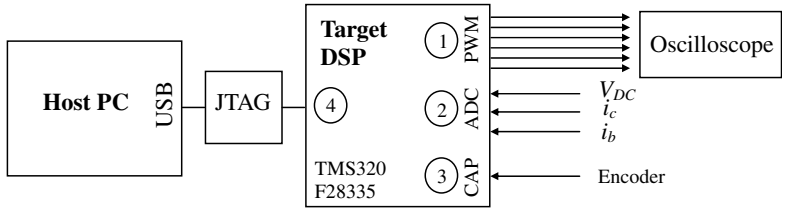


Figure 11.31 Hardware configuration of DSP programming examples.

In the TMS320F28335 target DSP, the three peripherals (PWM, CAP, ADC) and JTAG port implement the four functions.

1. The PWM ports output the six PWM pulses which control the inverter that feeds the induction motor.
2. Stator currents and voltage of inverter DC supply are acquired by the ADC ports to the DSP. With the acquired voltage of inverter DC supply and the PWM signals produced in the DSP, output voltages of the inverter may be obtained by their product.
3. Pulses yielded by a rotor speed encoder may be captured by the CAP port.
4. Through the on-board JTAG hardware interface, values of stator voltage, stator current, and rotor speed may be transferred to the host PC for real-time analysis and calculation. Complex intelligent algorithms that cannot run in the present-day DSP, such as ANN and GA, may be asynchronously run on the host PC by MATLAB® software with data acquired in real-time. The optimized parameters obtained by the intelligent algorithm in the host PC may be instantly transferred to the DSP to update the running program.

The software for the four programming examples include CCStudio_v3.3 with bios_5_33_06 software package (Texas Instruments Inc., 2009b) and MATLAB® 2008b with Embedded IDE Link CC software package (The MathWorks, Inc. 2008). The software configuration in the host PC and the application program loaded in the target DSP are shown in Figure 11.32.

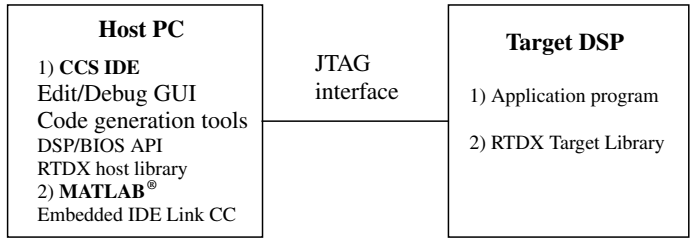


Figure 11.32 Software configuration of the host PC and target DSP.

In the software configuration, CCS IDE includes integrated Edit/Debug GUI (graphical user interface), code generation tools, DSP/BIOS, and RTDX host library. DSP/BIOS kernel is a real-time, multi-tasking software, also called a real-time DSP operating system (Texas Instruments Inc., 2009b). DSP/BIOS provide standardized API (Application Program Interface) functions for even the most sophisticated DSP applications. RTDX software consists of the target library and the host library. A small RTDX software library runs on the target DSP. This library transfers the required data to or from the host PC in real-time through a JTAG interface when the DSP application is running. On the host PC, the RTDX host library operates in conjunction with MATLAB® to obtain the target data or to send data to the target DSP application through the JTAG interface. 'Embedded IDE Link CC' software package of MATLAB® may be embedded to the Code Composer Studio software. Using RTDX technique, bi-directional real-time data exchange between MATLAB® software and the TMS320F28335 processor program can be easily implemented.

To verify that the board has been connected to the host PC, start MATLAB® and enter 'ccsboardinfo' in the MATLAB® command line.

If the development board eZdspF28335 is connected to the host PC, the following information will be shown in the MATLAB® window.

```
>> ccsboardinfo
Board          Board          Proc Processor          Processor
Num            Name               Num Name               Type
--- ------------------------------------------------------------
0            F28335_eZdsp          0                   cpu_0
```

If the TMS320F28335 experimenter kit is connected to the host PC, the following information will be shown in the MATLAB® window.

```
>> ccsboardinfo
Board          Board          Proc Processor          Processor
Num            Name               Num Name               Type
--- ------------------------------------------------------------
0          F28335 XDS100           0              TMS320C2800_0
           USB Emulator
```

11.8.1 Generation of 3-Phase Sinusoidal PWM

This example demonstrates programming 3-phase sinusoidal PWM for an inverter-fed induction motor in a TMS320F28335 processor (Texas Instruments Inc., 2009c). The 3-phase PWM operates at a synchronization frequency of 1 kHz. The three-phase PWM waveforms are produced by three events, namely PWM1, PWM2 and PWM3. PWM1 works as a master and a PWM1 event is started by the processor timer. Next, PWM1 triggers a hardware interrupt to start event PWM2 which then works as a slave. Finally, PWM2 triggers a hardware interrupt to start event PWM3 which then works as a slave.

Step 1 Connect the TMS320F28335 experiment board to the PC via USB interface. Start Code Composer Studio (CCS) and create a new project named as 'F28335_example_PWM.pjt' by selecting the 'New' item from the 'Project' menu on CCS.

Step 2 Write a main program 'Main_PWM.c' and add it to the project. The main program is listed below:

```
#include 'DSP2833x_Device.h'          // Peripheral address definitions
#include 'F28335_example_PWM.h'       // Main include file
void main(void)
{
InitSysCtrl();                       // Initialize the CPU
InitPieCtrl();                      // Initialize and enable the PIE
(Peripheral Interrupt Expansion)
InitGpio();                        / Initialize the shared GPIO pins
InitEPwm();                       // Initialize the 3-phase PWM
}
```

Step 3 Write the four initialization functions and add them to the project.
'InitSysCtrl()': Configure the registers of clocking and system control (Texas Instruments Inc., 2009d). The system clock is set at 75 MHz with a register script SysCtrlRegs.PLLSTS.bit. DIVSEL = 0x2.

'InitGpio()': Configure the GPIO (General-Purpose Input/Output) ports as PWM ports with following scripts (Texas Instruments Inc., 2009d).

```
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;      // Set GPIO0 port as EPWM1A port
GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;      // Set GPIO1 port as EPWM1B port
GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;      // Set GPIO2 port as EPWM2A port
GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;      // Set GPIO3 port as EPWM2B port
GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;      // Set GPIO4 port as EPWM3A port
GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1;      // Set GPIO5 port as EPWM3B port
```

'InitEPwm()': This example implements symmetrical PWM with PWM1 configured as a master, PWM2 and PWM3 configured as slaves. There are seven control register modules in the enhanced PWM (EPWM) system to implement various PWM control techniques as shown below:

• Time-Base Module	(TBCTL registers)
• Counter Compare Module	(CMPCL registers)
• Action Qualifier Module	(AQCTLA/B registers)
• Dead-Band Generator Module	(DBCTL registers)
• PWM Chopper (PC) Module	(PCCTL registers)
• Trip Zone Module	(TZCTL registers)
• Event Trigger Module	(ETSEL registers)

The configuration in the EPWM registers has a long script (please refer to the book companion website). To understand the full technical details, readers should consult the Enhanced Pulse Width Modulator (ePWM) Module Reference Guide (Texas Instruments Inc., 2009c).

After completing the configuration of EPWM registers, the following scripts in 'InitEPwm()' are necessary to make PWM interrupts work.

```
/* Enable CPU INT3 which is connected to EPWM1-3 INT */
IER |= M_INT3;

/* Enable EPWM INTn in the PIE: Group 3 interrupts 1-3 */
PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
PieCtrlRegs.PIEIER3.bit.INTx2 = 1;
PieCtrlRegs.PIEIER3.bit.INTx3 = 1;

/* Clear three PWM interrupt flags */
EPwm1Regs.ETCLR.bit.INT = 1;
EPwm2Regs.ETCLR.bit.INT = 1;
EPwm3Regs.ETCLR.bit.INT = 1;
```

Step 4 Write a header file 'F28335_example_PWM.h' including the following scripts and add it to the project.

```
// Set half period of the PWM and initial duty cycle

#define PWM_HALF_PERIOD 37500   // 1 kHz PWM at 75MHz system clock

#define PWM_DUTY_CYCLE 18750    //50% duty cycle
```

Step 5 Create a new DSP/BIOS configuration 'f28335_example_PWM.tcf' by selecting 'New' and 'DSP/BIOS configuration' from the 'File' menu in CCS and add the new DSP/BIOS to the project. PWM hardware interrupts may be set by assigning the three interrupt service routines, '_EPWM1_INT_ISR', '_EPWM2_INT_ISR', and '_EPWM3_INT_ISR' to the three PIE (Peripheral Interrupt Expansion) interrupt items 'PIE_INT3_1' 'PIE_INT3_2', and 'PIE_INT3_3', respectively. The assignment is done in the 'PIE INTERRUPTS' folder of the DSP/BIOS.

Step 6 Write PWM interrupt service routines (ISR) to adjust the duty cycle according to the sinusoidal reference function as follows and add them to the project.

```
#include 'DSP2833x_Device.h'            // Peripheral address definitions
#include 'F28335_example_PWM.h'         // Main include file
#include 'math.h'                       // Include sine function
double A1=0.0, A2=0.0, A3=0.0, B;
Uint32 C;
double TWO_PI_div20 = 0.3141592653589795;          // = 2π/20

void EPWM1_INT_ISR(void)              // PIE3.1 @ 0x000D60 EPWM1_INT (EPWM1)
{
PieCtrlRegs.PIEACK.all = PIEACK_GROUP3; // Must acknowledge the PIE group
A1=A1+TWO_PI_div20;
if (A1 >= TWO_PI)
A1=0.0;
B = sin(A1);                                    // Call a sine function
B=18750-0.8* B*18750;
C= (Uint32) B;                                // Converter to 32 bit Uint data
EPwm1Regs.CMPA.half.CMPA=C;                     // Update compare register
EPwm1Regs.ETCLR.bit.INT = 1;                    // Clear PWM1 interrupt flag
}

void EPWM2_INT_ISR(void)              // PIE3.2 @ 0x000D62 EPWM2_INT (EPWM2)
{
PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;  //Must acknowledge the PIE group
A2=A2+TWO_PI_div20;
if (A2 >= TWO_PI)
A2=0.0;
B = sin(A2-TWO_PI/3);          // Call a sine function with phase as -2π/3
B=18750-0.8* B*18750;
C= (Uint32) B;                              // Converter to 32 bit Uint data
EPwm2Regs.CMPA.half.CMPA=C;                 // Update compare register
EPwm2Regs.ETCLR.bit.INT = 1;                // Clear PWM2 interrupt flag
}

void EPWM3_INT_ISR(void)              // PIE3.3 @ 0x000D64 EPWM3_INT (EPWM3)
{
PieCtrlRegs.PIEACK.all = PIEACK_GROUP3; // Must acknowledge the PIE group
A3=A3+TWO_PI_div20;
if (A3 >= TWO_PI)
A3=0.0;
B = sin(A3+TWO_PI/3);            // Call a sine function with phase as 2π/3
B=18750-0.8* B*18750;
C=(Uint32) B;                       // Converter to 32 bit Uint data
EPwm3Regs.CMPA.half.CMPA=C;                 // Update compare register
EPwm3Regs.ETCLR.bit.INT = 1;                // Clear PWM3 interrupt flag
}
```

Note: The execution time of the sin function is 37 instruction cycles (Texas Instruments Inc., 2008).

Step 7 By selecting 'Rebuild All' in the 'Project' menu of the CCS window, the executable code is generated. The filename of the executable code may be set in the 'Build Options' window in the 'Project' menu. In this example, it is named as 'F28335_example_PWM.out'.

Step 8 Connect the Code Composer Studio (CCS) to the TMS320F28335 processor by selecting the item 'Connect' from the CCS 'Debug' menu.

Step 9 Reset the CPU and load the executable code 'F28335_example_PWM.out' into the processor by selecting the item 'Load Program' from the CCS 'File' menu.

Step 10 Connect pins GPIO0 (PWM1A) and GPIO1 (PWM1B) in the processor to the oscilloscope channel 1 and channel 2, and click key 'F5' to start the program in the processor. Waveforms of PWM1A and PWM1B can be observed on the oscilloscope as shown in Figure 11.33.

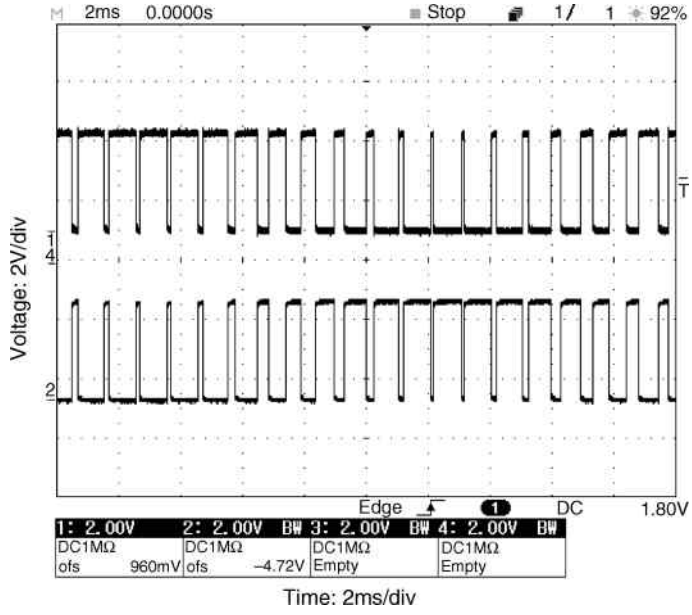


Figure 11.33 Waveforms of PWM1A and PWM1B displayed on the oscilloscope.

Click key 'Shift-F5' to halt the executable code. Connect pins of the processor, GPIO0 (PWM1A), GPIO2 (PWM2A), and GPIO4 (PWM3A) to the oscilloscope channels 1, 2, and 3, respectively. Click key 'F5' to start the program again. The three-phase sinusoidal PWM output PWM1A, PWM2A and PWM3A can be observed on the oscilloscope as shown in Figure 11.34.

Connect a low-pass filter consisting of a 1-kΩ resistance and a 4.7-μF to the pin GPIO0 (PWM1) of the processor and connect the output of the filter to channel 1 of the oscilloscope. The output waveform shown in Figure 11.35 may be observed.

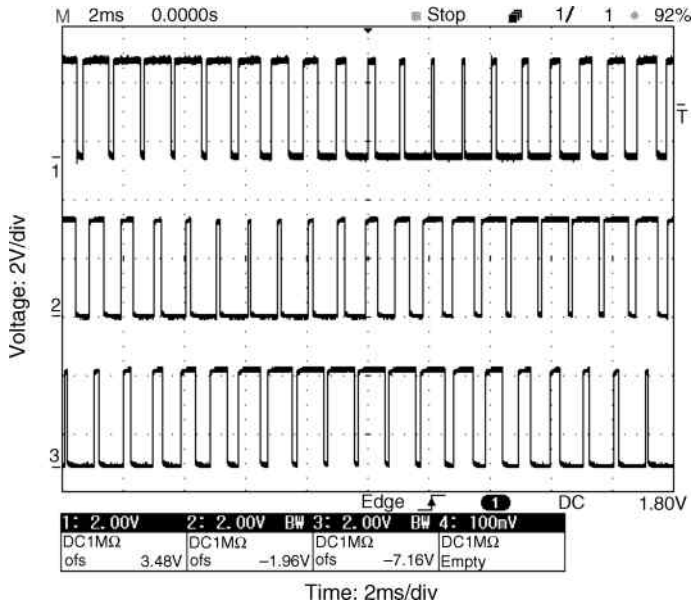


Figure 11.34 Waveforms of PWM1A, PWM2A, and PWM3A in the oscilloscope.

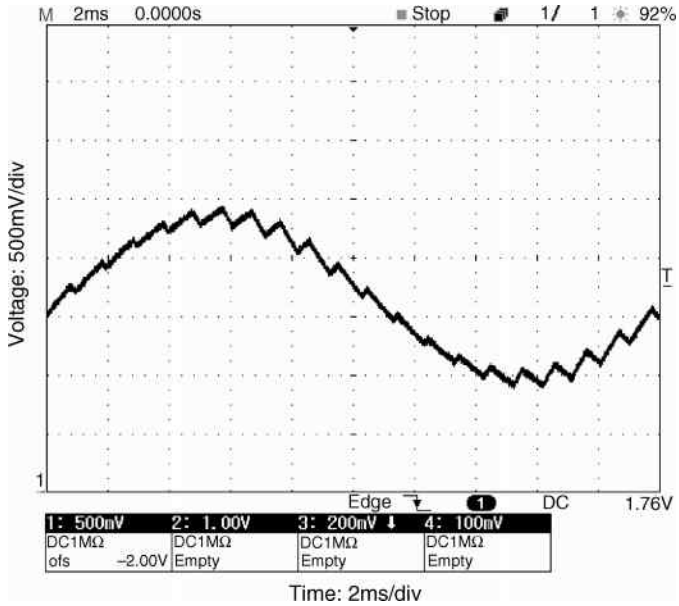


Figure 11.35 Filter output waveform from PWM1A.

11.8.2 RTDX Programming

This example demonstrates RTDX programming to exchange data between the TMS320F28335 processor and MATLAB®. In this example, the application program in the processor is started by a MATLAB® script and an array is sent to the processor from MATLAB®. In the main program in the processor, a software interrupt is posted to call an interrupt service routine for receiving the array from MATLAB®. After the processor has received the array, the value of every element in the array is increased by 1 and a second software interrupt is posted to call an interrupt service routine for sending the new array to MATLAB® from the processor.

Step 1 Create a new project 'RTDX.pjt'.

Step 2 Write a main program 'swi.c' as listed below and add it to the project.

```
#include <rtdx.h>                          // RTDX header file
#include 'swicfg.h'                        // Header file created by BIOS
RTDX_CreateInputChannel(ichan);            // Create a input channel
RTDX_CreateOutputChannel(ochan);           // Create a output channel
double A[64];

void main(Int argc, Char *argv[])
{
  RTDX_enableInput(&ichan);                // Enable the input channel
  RTDX_enableOutput(&ochan);               // Enable the output channel
  SWI_post(&SWI0);                         // Post software interrupt SWI0
}
```

Step 3 Create a new DSP/BIOS configuration 'swi.tcf' and add it to the project. Two software interrupts are set by adding two software interrupt items 'SWI0' and 'SWI1' into the DSP/BIOS. The two software interrupts are assigned to trigger the interrupt service routines '_swiFxn0' and '_swiFxn1'. By entering '_swiFxn0' and '_swiFxn1' into the items 'SWI0' and 'SWI1', respectively, the configuration is completed.

Step 4 Write the interrupt service routines and add them to the project.

```
void swiFxn0(void)          // Interrupt service routine called by software
interrupt SWI0
{
  RTDX_read( &ichan, &A, sizeof(A) );      // Read data from MATLAB
  RTDX_disableInput(&ichan);               // Disable the input channel
  SWI_post(&SWI1);                         // Post software interrupt SWI1
}

void swiFxn1(void)           // Interrupt service routine called by software
interrupt SWI1
{
  int i;
  for (i = 0; i < 64; i + +)
  A[i] = A[i] + 1;                    // Add 1 to every element of the array
  RTDX_write( &ochan, &A, sizeof(A) );                // Write data to MATLAB
  RTDX_disableOutput(&ochan);                // Disable the output channel
}
```

Step 5 Create an executable code 'RDTX.out'. The filename of the executable code may be set in 'Build Options' in CCS.

Step 6 Connect the Code Composer Studio (CCS) to the TMS320F28335 processor by selecting the item 'Connect' from the CCS 'Debug' menu.

Step 7 Reset the CPU, and load the executable code 'RDTX.out' into the processor.

Step 8 Enable RTDX in CCS by opening the 'RTDX Control' window from the 'RTDX' folder in the 'Tools' menu on the CCS, then selecting the 'RTDX Enable' box from the opened 'RTDX Control' window.

Step 9 Start MATLAB® and write a program 'Example_RTDX.m' as follows:

```
function void = Example_RTDX()
clear('all')
A = 0:0.1:6.3;
Send_array = sin(A);   % Create a array of sine function

% Connect MATLAB to CCS
cc = ticcs;                  % Get a handle to Code Composer Studio (CCS)
rx = cc.rtdx;                % created an alias rx to the RTDX portion of cc
open(rx,'ichan','w')         % Open a channel for write access
enable(rx,'ichan')           % Enable write access
open(rx,'ochan','r')         % Open a channel for read access
enable(rx,'ochan')           % Enable read access
flush(rx,'ochan');           % Flush data from ochan channels

% Start processor application and write data from MATLAB to processor
cc.run('run');                          % Start the processor program
writemsg(rx,'ichan',Send_array);        % Write data to the processor

% Read the data from processor to MATLAB
% Determine the number of messages
number_msgs = msgcount(rx,'ochan')
if ( number_msgs >=1)
Read_array = cc.rtdx.readmsg('ochan','double', number_msgs);
end                          % Read data to MATLAB
halt(cc);                    % Halt the processor
disable(rx,'all')
close all;

% Plot waveforms
plot(Send_array);            % Plot the write data
hold on;
plot(Read_array);            % Plot the read data
axis([1 64 -1.5 2.5]);
end
```

Step 10 Run the program 'Example_RTDX.m' in MATLAB® platform. The results are shown in Figure 11.36.

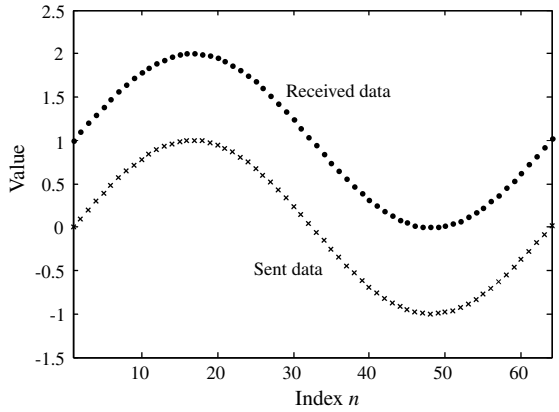


Figure 11.36 Data sent to the processor and data received from it.

11.8.3 ADC Programming

This example demonstrates ADC (analog-to-digital converter) programming to acquire analog signals from an ADC port of the TMS320F28335 processor and to transfer them to MATLAB® by RTDX technique. The ADC module in TMS320F28335 has following features (Texas Instruments Inc., 2007).

- 12-bit ADC core with built-in sample-and-hold
- Analog input range: 0.0–3.0 V
- Fast conversion rate: Up to 80 ns at 25-MHz ADC clock.
- 16 ADC channels
- Sixteen result registers to store conversion values and the digital value of the input analog voltage is derived by: Digital Value = 4096*(Input Analog Voltage)/3.

In this example, the application program in the processor is started by a MATLAB® script. The ADC in the processor is triggered by PWM4 timer at a sampling rate of 2.5 kHz. From the 'ADCINA0' port of the processor, analog voltage is converted to digital value. After the conversion is completed, a hardware interrupt is triggered to call an interrupt service routine 'ADCINT_ISR'. In the routine 'ADCINT_ISR', a software interrupt is posted to call an interrupt service routine 'AdcSwi'. In the routine 'AdcSwi', the digital value is saved to an array. After 50 samples have been acquired, a task interrupt is issued by the routine 'AdcSwi' to call an interrupt service routine 'task' which sends the samples acquired to MATLAB®.

Step 1 Create a new project 'F28335_example_ADC.pjt'.

Step 2 Write the main program 'Main_ADC.c' as follows:

```
#include 'DSP2833x_Device.h'              // Peripheral address definitions
#include 'F28335_example.h'               // Main include file
#include <rtdx.h>                         // RTDX
Uint16 AdcBuf[50];                        // ADC data buffer allocation
RTDX_CreateOutputChannel(ochan);

void main(void)
{
  InitSysCtrl();                         // Initialize the CPU
  InitAdc();                             // Initialize the ADC
  InitEPwm();                            // Initialize the PWM
  RTDX_enableOutput(&ochan);             // Enable the output channel
}
```

Step 3 Write the initialization functions as in **Step 3** in Section 11.8.1.

Step 4 Create a new DSP/BIOS configuration 'f28335_example_ADC.tcf' and add it to the project. ADC hardware interrupt is set by entering '_ADCINT_ISR' into 'PIE_INT1_6' in the DSP/BIOS. The hardware interrupt 'PIE_INT1_6' will call the interrupt service routine 'ADCINT_ISR'. A software interrupt is set by adding a software interrupt item 'ADC_swi' into the DSP/BIOS and entering '_AdcSwi' into 'ADC_swi'. The software interrupt 'ADC_swi' will call the interrupt service routine 'AdcSwi'. A task interrupt is set by adding task interrupt item 'TSK1' into the DSP/BIOS and entering '_task' into the task interrupt item 'TSK1'which will call the interrupt service routine 'task'.

Step 5 Write the software interrupt service routines and task interrupt service routines:

```
void AdcSwi(void)          // Interrupt service routine called by software
interrupt ADC_swi

{
static Uint16 *AdcBufPtr = AdcBuf;                    // Pointer to buffer
AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;             // Reset Sequencing Control
Registers SEQ1
AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;        // Clear ADC SEQ1 interrupt flag
*AdcBufPtr++ = AdcRegs.ADCRESULT0 >> 4;              // Read the ADC result

  if( AdcBufPtr == (AdcBuf + 50) )
  {
    AdcBufPtr = AdcBuf;                 // Rewind the pointer to beginning
    TSK_yield();                             // Yield a task interrupt
  }
}

void task(void)                // Interrupt service routine called by task
interrupt TSK1

{
  RTDX_write( &ochan, &AdcBuf, sizeof(AdcBuf) );     // Write data to MATLAB
  RTDX_disableOutput(&ochan);              // Disable the output channel
}
```

Step 6 Create an executable code 'F28335_ADC.out'. The filename of the executable code may be set in the item 'Build Options' in CCS.

Step 7 Connect the Code Composer Studio (CCS) to the TMS320F28335 processor by selecting the item 'Connect' from the CCS 'Debug' menu.

Step 8 Reset the CPU, and load the executable code 'F28335_ADC.out' into the processor.

Step 9 Enable RTDX in the CCS window as described in **Step 7** of the example in Section 11.8.2.

Step 10 Start MATLAB® and write a program 'Example_ADC.m' as follows:

```
function void = Example_ADC()

clear('all')

% Connect MATLAB to CCS
cc = ticcs;                 % Get a handle to Code Composer Studio (CCS)
rx = cc.rtdx;               % created an alias rx to the RTDX portion of cc
open(rx,'ochan','r')        % Open a channel for read access
enable(rx,'ochan')          % Enable read access

% Start processor application from MATLAB platform
cc.run('run');              % Start the processor program

% Read the ADC data from processor to MATLAB
number_msgs = msgcount(rx,'ochan')      % Determine the number of messages
Read_array = cc.rtdx.readmsg('ochan','int16', number_msgs);
                % Read data from the processor to MATLAB
Voltage = double(Read_array)*3/4096;      % Coverter ADC digital value
                                          % to analog voltage value
halt(cc);                                 % Halt the processor

% Plot waveform
plot(Voltage);                            % Plot the read voltage waveform
end
```

Step 11 Connect the pin 'ADCINA0' on the TMS320F28335 processor to ground.

Step 12 Run the program 'Example_ADC.m' in the MATLAB® window. The results are shown in Figure 11.37.

Step 13 Connect the positive terminal of a battery to the pin 'ADCINA0' on the TMS320F28335 processor and the negative terminal to the ground of the processor. The battery voltage measured using a multimeter is about 1.6 V.

Step 14 Restart the processor application by selecting 'Restart' in the 'Debug' menu on CCS.

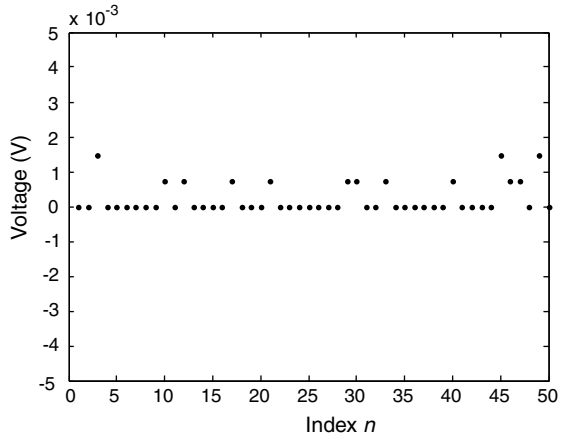


Figure 11.37 ADC result of zero voltage from the processor.

Step 15 Run the program 'Example_ADC.m' in MATLAB®. The results are shown in Figure 11.38.

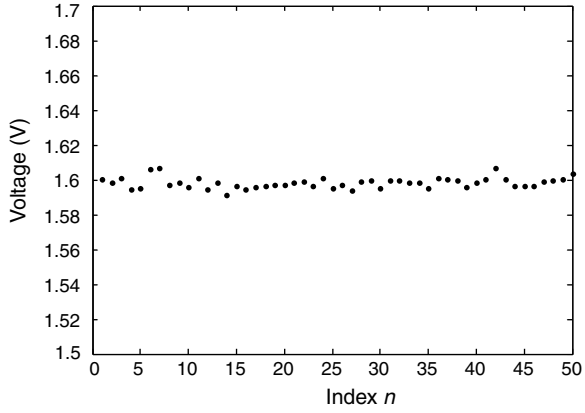


Figure 11.38 Battery voltage acquired by the DSP via an ADC.

11.8.4 CAP Programming

The enhanced capture (eCAP) module (Texas Instruments Inc., 2009e) in TMS320F28335 processor may be used to capture a pulse and to measure its period and duty cycle. In this example, the application program in the processor is started by a MATLAB® script. The continuous sinusoidal PWM pulse created from GPIO0 port of the processor serves as the signal source. By connecting the GPIO0 pin to GPIO5 pin, the PWM pulse is captured to the processor from GPIO5 port. After the eCAP module has captured two pulses, a hardware interrupt 'PIE_INT4_1' is triggered. The hardware interrupt will call an interrupt service routine 'ECAP1_INT_ISR' which calculates the period and duty cycle of the captured

pulse and then post a software interrupt 'SWI0'. In an interrupt service routine 'swi_RTDX', the period and duty cycle of the pulses are sent to MATLAB®. After the period and duty cycle of the pulses are received by MATLAB®, the waveform of the pulses is reconstructed and plotted. Figure 11.39 shows a pulse capture process.

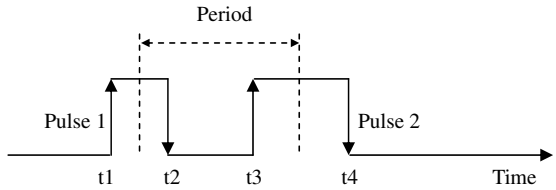


Figure 11.39 Capture process of two pulses in TMS320F28335.

In the capture process:

Time 't1', which coincides with a rising edge, is captured into register 'ECap1Regs.CAP1';
Time 't2', which coincides with a falling edge, is captured into register 'ECap1Regs.CAP2';
Time 't3', which coincides with a rising edge, is captured into register 'ECap1Regs.CAP3';
Time 't4', which coincides with a falling edge, is captured into register ECap1Regs.CAP4'.

The period and duty cycle of the pulses may be calculated by the values in the registers as follows:

Duty cycle of pulse 1 = t1 − t2;
Duty cycle of pulse 2 = t4 − t3;
Period = [(t3 − t1) + (t4 − t2)]/2.

Step 1 Create a new project 'F28335_example_CAP.pjt'.
Step 2 Write a main program 'F28335_example_CAP.c' as follows and add it to the project.

```
#include 'DSP2833x_Device.h'          // Peripheral address definitions
#include 'F28335_example_CAP.h'       // Main include file
#include <rtdx.h>                      // RTDX
RTDX_CreateOutputChannel(ochan);
Uint32 PwmDuty;                       // Measured PWM duty cycle
Uint32 PwmPeriod;                     // Measured PWM period
Uint32 PwmDuty2;                      // Measured PWM duty cycle 2
Uint32 PwmPeriod2;                    // Measured PWM period 2
Uint32 To_pc[4];

void main(void)
{
  InitSysCtrl();                      // Initialize the CPU
  InitGpio();                         // Initialize the GPIO pins
  InitEPwm();                         // Initialize the PWM
  InitECap();                         // Initialize the Capture units

}
```

Step 3 Write initialization functions which are similar to **Step 3** in Section 11.8.1.

Step 4 Write a header file 'F28335_example_CAP.h' and add it to the project, as in **Step 4** in Section 11.8.1.

Step 5 Create a new DSP/BIOS configuration 'F28335_example_CAP.tcf' and add it to the project. CAP hardware interrupt may be set by entering '_ECAP1_INT_ISR' into 'PIE_INT4_1' in the DSP/BIOS. The hardware interrupt 'PIE_INT4_1' will call the interrupt service routine 'ECAP1_INT_ISR'. A software interrupt may be set by adding a software interrupt item 'SWI0' into the DSP/BIOS and entering 'swi_CAP' into the item 'SWI0'. The software interrupt 'SWI0' will call the interrupt service routine 'swi_CAP'.

Step 6 Write an interrupt service routine to create a PWM pulse as follows and add the routine to the project.

```
#include 'DSP2833x_Device.h'       // Peripheral address definitions
#include 'F28335_example_CAP.h'   // Main include file
#include 'math.h'
double AA=0.0, CC;
Uint32 BB;
double TWO_PI = 6.28318530717959;
double TWO_PI_div20 = 0.3141592653589795;

void EPWM1_INT_ISR(void)     // PIE3.1 @ 0x000D60 EPWM1_INT (EPWM1)
{
PieCtrlRegs.PIEACK.all = PIEACK_GROUP3; // Must acknowledge the PIE group

AA=AA+TWO_PI_div20;
if (AA >= TWO_PI)
AA=0.0;
CC = sin(AA);                        // Call sine function
CC=CC*18750;
CC=18750-CC*0.8;
BB=(Uint32)CC;                       // Converter to 32 bit Uint data
EPwm1Regs.CMPA.half.CMPA=BB;         // Update compare register
EPwm1Regs.ETCLR.bit.INT = 1;         // Clear PWM1 interrupt flag
}
```

Step 7 Write a CAP interrupt service routine to calculate the period and duty cycle of the captured pulses as follows and add the routine to the project.

```
void ECAP1_INT_ISR(void)        // PIE4.1 @ 0x000D70 ECAP1_INT (ECAP1)
{
PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;   // Must acknowledge the PIE
group
PwmDuty = (int32)ECap1Regs.CAP2 - (int32)ECap1Regs.CAP1;
PwmPeriod = (int32)ECap1Regs.CAP3 - (int32)ECap1Regs.CAP1;
PwmDuty2 = (int32)ECap1Regs.CAP4 - (int32)ECap1Regs.CAP3;
PwmPeriod2 = (int32)ECap1Regs.CAP4 - (int32)ECap1Regs.CAP2;
PwmPeriod = (PwmPeriod + PwmPeriod2)/2;
SWI_post(&SWI0);                          // Post software interrupt SWI0
ECap1Regs.ECCLR.bit.CEVT4 = 1;          // Clear the CEVT4 flag
}
```

Step 8 Write an interrupt service routine 'swi_CAP' as follows.

```
void swi_CAP(void)
{
To_pc[0] = PwmDuty;                    // Duty cycle of pulse 1
To_pc[1] = PwmPeriod;                  // Period of pulse 1
To_pc[2] = PwmDuty2;                   // Duty cycle of pulse 2
To_pc[3] = PwmPeriod;                  // Period of pulse 2 as same as pulse1
RTDX_write( &ochan, &To_pc, sizeof(To_pc) );          // Write data to
MATLAB
}
```

Step 9 Create an executable code 'F28335_example_CAP.out'. The filename of the executable code may be set in the 'Build Options' in CCS.

Step 10 Connect the Code Composer Studio (CCS) to the TMS320F28335 processor by selecting item 'Connect' from the CCS 'Debug' menu.

Step 11 Reset the CPU, and load the executable code 'F28335_example_CAP.out' into the DSP.

Step 12 Enable RTDX in CCS window as described in **Step 7** in the RTDX programming example in Section 11.8.2.

Step 13 Start MATLAB® and write a MATLAB® program 'Example_CAP.m' as follows.

```
function void = Example_CAP()
clear('all')

% Connect MATLAB to CCS
cc = ticcs;     %Get a handle to Code Composer Studio (CCS)
rx = cc.rtdx;   % created an alias rx to the RTDX portion of cc
open(rx,'ochan','r')   % Open a channel for read access
enable(rx,'ochan')     % Enable read access

% Start processorP application from MATLAB platform
cc.run('run');     %Start the processor application program
pause(0.02);       %Pause MATLAB 0.02s
halt(cc);          %Halt the processor application program

% Read the captured data from processor to MATLAB
number_msgs = msgcount(rx,'ochan')
              %Obtain number of unread message in RTDX channel
out_array = 0;
if ( number_msgs > = 1)
out_array = cc.rtdx.readmsg('ochan','int32', number_msgs);
                              %Read data from RTDX channel
A = cell2mat(out_array);      %Convert data to a single matrix
end

% Separate duty and period from the matrix A
Period = [];
```

```
Duty = [];
B = double(A)           %Convert int32 to double
C = B/(75*1e6)/2        %Convert to real value at processor Clock as 75 MHz
[n,m] = size(C)
for i = 1:m/2
Period(i) = C(i*2);       %Separate Period from the matrix
Duty(i) = C(i*2-1);       %Separate Duty from the matrix
end

% Build waveform of a sine cycle by captured duties and periods
t = 0:1e-6:1e-3;          %Create 1001 points on every pulse period
[nt,mt] = size(t);
Period(1) = [];           %Clear first period sample
```

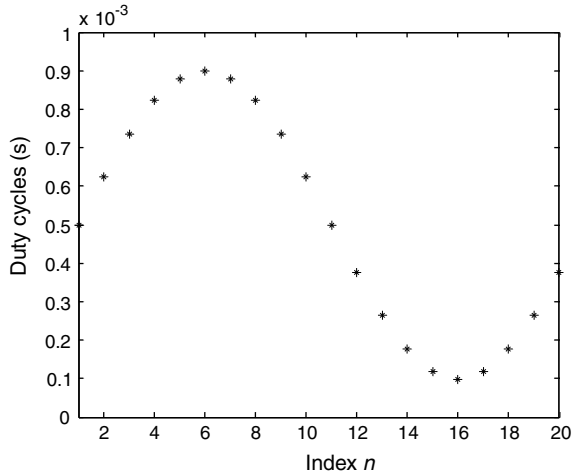


Figure 11.40 Duty cycles of pulses captured by the processor.

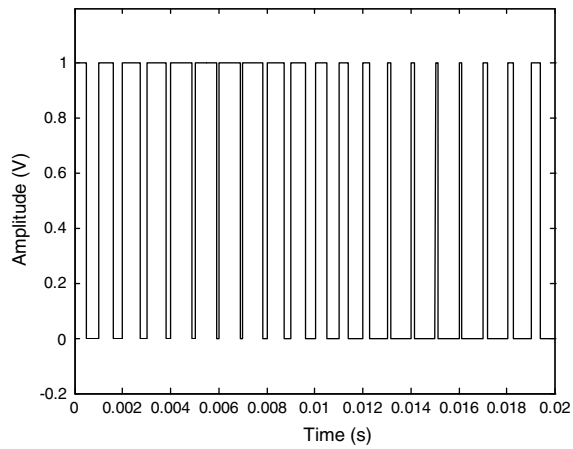


Figure 11.41 Waveform of pulses reconstructed by the data captured by the processor.

```
Duty(1) = [];              %Clear first Duty sample
mm = 20;   %20 samples in a cycle with 50 Hz reference and 1kHz carrier
p = [];
timeplot = [];
for j = 1:mm,
  if(t(i)<Duty(j))
    p(i + mt*(j-1)) = 1.0;
  else
    p(i + mt*(j-1)) = 0.0;
  end
  timeplot(i + mt*(j-1)) = (i + mt*(j-1))*1e-6;
end
end

% Plot pulse waveform and duty cycle data
plot(Duty,'.')                    % Plot duty cycle
axis([1 20 0 2e-3]);
plot(timeplot, p);                % Plot pulse waveform
axis([0 max(timeplot) -0.2 1.2]);
end
```

Step 14 Connect the GPIO0 pin of the processor to the GPIO5 pin by a wire and connect GPIO0 pin to the oscilloscope channel 1.

Step 15 Run the program 'RTDX_ MATLAB_CAP.m' in MATLAB®. Duty cycles of pulses are shown in Figure 11.40 and waveform of pulses is shown in Figure 11.41.

On the oscilloscope channel 1, waveform of continuous sinusoidal PWM source may be observed as shown in Figure 11.42.

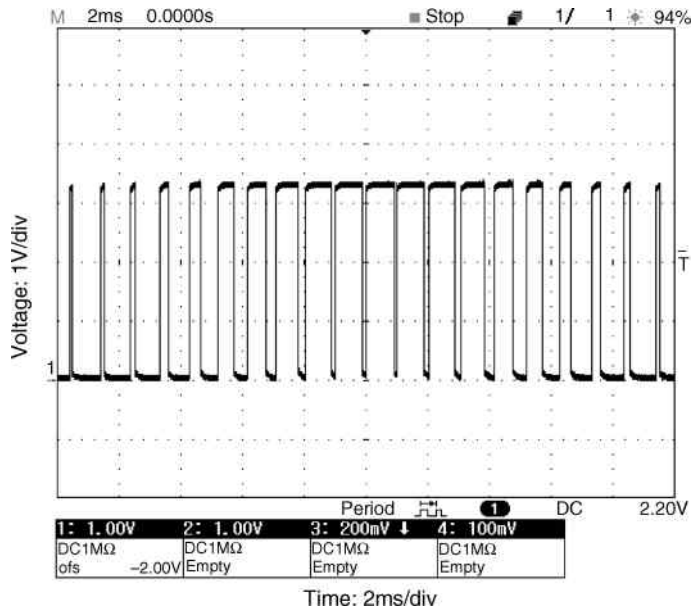


Figure 11.42 Waveform of continuous 1 kHz sinusoidal PWM.

11.9 Summary

This chapter has presented four experiments for verifying some of the intelligent control algorithms developed in previous chapters. The experimental system consists of an ADMC331 fixed-point DSP board (Analog Devices Inc.), an IGBT inverter power module IRPT1058A (International Rectifier Inc.), a data-acquisition board PCL818HG (Advantech Co.Ltd.), an encoder GBZ02 with a resolution of 200 pulses/revolution (China Sichuan Opto-electronic Co.), three-phase current sensor 3I411A (China WB institute), a host PC, a data-acquisition PC, and a three-phase induction motor. The hardware systems and devices from different manufacturers have been successfully integrated to give a functional experimental drive system. The system has the features of low cost, convenience in data acquisition, and expansion capability. However, program development at assembly code level is tedious and time-consuming. Based on the simulation models developed in previous chapters, three experimental programs (motor run up, fuzzy/PI control, and GA-EKF speed estimation) are developed with DSP assembly language, C++ language, and MATLAB® language.

Experiment 2 has verified the accuracy of the voltage-input model, PWM model, encoder model, and decoder model in Chapter 3. Experiments 3 and 4 have basically verified the feasibility of the Fuzzy/PI two-stage controller developed in Chapter 6 and the GA-EKF speed estimator developed in Chapter 9. The insight and experience gained have provided a firm foundation for the practical implementation of other intelligent control algorithms. Additional technical problems need to be tackled, however, in order to advance the drive technology to production on a commercial scale. These include program code optimization, writing code to PROM, temperature rise consideration, vibration, and packaging.

In Section 11.8, basic DSP programming techniques necessary for induction motor control are presented by way of four examples with reference to the TMS320F28335 processor. These techniques enable the necessary feedback signals, stator voltages, stator currents, and rotor speed to be acquired and transferred to MATLAB® of a host PC in real-time for use by a complex intelligent algorithm.

References

Analog Devices (1992) Digital Signal Processing Applications using the ADSP-2100 Family, Volume 1, Analog Devices.

Ben-Brahim, L. and Kawamura, A. (1992) A fully digitized field-oriented controlled induction motor drive using only current sensors. *IEEE Transactions on Industrial Electronics*, **39**(3), 241–249.

Kubota, H., Matsuse, K., and Nakano, T. (1993) DSP-based speed adaptive flux observer of induction motor. *IEEE Transactions on Industry Applications*, **29**(2), 344–348.

Texas Instruments Inc. (2007) TMS320x2833x Analog-to-Digital Converter (ADC) Module Reference Guide, Literature Number: SPRU812A.

Texas Instruments Inc. (2008) C28x Floating Point Unit fast RTS Library, Module User's Guide, C28x Foundation Software.

Texas Instruments Inc. (2009a) TMS320F28335, TMS320F28334, TMS320F28332 Digital Signal Controllers (DSCs) Data Manual, Literature Number: SPRS439F.

Texas Instruments Inc. (2009b) TMS320C28x DSP/BIOS 5.x Application Programming Interface (API) Reference Guide, Literature Number: SPRU625K.

Texas Instruments Inc. (2009c) TMS320x2833x, 2823x Enhanced Pulse Width Modulator (ePWM) Module Reference Guide, Literature Number: SPRUG04A.

Texas Instruments Inc. (2009d) TMS320x2833x, 2823x System Control and Interrupts Reference Guide, Literature Number: SPRUFB0C.

Texas Instruments Inc. (2009e) TMS320x2833x, 2823x Enhanced Capture (eCAP) Module Reference Guide, Literature Number: SPRUFG4A.

The MathWorks Inc. (2008a) Embedded IDE Link™ 4, User's Guide, For Use with Texas Instruments' Code Composer Studio™.

Xu, X. and Novotny, W. (1996) Implementation of direct stator flux orientation control on a versatile DSP based system, in *Sensorless Control of AC Motor Drives* (eds K. Rajashekara, A. Kawamura, and K. Matsuse), IEEE Press, New Jersey.

Zhong, H.Y., Messinger, H.P., and Rashad, M.H. (1991) A new microcomputer-based direct torque control system for three-phase induction motor. *IEEE Transactions on Industry Applications*, **27**(2), 294–298.

12

Conclusions and Future Developments

This book has investigated the induction motor control problem from three perspectives, namely the theory level, algorithm design level, and hardware implementation level:

1. Modern nonlinear control theory and artificial intelligence principles form the basis of induction motor control. Development of a control theory should be based on an understanding of the physical system, from which an abstract mathematical model (principle or formulation) is derived.
2. Control algorithm of the drive system is based on an understanding of the control theory from which a calculation model is produced for achieving the desired performance. In the book, the intelligent control algorithms presented are based on modern nonlinear control theory as well as Kalman filter, expert-system, fuzzy-logic, artificial neural-network, and genetic algorithm principles.
3. Hardware implementation is based on the fine details of the control algorithm, from which a real physical model is constructed. An ADMC331 DSP-based experimental system has been constructed for verifying the fuzzy-logic control algorithm and GA-EKF speed estimation algorithm for an induction motor. ADSP-21xx Family Assembler and Linker are used for generating an executable program for the experimental investigations.

Based on an understanding of induction motor control and nonlinear feedback control theory, the induction motor control algorithms and speed estimation algorithms are presented systematically in Chapter 2.

This book has presented four intelligent induction motor control schemes mainly at the algorithm level, namely expert-system acceleration control, hybrid fuzzy/PI two-stage control, neural-network DSC control, and GA-EKF sensorless control. The expert-system is based on hard or precise computation, whereas the fuzzy-logic, neural-network, and genetic algorithm are based on soft or approximate computation (Bose, 1997). At the control algorithm level, this book has demonstrated that expert-system, fuzzy-logic, neural-network, and genetic algorithm are effective in different aspects of induction motor control. At the hardware level,

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

feasibility of the fuzzy/PI two-stage control and GA-EKF sensorless control has essentially been verified.

12.1 Main Contributions of the Book

The main contributions of the book are as follows:

1. The relationship between the 12 fifth-order nonlinear equations of an induction motor is investigated. Based on an understanding of induction motor control and nonlinear feedback control theory, the induction motor control algorithms and speed estimation algorithms are summarized systematically.
2. Three induction motor models are developed using the MATLAB®/Simulink software. They are the current-input model, voltage-input model, and discrete-state model. The current-input model requires the least amount of calculations and is suitable for the study of current-controlled induction motor drives. The voltage-input model involves matrix calculations and can be used to study voltage-controlled induction motor drive systems. The discrete-state model has the most compact representation, but is computationally intensive. In addition, a PWM model, an encoder model, and a decoder model have been developed.
3. Expert-system-based acceleration control principle and a production system algorithm are presented in this book. The control scheme is quite different from the usual vector control schemes that depend on flux and torque calculations. The expert-system controller has the following characteristics: (*1*) rotor acceleration of an induction motor is controlled, (*2*) it has a small control error but no cumulative error, (*3*) it is independent of the parameters of the induction motor, so the same controller can be used for different induction motors, (*4*) the control may be performed at any time and state, whereas conventional vector control must be continuously performed starting from an initial state, (*5*) its execution time should be less than that of the conventional vector-control, because the expert-system algorithm consists mainly of logic operations, and (*6*) since the acceleration values are obtained by a differential operation on the angular speed, some error may be produced by the speed sensor noise in a practical system.
4. A hybrid fuzzy/PI two-stage control algorithm is presented. The fuzzy-logic-based two-stage control strategy enables a scalar controller to give a performance approaching that of a field-oriented controller. Consequently, the rotor speed response is almost the same as a field-oriented controller. The fuzzy controller has the advantages of simplicity and robustness such as insensitivity to motor parameter changes, input current noise, noise in the measured speed, and magnetic saturation. Due to the excellent speed response over the whole speed range, the method should find applications in practical industrial drive systems.
5. A neural-network-based direct self control algorithm is presented. The neural-network-based DSC with seven layers of neurons is developed to improve the performance of a DSP-based direct self controller. The execution time of control is decreased from 250 μs (for the DSP-based controller) to 21 μs (for the ANN-based controller). Simulation results show that the torque and flux errors resulting from the long execution time are almost eliminated. Hence, the ANN-based DSC drive system has better robustness against current noise and load changes than a DSP-based DSC drive system. Using neural-network

techniques, hardware implementation of DSC presents less problems and it is envisaged that neural-network-based DSC will gain wider acceptance in future.

6. A novel method is presented to achieve good performance of an extended Kalman filter (EKF) for speed estimation of an induction motor drive. A real-coded genetic algorithm (GA) is used to optimize the noise covariance and weight matrices of the EKF, thereby ensuring filter stability and accuracy in speed estimation. Simulation studies on a constant V/Hz controller, a direct self controller, and a field-oriented controller demonstrate the efficacy of the presented method.
7. Integral equations of the induction motor are presented. Using the integral equations, a linear-neural-network model of induction motor may be trained with measured data from a running induction motor. Almost all the machine parameters can be derived directly from the trained neural network models. With the estimated parameters, load, stator flux, and rotor speed may be estimated for induction motor control. With the estimated rotor speed, a simulation programming example of integral model-based sensorless control of induction motor is presented.
8. Performance of conventional PWM inverter is improved by genetic algorithms. To reduce total harmonic distortion and to spread the harmonic energy of PWM inverter output waveform, four GA optimization strategies are presented in the book. They are (*1*) GA-optimized random-carrier-frequency PWM, (*2*) GA-optimized random-pulse-position PWM, (*3*) GA-optimized random-pulse-width PWM, and (*4*) GA-optimized hybrid random pulse-position and pulse-width PWM. A single-phase inverter is employed for the optimization study.
9. This book has presented five experiments to verify the intelligent control algorithms for an induction motor: (*1*) An experiment to determine the electrical parameters of an induction motor, (*2*) a motor run-up experiment to verify the induction motor model, PWM model, encoder model, and decoder model, (*3*) a DSP-based experiment for the fuzzy/PI two-stage controller to verify the feasibility of the fuzzy/PI two-stage control algorithm, (*4*) a GA-EKF experiment to verify the feasibility of the GA-EKF speed estimation algorithm, (*5*) GA-optimized single-phase random-carrier-frequency PWM inverter implemented by a TMS320F2812 DSP board and an IRAMX16UP60A inverter module.
10. Source codes of the programming examples and control algorithms presented in the book are included on the book companion website for easy reference by the reader. These program modules could form the basis for more advanced intelligent induction control applications that the readers may wish to investigate.

12.2 Industrial Application of New Induction Motor Drives

The demand for high-performance induction motor drives is rapidly increasing, particularly in the area of traction, electric vehicles, oil-drilling and aerospace applications. An ideal induction motor drive should have the following functions:

A. High-performance control
B. Performance should be unaffected by machine parameter variations
C. Possibility of speed-sensorless control
D. Low cost and fast dynamic response

Currently, vector control technique has only provided a practical solution for target A and parameter estimation technique may be a possible solution for target B, while speed and position estimation techniques are being developed to achieve target C.

Addressing the unresolved problems with present-day induction motor controllers, this book investigated and presented four artificial intelligence (AI) control techniques for the induction motor drive. They include an expert-system-based controller which gives a complete solution for targets A and B, a genetic algorithm optimized extended Kalman filter with high-precision speed estimation for a sensorless drive and an integral model based sensorless drive which give a practical solution for target C, and a low-cost neural-network-based vector controller will give a hardware solution for target D. In future, ANN-based induction motor controller may be integrated in several ASIC (application specific integrated circuit) chips with fast parallel calculation and low hardware cost to replace the present DSP (digital signal processor) based controller.

A comparison of various controllers on advantages, disadvantages, and cost is summarized in Table 12.1 that may serve as an application reference. The various controllers are designed to suit different application conditions, while their technical details could be found from the relevant chapters.

The constant V/Hz controller may be used in a low-performance drive, but it has a low control accuracy and slow transient response. On the other hand, FOC, DSC, and expert-system acceleration controller are suitable for high-performance applications and they have high control accuracy and a fast transient response. The fuzzy/PI two-stage controller has a performance that is intermediate between that of a constant V/Hz controller and FOC.

In practical industrial applications, induction motor drives are generally selected according to the desired performance and cost of the controllers. A high-performance controller (except the ANN-based controller) is currently implemented by a DSP device that will result in high cost. A low or medium performance controller is generally implemented by a dedicated microprocessor that helps to reduce the cost. Although the software implementation of low performance controller is easier compared with a high-performance controller, the hardware implementation of the two controllers presents almost the same level of complexity, because different a.c. drives employ basically the same components, such as an inverter, a controller, an induction motor, and some sensors.

In applications requiring a high-performance a.c. drive which is unaffected by machine parameter variations, such as electric vehicles and precision machine tools, the expert-system controller presented in the book should be considered.

When a low cost drive is desired, such as oil-drilling, civil-engineering equipment and pump, it is recommended to replace the current constant V/Hz controller by the fuzzy/PI two-stage controller developed.

Elimination of the speed sensor will increase the reliability and ruggedness of the overall drive system, hence the GA-EKF sensorless drive may be used for applications that require speed control in adverse environments or submerged drive systems.

The ANN-based induction motor controller has fast parallel performance and low cost. It is envisaged that when special ANN chips for motor control are available on the market, the ANN-based induction motor controller will gradually supersede the expensive DSP device for implementing an induction motor drive.

Table 12.1 Comparison of various induction motor controllers.

Controller	Advantages	Disadvantages	Cost
Constant V/Hz controller	• Stable	• Low-performance	• Low
		• Performance dependent on machine parameters	• Micro-processor based
Fuzzy controller	• Performance approximates that of FOC		• Low
	• Stable	• Medium-performance	• Micro-processor based
	• Performance is less dependent on machine parameters		
FOC and DSC	• High-performance	• Performance is dependent on machine parameters	• Higher
	• Stable	• Integral error accumulation	• DSP based
	• Flux and torque are controlled		
Expert system controller	• High-performance	• Performance is dependent on high-precision speed sensor	• Highest
	• Performance is independent of machine parameter variations		• DSP based and high precision encoder
	• No integral error accumulation		
ANN controller	• High-performance	• Performance is dependent on machine parameters	• Lowest
	• Flux and torque are controlled	• Integral error accumulation	• Neural device based
	• Shortest time delay of controller		
	• Robust structure		
Controller with GA-EKF estimator	• No speed sensor	• Performance is dependent on machine parameters	• Higher • DSP based
Controller with integral model estimator	• No speed sensor	• Performance is dependent on machine parameters	• Higher • DSP based

12.3 Future Developments

Future developments on intelligent control of induction motor are briefly discussed as follows:

12.3.1 Expert-System-based Acceleration Control

1. Although feasibility of the expert-system-based acceleration controller has been confirmed from computer simulation studies, it should be verified at the hardware level. The controller hardware may be implemented on a DSP (digital signal processor) with a high-precision encoder.
2. In order to improve the expert-system controller performance, the control rules and control knowledge could be refined by other AI techniques, such as Fuzzy logic, ANN or GA.

12.3.2 Hybrid Fuzzy/PI Two-Stage Control

1. To develop a current-magnitude fuzzy control in place of the PI control.
2. To develop a design method for the fuzzy controller that can accommodate the effect of load changes, disturbances, and parameter variations.
3. To improve the fuzzy/PI control algorithm at the computer simulation stage by using some AI techniques such as ANN-Fuzzy or GA-Fuzzy to optimize the membership functions instead of using the fairly standard and straightforward fuzzy method.

12.3.3 Neural-Network-based Direct Self Control

1. The neural-network-based DSC should be further improved in order to decrease the types and number of neurons.
2. A neural-network-based speed sensorless DSC can be developed by designing a sub-network for speed estimation.
3. To develop a neural-network controller whose performance is immune to disturbances and motor parameter variations.

12.3.4 Genetic Algorithm for an Extended Kalman Filter

Kalman filter technique has been widely used in different engineering disciplines. Further application of the GA-EKF method presented in this book need to be explored.

12.3.5 Parameter Estimation Using Neural Networks

Neural-network-based parameter estimation employs integral models of induction motor. The advantage of the integral models is that they are insensitive to measurement noise created by rapid turn-on or turn-off of the semiconductor switch devices. New control algorithm based on integral equations of induction motor and real-time parameter measurement based on reset-integral method need to be developed.

12.3.6 Optimized Random PWM Strategies Based on Genetic Algorithms

GA-optimized random PWM strategies will obviously reduce total harmonic distortion and boost energy converter efficiency for PWM inverters. Further work would be to extend the technique to three-phase PWM inverters that find wider applications in induction motor drives.

12.3.7 AI-Integrated Algorithm and Hardware

With the aid of modern nonlinear control theory and development of artificial intelligence techniques, induction motor drives will continue to evolve. It is envisaged that more advanced drives employing AI-integrated algorithm and ANN-integrated hardware will emerge, for example, a novel fuzzy-expert induction motor controller with GA-EKF speed estimator and implemented by ANN-integrated chips.

Reference

Bose, B.K. (1997) Intelligent control and estimation in power electronics and drives. The 1997 IEEE International Electric Machines and Drives Conference, Milwaukee, Wisconsin, USA, May 18–21.

Appendix A Equivalent Circuits of an Induction Motor

The 'Γ' equivalent circuit model of an induction motor in the stator reference-frame is shown in Figure A.1.

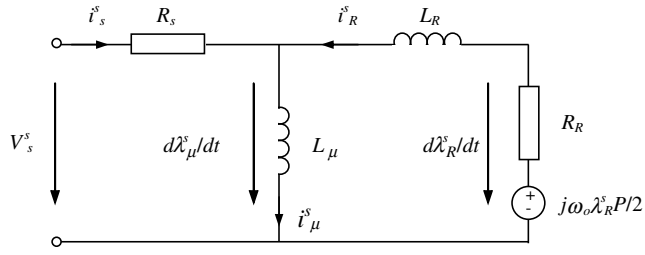


Figure A.1 'Γ' equivalent circuit model of an induction motor.

The 'T' equivalent circuit model of an induction motor in the stator reference-frame is shown in Figure A.2.

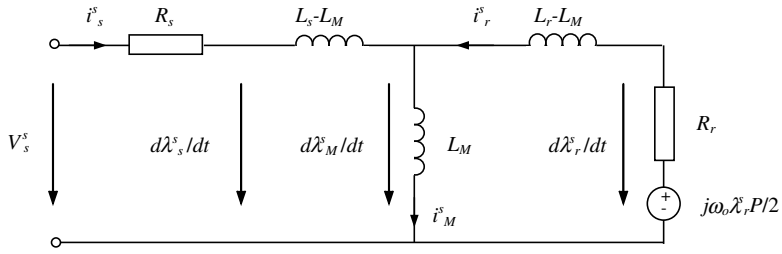


Figure A.2 'T' equivalent circuit model of an induction motor.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

Appendix B Parameters of Induction Motors

The parameters of the three induction motors used for the simulation studies in the book are listed below.

Table B.1 Motor 1.

Type	three-phase, wye-connected, squirrel-cage induction motor
Rated power	$KW_{rat} = 7.5\,\text{kW}$
Rated stator voltage	$V_{s,rat} = 220\,\text{V}$
Rated frequency	$f_{rat} = 60\,\text{Hz}$
Rated speed	$n_{M,rat} = 1160\,\text{r/min}$
Number of poles	$P = 6$
Stator resistance	$R_s = 0.282\,\Omega\text{/ph}$
Stator leakage reactance	$X_{ls} = 0.512\,\Omega\text{/ph}$
Rotor resistance referred to stator	$R_r = 0.151\,\Omega\text{/ph}$
Rotor leakage reactance referred to stator	$X_{lr} = 0.268\,\Omega\text{/ph}$
Magnetizing reactance	$X_m = 14.865\,\Omega\text{/ph}$
Moment of inertia of the rotor	$J_M = 0.4\,\text{kg m}^2$
Coefficient of friction	$C_f = 0.124$

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

Table B.2 Motor 2.

Type	three-phase, wye-connected, squirrel-cage induction motor
Rated power	$KW_{rat} = 0.75\,\text{kW}$
Rated stator voltage	$V_{s,rat} = 220\,\text{V}$
Rated frequency	$f_{rat} = 60\,\text{Hz}$
Rated speed	$n_{M,rat} = 1770\,\text{r/min}$
Number of poles	$P = 4$
Stator resistance	$R_s = 3.353\,\Omega/\text{ph}$
Stator leakage reactance	$X_{ls} = 1.073\,\Omega/\text{ph}$
Rotor resistance referred to stator	$R_r = 1.991\,\Omega/\text{ph}$
Rotor leakage reactance referred to stator	$X_{lr} = 1.073\,\Omega/\text{ph}$
Magnetizing reactance	$X_m = 1.029\,\Omega/\text{ph}$
Moment of inertia of the rotor	$J_M = 0.05\,\text{kg m}^2$

Table B.3 Motor 3 (Bodine Electric Company model 295).

Type	three-phase, wye-connected, squirrel-cage induction motor
Rated power	$kW_{rat} = 0.147\,\text{kW}$
Rated stator voltage	$V_{s,rat} = 230\,\text{V}$
Rated frequency	$f_{rat} = 60\,\text{Hz}$
Rated speed	$n_{M,rat} = 1790\,\text{r/min}$
Number of poles	$P = 4$
Stator resistance	$R_s = 14.6\,\Omega/\text{ph}$
Stator leakage reactance	$X_{ls} = 8.37\,\Omega/\text{ph}$
Rotor resistance referred to stator	$R_r = 12.76\,\Omega/\text{ph}$
Rotor leakage reactance referred to stator	$X_{lr} = 19.53\,\Omega/\text{ph}$
Magnetizing reactance	$X_m = 111.7\,\Omega/\text{ph}$
Moment of inertia of the rotor	$J_M = 0.001\,\text{kg m}^2$
Coefficient of friction	$C_f = 0.000124$

Appendix C M-File of Discrete-State Induction Motor Model

M-file: Induction_motor.m

```
function [sys,x0] = Induction_motor(t,x,u,flag,M)
global Tr Ts Lr Ls Lm K1 Kr Rs Rr J Pole;
Lr=0.0417; Ls=0.0424; Lm=0.041; Rs=0.294; Rr=0.156;
Tr=Lr/Rr; J=0.8;Pole=6;
K1=(1-Lm*Lm/Ls/Lr)*Ls;
Kr=Rs+Lm*Lm*Rr/Lr/Lr;

if flag==0
    x0=zeros(5,1);
    sys=[0,5,5,3,0,0];

elseif flag==2
    diff_FI=[(1-Kr/K1*M)*x(1)+Lm*Rr/Lr/Lr/K1*x(3)*M
    +Lm/Lr*x(5)/K1*x(4)*M*Pole/2;
      (1-Kr/K1*M)*x(2)-Lm/Lr*x(5)*M/K1*x(3) *Pole/2
    +Lm*Rr/Lr/Lr/K1*x(4)*M;
    Lm/Tr*x(1)*M+(1-1/Tr*M)*x(3)-x(5)*x(4)*M*Pole/2;
    Lm/Tr*x(2)*M+x(5)*x(3)*M*Pole/2+(1-1/Tr*M)*x(4);
    x(5)+Pole*Lm*(x(3)*x(2)-x(4)*x(1))*M/(J*Lr)/3-u(3)*M/J]
    +[u(1)/K1*M;u(2)/K1*M;0;0;0];
  sys=diff_FI;

elseif flag==3
    sys=x;

elseif flag==4
    sys=(round(t/M)+1)*M;

else
    sys=[];
    end
```

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

Appendix D Expert-system Acceleration Control Algorithm

The expert-system acceleration control algorithm is implemented by a production system (Buchanan and Shortliffe, 1984; Hayes-Roth, 1985). The knowledge base of the controller is represented by the set of production rules that represents the expertise of the control area. Case-specific data of the knowledge base are kept in the working memory of production system. Finally, the inference engine is implemented by the recognize-act cycle of the production system.

The following set of symbols is defined as:

a^*	acceleration command
a	actual acceleration
Δa	an increment of acceleration $\Delta a = a(t + \Delta t) - a(t)$
i	denotes area in which the stator current vector lies
n	a number which denotes a stator voltage vector $(mod(n) = 7)$
m	a temporary register of the stator voltage vector
t	time counter
t_h	retaining time
A_1	$a^* \gg 0$
A_2	$a^* \approx 0$
A_3	$a^* \ll 0$
B_1	$\Delta a(V_s(n)) > \eta \Delta a(V_s(n+1))$
B_2	$\Delta a(V_s(n)) \leq \eta \Delta a(V_s(n+1))$
B_3	$\Delta a(V_s(n)) > \eta \Delta a(V_s(n+2))$
B_4	$\Delta a(V_s(n)) \leq \eta \Delta a(V_s(n+2))$
B_5	$\Delta a(V_s(n)) < \eta \Delta a(V_s(n-1))$
B_6	$\Delta a(V_s(n)) \geq \eta \Delta a(V_s(n-1))$
B_7	$\Delta a(V_s(n)) < \eta \Delta a(V_s(n-2))$
B_8	$\Delta a(V_s(n)) \geq \eta \Delta a(V_s(n-2))$
C_1	$n = 0$ number of zero voltage vector
C_2	$n \neq 0$ nth number of non-zero voltage vector

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

D_1	$a \geq a^* + \varepsilon$
D_2	$a^* - \varepsilon < a < a^* + \varepsilon$
D_3	$a \leq a^* - \varepsilon$
E	$t \geq t_h$ (comparison stage)
¬E	$0 < t < t_h$ (retaining stage)
F	$\omega_o^* > 0$
¬F	$\omega_o^* < 0$
H	X_1, X_2 or X_3 has been performed when $t > t_h$
¬H	X_1, X_2 or X_3 has not been performed when $t > t_h$
S	$a^* = 0 \wedge \omega_o = 0$
$V_s(k)$	a stator voltage supplied to induction motor
X_1	Supply $V_s(n)$ and $V_s(n+1)$ to induction motor in succession
X_2	Supply $V_s(n)$ and $V_s(n+2)$ to induction motor in succession
X_3	Supply $V_s(n)$ and $V_s(n-1)$ to induction motor in succession
ω_o^*	rotor speed command
ω_o	actual rotor speed
ε	threshold of acceleration error
η	preferential parameter

The production system consists of the following set of rules:

Rule#Condition	Action	Note
1. $E \wedge \neg H \rightarrow n = i$		Detect region of current vector
2. $E \wedge A_1 \wedge \neg H \rightarrow X_1$		Input comparison voltages
3. $E \wedge A_2 \wedge \neg H \rightarrow X_2$		
4. $E \wedge A_3 \wedge \neg H \rightarrow X_3$		
5. $(E \wedge A_1 \wedge H \wedge B_1) \vee (E \wedge A_2 \wedge H \wedge B_3) \vee (E \wedge A_3 \wedge H \wedge B_5) \vee (E \wedge A_2 \wedge H \wedge B_7) \rightarrow k = n, t = 0$		Select an optimum voltage vector, and modify the rule base
6. $E \wedge A_1 \wedge H \wedge B_2 \rightarrow k = n+1, t = 0$		
7. $E \wedge A_2 \wedge F \wedge H \wedge B_4 \rightarrow k = n+2, t = 0$		
8. $E \wedge A_2 \wedge \neg F \wedge H \wedge B_8 \rightarrow k = n-2, t = 0$		
9. $E \wedge A_3 \wedge H \wedge B_6 \rightarrow k = n-1, t = 0$		
10. $(A_1 \wedge \neg E \wedge D_1 \wedge C_1) \vee (A_3 \wedge \neg E \wedge D_3 \wedge C_1) \rightarrow V_s(k)$		Keep zero voltage
11. $(A_1 \wedge \neg E \wedge D_1 \wedge C_2) \vee (A_3 \wedge \neg E \wedge D_3 \wedge C_2) \rightarrow m = k, k = 0, V_s(k)$		Insert zero voltage
12. $\neg E \wedge D_2 \rightarrow V_s(k)$		Retaining voltage
13. $(A_1 \wedge \neg E \wedge D_3 \wedge C_1) \vee (A_3 \wedge \neg E \wedge D_1 \wedge C_1) \rightarrow k = m, V_s(k)$		Restore optimum voltage
14. $(A_1 \wedge \neg E \wedge D_3 \wedge C_2) \vee (A_3 \wedge \neg E \wedge D_1 \wedge C_2) \rightarrow V_s(k)$		Keep optimum voltage

A production system is defined by (George, 1989):

1. ***The set of production rules.*** These are called *productions*. A production is a condition-action pair.

2. ***Working memory*** contains a description of the current state in a reasoning process. This description is a pattern that is matched against the condition part of a production to select appropriate problem-solving action.
3. ***The recognize-act cycle.*** The patterns in working memory are matched against the conditions of the production rules; this produces a subset of the productions, called the conflict set, whose conditions match the patterns in working memory. One of the productions in the conflict set is then selected (conflict resolution) and the production is fired. After the selected production rule is fired, the control cycle repeats with the modified working memory.

References

Buchanan, B. and Shortliffe, E.H. (1984) *Rule-Based Expert Systems*, MYCIN, Addison-Wesley, Reading, MA.
George, F.L. and William, A.S. (1989) *Artificial Intelligence and the Design of Expert System*, Benjamin/Cummings Publishing Company, Inc.
Hayes-Roth, F. (1985) *Building Expert Systems*, Addison-Wesley, Reading, MA.

Appendix E Activation Functions of Neural Network

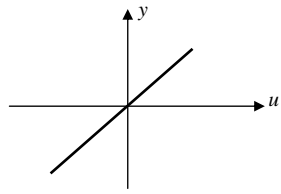


Figure E.1 Activation function of linear neuron: $y = u$.

$$y = \begin{cases} 0 & u < 0 \\ 1 & u \geq 0 \end{cases} \qquad y = \begin{cases} -1 & u < 0 \\ 1 & u \geq 0 \end{cases}$$

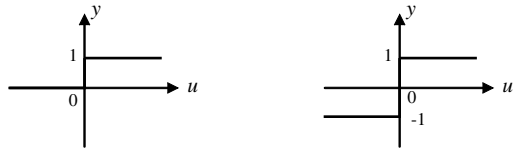


Figure E.2 Activation function of Hard Limit neuron.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

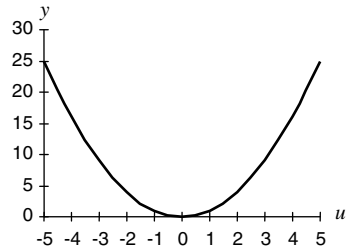


Figure E.3 Activation function of Square neuron: $y = u^2$.

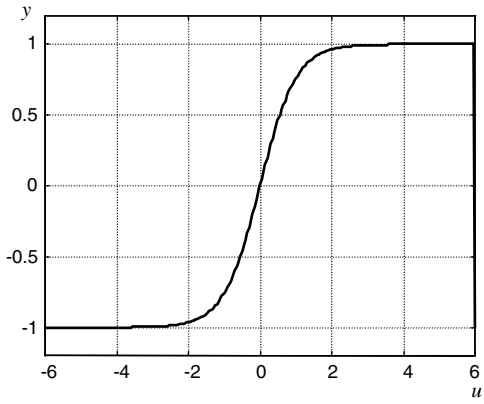


Figure E.4 Activation function of Tan-sigmoid neuron: $y = \tanh(u)$.

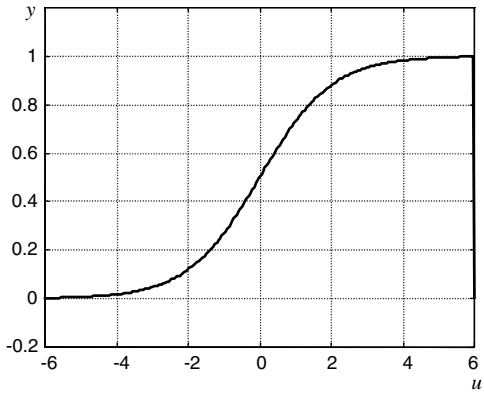


Figure E.5 Activation function of Log-sigmoid neuron: $y = \frac{1}{1+e^{-u}}$.

Appendix F M-File of Extended Kalman Filter[1]

```
function [sys,x0] = kalman(t,x,u,flag,T)
% input, u: Vds, Vqs, Ids, Iqs
% states, x: Ids, Iqs, Fids, Fiqs, speed
% output: Ids, Iqs, Fids, Fiqs, speed

%sample time: T
global Tr Ts Lr Ls Lh K1 Kr Rs Rr Q R GQG G x_1 P_1 K P h
Y out;

% Motor parameters
Lr=0.0417; Ls=0.0424; Lh=0.041; Rs=0.294; Rr=0.156; H_pole=6/2;Tr=Lr/Rr;
Kl=(1-Lh*Lh/Ls/Lr)*Ls; Kr=Rs+Lh*Lh*Rr/Lr/Lr;

% initial parameters and states
if flag==0
    x0=zeros(5,1);
    K=zeros(5,2);
    P=[1 0 0 0 0;
       0 1 0 0 0;
       0 0 1 0 0;
       0 0 0 1 0;
       0 0 0 0 1];
    G=[1e-10 0 0 0 0;
       0 1e-10 0 0 0;
       0 0 1e-10 0 0;
       0 0 0 1e-10 0;
       0 0 0 0 1e-10];
```

[1] Portions reprinted by permission of K.L. Shi, T.F. Chan, Y.K. Wong and S.L. Ho, "Speed estimation of induction motor using extended Kalman filter," IEEE 2000 Winter Meeting, vol. 1, pp. 243–248, January 23–27, 2000, Singapore. © 2000 IEEE.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

```
    Q=[1e-5 0 0 0 0;
       0 1e-5 0 0 0;
       0 0 1e-5 0 0;
       0 0 0 1e-5 0;
       0 0 0 0 1e-5];
    R=[0.001 0
       0 0.001];
    GQG=G*Q*G';
    sys=[0,5,5,4,0,0];

%Update discrete states
elseif abs(flag)==2

% Set measurement variable
     U=[u(1);u(2)]; Y=[u(3);u(4)];

% Equation (7.5): Prediction of state
    dif_F=[1-Kr/K1*T,0,Lh*Rr/Lr/Lr/K1*T,Lh/Lr*x(5)*H_pole/K1*T,Lh/Lr*x
(4)/K1*T;
      0,1-Kr/K1*T,-Lh/Lr*x(5)*H_pole/K1*T,Lh*Rr/Lr/Lr/K1*T,
      -Lh_K/Lr_K*x(3)/K1_K*T;
      Lh/Tr*T,0,1-T/Tr,-x(5)*H_pole*T,-x(4)*T;
      0,Lh/Tr*T,x(5)*H_pole*T,1-T/Tr,x(3)*T;
      0,0,0,0,1];
x_1=[ dif_F(1,1)*x(1)+dif_F(1,3)*x(3)+dif_F(1,4)*x(4);
      dif_F(2,2)*x(2)+dif_F(2,3)*x(3)+dif_F(2,4)*x(4);
      dif_F(3,1)*x(1)+dif_F(3,3)*x(3)+dif_F(3,4)*x(4);
      dif_F(4,2)*x(2)+dif_F(4,3)*x(3)+dif_F(4,4)*x(4);
      dif_F(5,5)*x(5)]+T*[u(1)/K1;u(2)/K1;0;0;0 ];

% Equation(7.7): Estimation of error covariance matrix
    GQG=GQG-dif_F*GQG*dif_F'*T;
    P_1=dif_F*P*dif_F'+GQG;

% Equation(7.14): Calculation of h and diff_h
    h=[x(1);x(2)]; dif_h=[1 0 0 0 0;0 1 0 0 0];

% Equation(7.9): Calculation of Kalman Filter
    K=P_1*dif_h'*inv(dif_h*P_1*dif_h'+R);

% Equation(7.11): Estimation of Kalman Filter
    out=x_1+K*(Y-h);
    sys=out;

% Equation(7.12): Update of the error covariance matrix
    P=P_1-K*dif_h*P_1;

elseif flag==3
       sys=out;

elseif flag==4
       sys=(round(t/T)+1)*T;

else
       sys=[];
end
```

Appendix G ADMC331-based Experimental System

Figure G.1 Complete experimental system with host PC and notebook computer for data acquisition.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

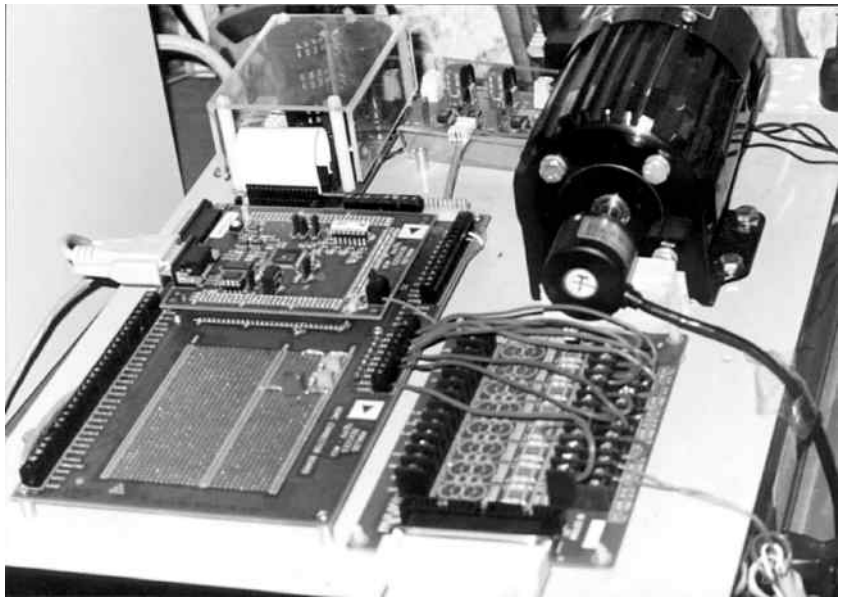


Figure G.2 Close-up view of the ADMC331 kit and the induction motor.

Appendix H Experiment 1: Measuring the Electrical Parameters of Motor 3

The motor is a three-phase 147-W, 230-V induction motor (Model 295 Bodine Electric Co.). The motor is Y-connected with no access to the neutral point.

1. **DC Resistance Test**
   As shown in Figure H.1, a DC voltage V_{DC} is applied so that the current I_{DC} is close to the motor rating.
   Because the machine is Y-connected, $R_s = R_{dc}/2 = (V_{DC}/I_{DC})/2$.
   From measurement, $V_{DC} = 30.6$ V, $I_{DC} = 1.05$ A.
   Hence,

$$R_s = \frac{R_{DC}}{2} = \frac{(30.6/1.05)}{2} = 14.6\,\Omega/\text{phase}.$$

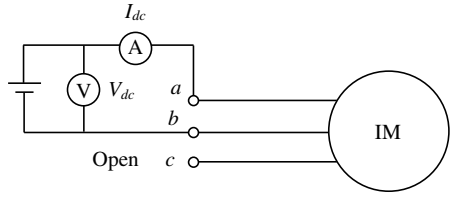


Figure H.1 Circuit for DC resistance test.

2. **No Load Test**
   The setup for the no-load test and locked-rotor test is shown in Figure H.2.

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

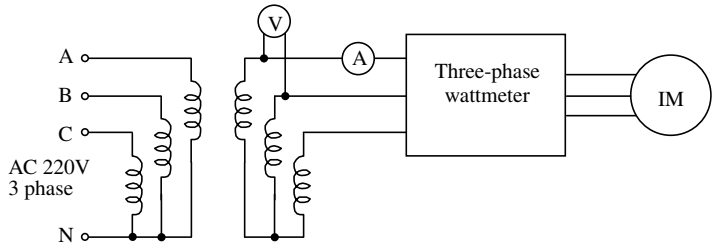


Figure H.2 Circuit for no-load and locked-rotor test.

With the motor running at no load, measure *V, I* & *P* to find the machine reactance $X_n \approx X_{ls} + X_m$. The measured data are as follows:

Frequency (Hz)	50
Voltage (V)	220
Current (A)	1.22
Real power (W)	128.2

At no load the per-unit slip s is approximately zero, hence the equivalent circuit is as shown in Figure H.3.

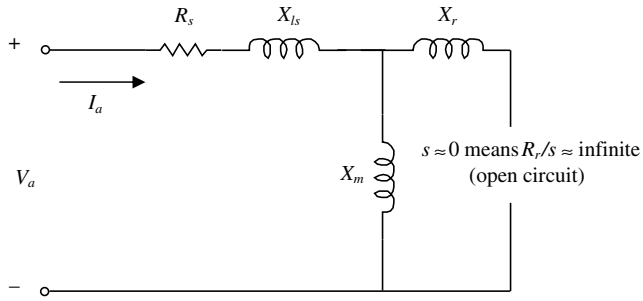


Figure H.3 Equivalent circuit of three-phase induction motor under no-load test.

The real power *P* represents,

1. Hysteresis and Eddy current losses (core losses)
2. Friction and windage losses (rotational losses)
3. Copper losses in stator and rotor (usually small at no load)

Phase voltage

$$V_a = \frac{V}{\sqrt{3}} = \frac{220}{\sqrt{3}} = 127\ V$$

Phase current

$$I_a = 1.22\,A$$

Phase real power

$$P_a = P/3 = 128.2 \div 3 = 42.73\,W$$

Phase reactive power

$$Q_a = \sqrt{(V_a I_a)^2 - P_a^2} = \sqrt{(127 \times 1.22)^2 - 42.73^2)} = 148.93\,VAr$$

$$X_n = \frac{Q_a}{I_a^2} = \frac{148.93}{1.22^2} = 100.06\,\Omega$$

Since $s \approx 0$,

$$X_n \approx X_{ls} + X_m.$$

3. Locked-rotor Test

With the rotor locked, the rotor speed is zero and the per-unit slip is equal to unity. The equivalent circuit is as shown in Figure H.4 or Figure H.5.

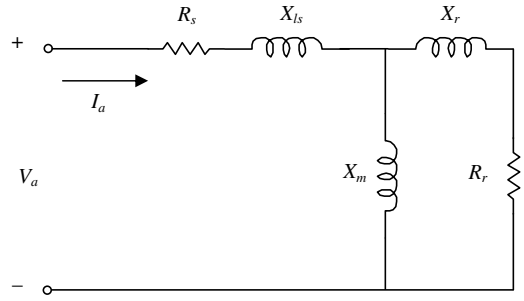


Figure H.4 Equivalent circuit of three-phase induction motor under locked-rotor test.

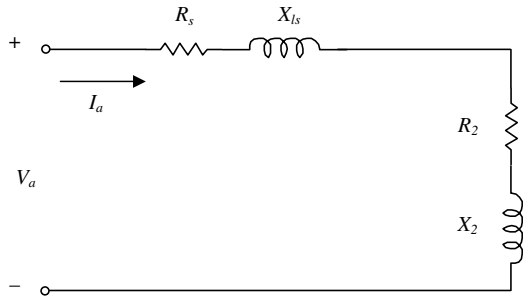


Figure H.5 Simplified equivalent circuit of three-phase induction motor under locked-rotor test.

The test data are:

Frequency (Hz)	50
Voltage (V)	69.32
Current (A)	1.2
Real power (W)	103.35

Phase voltage

$$V_a = \frac{V}{\sqrt{3}} = \frac{69.32}{\sqrt{3}} = 40.02\ V$$

Phase current

$$I_a = 1.2\ A$$

Active power per phase

$$P_a = \frac{P}{3} = \frac{103.35}{3} = 34.45\ W$$

Reactive power per phase

$$Q_a = \sqrt{(V_a I_a)^2 - P_a^2} = \sqrt{(40.02 \times 1.2)^2 - 34.45^2)} = 33.46\ VAr$$

For a class C motor,

$$X_{ls} = 0.3 \times \frac{Q_a}{I_a^2} = 0.3 \times \frac{33.46}{1.2^2} = 6.97\,\Omega$$

$$X_{lr} = 0.7 \times \frac{Q_a}{I_a^2} = 0.7 \times \frac{33.46}{1.2^2} = 16.27\,\Omega.$$

From the no-load test, $X_n = 100.06\,\Omega$, so

$$X_m = X_n - X_{ls} = 100.06 - 6.97 = 93.09\,\Omega$$

$$R = \frac{P_a}{I_a^2} = \frac{34.45}{1.2^2} = 23.92\,\Omega.$$

From Figure H.5,

$$R_2 = R - R_s = 23.92 - 14.6 = 9.32\,\Omega.$$

Comparing Figures H.4 and H.5,

$$R_2 + jX_2 = \frac{(R_r + jX_{lr}) \times jX_m}{(R_r + jX_{lr}) + jX_m}$$

$$R_2 \approx \frac{R_r X_m^2}{R_r + (X_{lr} + X_m)^2}$$

$$R_r = R_2 \times \left(\frac{X_{lr} + X_m}{X_m}\right)^2 = 9.32 \times \left(\frac{16.27 + 93.09}{93.09}\right)^2 = 12.86\,\Omega.$$

Summarizing,

1. Stator winding resistance $R_s = 14.6\,\Omega$/phase
2. Rotor winding resistance $R_r = 12.76\,\Omega$/phase
3. Magnetizing reactance $X_m = 93.09\,\Omega$/phase

The magnetizing inductance per phase is

$$L_m = \frac{X_m}{2\pi f} = \frac{93.09}{2\pi \times 50} = 0.2963\,H.$$

Stator leakage reactance $X_{ls} = 6.97\,\Omega$/phase
The stator inductance per phase is

$$L_{ls} = \frac{X_{ls}}{2\pi f} = \frac{6.97}{2\pi \times 50} = 0.0222\,H.$$

Rotor leakage reactance $X_{lr} = 16.27\,\Omega$/phase,
The rotor leakage inductance per phase is

$$L_{lr} = \frac{X_{lr}}{2\pi f} = \frac{16.27}{2\pi \times 50} = 0.0518\,H.$$

Appendix I DSP Source Code for the Main Program of Experiment 2

```
.MODULE/RAM/SEG=USER_PM1/ABS=0x30 main;
#include <c:\ADMC331\TgtFiles\admc331.h>
#include <c:\ADMC331\TgtFiles\romutil.h>
#include <c:\ADMC331\TgtFiles\macro.h>
#include <c:\ADMC331\TgtFiles\constant.h>
#include <dac.h>
#include <pwm331.h>

.VAR/DM/RAM/SEG=user_dm VphaseA;
.VAR/DM/RAM/SEG=user_dm ThetaA;
.VAR/DM/RAM/SEG=user_dm VphaseB;
.VAR/DM/RAM/SEG=user_dm ThetaB;
.VAR/DM/RAM/SEG=user_dm VphaseC;
.VAR/DM/RAM/SEG=user_dm ThetaC;

.ENTRY PWMSYNC_ISR;
.ENTRY PWMTRIP_ISR;

.VAR/DM/RAM/SEG=USER_DM Vdc;
.VAR/DM/RAM/SEG=USER_DM Vdc_max_inv;
.INIT Vdc_max_inv : Vdc_INV;

{ Phase increment constant :

    Desired frequency = 60 Hz
    PWM frequency = 10 kHz
    PWM cycles per period = 10000/60 =
    [-1,1] = [-pi, pi]
    phase increment = 2 * 60 * 32768 / 10000 = 393 }
.CONST delta = 393;

{ Initialization Code }
STARTUP:
```

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 ISBN: 978-0-470-82556-3

```
    ICNTL = 0x00; {Configure interrupt format:
        disable nested interrupts
        IRQ0,1,2 level sensitive }
    ay0 = 0x00E; {Enable s/w and SPORT1 interrupts for debugger }
        ar = IMASK;
        ar = ar OR ay0;
        IMASK = ar;

        { initialize DAC routines }
        call init_DAC;

        { initialize the PWM block }
        call init_PWM;

        { initialize phase angles }
        ar = 0x0000;
        dm(ThetaA) = ar;
        ay0 = TwoPiOverThree;
        ar = ar + ay0;
        dm(ThetaB) = ar;
        ar = ar + ay0;
        dm(ThetaC) = ar;

        IFC = 0x80; {Clear any pending IRQ2 interrupt }
        ay0 = 0x200; {Enable IRQ2 interrupts }
        ar = IMASK;
        ar = ar OR ay0;
        IMASK = ar;

        { do first DAC write to start autobuffer process }
        call update_DAC;
{ Main loop : empty, just waits for interrupts }

MAINLOOP:

        jump MAINLOOP;

PWMSYNC_ISR:
        IMASK = 0x06; { enable Sport1 interrupts for debugger }

        { increment phase angles }
        ax0 = dm(ThetaA);
        ay0 = delta;
        ar = ax0 + ay0;
        dm(ThetaA) = ar;

        ax0 = dm(ThetaB);
        ay0 = delta;
        ar = ax0 + ay0;
        dm(ThetaB) = ar;

        ax0 = dm(ThetaC);
        ay0 = delta;
        ar = ax0 + ay0;
        dm(ThetaC) = ar;
```

```
        { calculate phase voltages by taking sine of angleand then
        mulitplying by 0.5 to put on the range[-0.5, 0.5] }
        ax0 = dm(ThetaA);
        M5 = 1;
        L5 = 0;
        call ADMC_SIN;
        my1 = ar;
        mr = 0;
        mx0 = 0x4000;
        mr = mr + mx0 * my1 (SS);
        dm(VphaseA) = mr1;

        ax0 = dm(ThetaB);
        M5 = 1;
        L5 = 0;
        call ADMC_SIN;
        my1 = ar;
        mr = 0;
        mx0 = 0x4000;
        mr = mr + mx0 * my1 (SS);
        dm(VphaseB) = mr1;

        ax0 = dm(ThetaC);
        M5 = 1;
        L5 = 0;
        call ADMC_SIN;
        my1 = ar;
        mr = 0;
        mx0 = 0x4000;
        mr = mr + mx0 * my1 (SS);
        dm(VphaseC) = mr1;

        { write to PWM generator }
        sr1 = dm(VphaseA);
        ay1 = dm(VphaseB);
        my1 = dm(VphaseC);
        call WR_PWM_DUTY;
        call read_ADC;
        dm(Vdc) = ar;

        call Vdc_HANDLING;

        { write to DAC }
        write_DAC(Vdc_as, DAC1)
        write_DAC(Vdc_bs, DAC2)
        write_DAC(Vdc_cs, DAC3)

        rti;
PWMTRIP_ISR:

    imask = 0x06; { unmask SPORT1 interrupts for debugger}

    { check the PWMTRIP input, if it has gone high restart the PWM block }

    AR = DM(SYSSTAT);
```

```
    AR = TSTBIT 0 of AR;
    IF NE JUMP RESTART_PWM;

    CNTR = H#3FF ;
    DO Wait0 UNTIL CE; { wait 10us }
Wait0: NOP;

    { check the PWMTRIP input again, if it has gone high restart the PWM block }
    AR = DM(SYSSTAT);
    AR = TSTBIT 0 of AR;
    IF NE JUMP RESTART_PWM;

    RTI;

{ After a shutdown - restart the PWM. }

RESTART_PWM:

    CNTR = H#3FF ;
    DO Wait20 UNTIL CE; { wait 10us }
Wait20: NOP;

    IFC = 0X80; { clear IRQ2 interupt }
    AR = DM(IRQFLAG);
    call INIT_PWM;
    RTI;
{ Measure the bus-voltage for correction of the phase voltages. }

Vdc_HANDLING:

    { calculate Vdc(measured) / Vdc(max) }

    MX0 = DM(Vdc_max_inv);
    MY0 = DM(Vdc);
    MR = MX0*MY0 (SS);

    { Vdc_as = VphaseA * Vdc(measured) / Vdc(max) }
    MX0 = MR1;
    MY0 = DM(VphaseA);
    MR = MR1*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Vdc_as) = SR1;

    { Vdc_bs = VphaseB * Vdc(measured) / Vdc(max) }
    MY0 = DM(VphaseB);
    MR = MX0*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Vdc_bs) = SR1;
    { Vdc_cs = VphaseC * Vdc(measured) / Vdc(max) }
    MY0 = DM(VphaseC);
    MR = MX0*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Vdc_cs) = SR1;
    RTS;
.ENDMOD;
```

Appendix J DSP Source Code for the Main Program of Experiment 3

```
.MODULE/RAM/SEG=USER_PM1/ABS=0x30  main;
#include <c:\adi\mcd331\TgtFiles\admc331.h>;
#include <c:\adi\mcd331\TgtFiles\macro.h>;
#include <c:\adi\mcd331\TgtFiles\romutil.h>
{Local program variable definitions}
{ Three phase stator currents }
.VAR/DM/RAM/SEG=USER_DM Ias;
.VAR/DM/RAM/SEG=USER_DM Ibs;
.VAR/DM/RAM/SEG=USER_DM Ics;
.VAR/DM/RAM/SEG=user_dm ThetaA;
.VAR/DM/RAM/SEG=user_dm ThetaB;
.VAR/DM/RAM/SEG=user_dm ThetaC;

{ Phase current offsets, measured at startup }
.VAR/DM/RAM/SEG=USER_DM Ia_offset;
.VAR/DM/RAM/SEG=USER_DM Ib_offset;
.VAR/DM/RAM/SEG=USER_DM Ic_offset;

{ DC bus voltage: actual and 1/max }
.VAR/DM/RAM/SEG=USER_DM Vdc;
.VAR/DM/RAM/SEG=USER_DM Vdc_max_inv;

{ Three reference voltages }
.VAR/DM/RAM/SEG=USER_DM VrefA;
.VAR/DM/RAM/SEG=USER_DM VrefB;
.VAR/DM/RAM/SEG=USER_DM VrefC;

.VAR/DM/RAM/SEG=USER_DM Omega_set;

{ Real time clock variables }
.VAR/DM/RAM/SEG=USER_DM Count;
.VAR/DM/RAM/SEG=USER_DM Time;
```

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3

```
{ Flag indicating whether first PWMSYNC has occurred }
.VAR/DM/RAM/SEG=USER_DM firstFlag;

{ Direction flag }
.VAR/DM/RAM/SEG=USER_DM dir_flag;
.CONST CW_FLAG = 0x01;
.CONST CCW_FLAG = 0x00;
{ for debugging only }
.VAR/DM/RAM/SEG=USER_DM StepTime;

{ Local program variable initialization }

.INIT Vdc_max_inv : Vdc_INV;
.INIT StepTime : 1; { time to apply speed command from potentiometer, in 0.1s
increments }

{ Subroutines defined in this module }

.ENTRY PWMSYNC_ISR;
.ENTRY PWMTRIP_ISR;

{PWM frequency = 10 kHz
2 * pi * time increment = 2 * 32768 / 10000 = 7}

.CONST delta = 7;

{ Subroutines defined in other modules  }

#include <dac.h>
#include <admc331.h>
#include <pwm331.h>
#include <ir_init.h>
#include <model.h>
#include <mathfix.h>
#include <math_32b.h>
#include <enco.h>
#include <cntrl.h>
#include <capture.h>
#include <speedset.h>

{ Global variable definitions  }

.GLOBAL VrefA;
.GLOBAL VrefB;
.GLOBAL VrefC;
.GLOBAL Omega_set;
.GLOBAL ImRef;
.GLOBAL Freq;
.GLOBAL Fslip
.GLOBAL IaRef;
.GLOBAL IbRef;
.GLOBAL IcRef;

{ Initialization Code }
```

```
STARTUP:
    ay0 = 0x00E;  {Enable s/w, and SPORT1 interrupts }
    ar = IMASK;
    ar = ar OR ay0;
    IMASK = ar;
  ICNTL = 0x00; {Configure interrupt format: disable nested interrupts
  IRQ0,1,2 level sensitive }

    { set PIO(23) as output }
    ax0 = 0x80;
    dm(PIODIR2) = ax0;
    ax0 = 0x00;
    dm(PIODATA2) = ax0;

    { set time to 0 }
    ax0 = 0x0000;
    dm(Time) = ax0;

    call init_speed_est;

    { initialize the data capture functions: number of samples = 100
        1 sample per 200 PWM cycles
        1 sample per 20 ms
        50 Hz sample rate
        2s of total time }

    call init_capture;
    ar = 99;{199;}
    call init_undersample;

    { initialize current magnitude PI controllers }
    call init_ MCPI_Controller;

    { initialize fuzzy frequency controllers }
    call init_ FUZZY_Controller;

    { initialize stator current PI controllers }
    call init_SCPI_Controllers;

    { initialize the IR PowIRtrain module }
    call init_IR;

    { initialize the DAC }
    call init_DAC;

    { initialize the ADC }
    call init_ADC;

    { initialize the encoder }
    call init_enco;

    { initialize a counter to measure current offsets }
    ax0 = 16;
    dm(Count) = ax0;
    ax0 = 0x0001;
```

```
    dm(firstFlag) = ax0;
    ax0 = 0x0000;
    dm(Ia_Offset) = ax0;
    dm(Ib_Offset) = ax0;
    dm(Ic_Offset) = ax0;

    { initialize phase angles }
    ar = 0x0000;
    dm(ThetaA) = ar;
    ay0 = TwoPiOverThree;
    ar = ar + ay0;
    dm(ThetaB) = ar;
    ar = ar + ay0;
    dm(ThetaC) = ar;

    { initialize counters used for a real time clock subsystem }
    ax0 = 0x0000;
    dm(Count_01s) = ax0;
    dm(Time) = ax0;

  { initialize the speed setpoint functions }
    call init_speed;

    call INIT_PWM; {Initialize PWM registers and ISRs }

    IFC = 0x80;  {Clear any pending IRQ2 interrupt }
    ay0 = 0x200;  {Enable IRQ2 interrupts }
    ar = IMASK;
    ar = ar OR ay0;
    IMASK = ar;

    call update_DAC;

MAINLOOP:
    ax0 = dm(Time);
    ay0 = dm(StepTime);
    ar = ax0 - ay0;
    if ne jump MAINLOOP;
    ar = ax0 + 1;     { increment time }
    dm(Time) = ar;

    { enable capture }
    call enable_A;
    call enable_B;
    call enable_C;
    call enable_D;

    { do initial capture }
    call record;

    { apply apply speed command request from
     potentiometer}
    ax0 = dm(Omega_set);
    ay0 = dm(speedset);
    ar = ax0 + ay0;
    dm(Omega_set) = ar;
```

```
   jump MAINLOOP;

{ end of main loop }

PWMSYNC_ISR:      { PWM Interrupt Service Routine }
    imask = 0x20E;{unmask IRQ2, s/w, SPORT1 interrupts}
                  { IRQ2 used for tacho or resolver pulses }
                  { s/w, SPORT1 used for debugger }
    ena sec_reg;
    ax0 = dm(Count_0s);
    ar = ax0 + 1;
    ay0 = 1000;
    af = ax0 - ay0;
    if eq jump tic;
    dm(Count_0s) = ar;
    jump cont3;

tic:  ax0 = 0x0000;
    dm(Count_0s) = ax0;
    ax0 = dm(Time);     { increment Time }
    ar = ax0 + 1;
    dm(Time) = ar;

cont3:  nop;
    { do nothing in the first PWMSYNC cycle since the
     ADCs do not yet have valid values. }
    ar = dm(firstFlag);
    ar = pass ar;
    if eq jump not1st;
    ar = 0x0000;
    dm(firstFlag) = ar;
    jump end_PWM;

    { in PWMSYNC cycles 2 through 17 measure the current offsets. }
not1st:  ar = dm(Count);
    ar = pass ar;
    if eq jump cont4;
    call read_ADC;
    call Calc_I_Offsets;
    jump end_PWM;

cont4:  call read_ADC;
    dm(Vdc) = ar;

    call Vdc_HANDLING;

    { stator current PI Controllers }
{ decide if it is time to execute the fuzzy and current magnitude
controllers }
    ar = dm(Vel_count);
    af = pass ar;
    if le jump do_speed;
    ar = ar - 1;
    dm(Vel_count) = ar;
    jump CHECK_Vabc;
```

```
do_speed:
    ar = VEL_COUNT_MAX-1;
    dm(Vel_count) = ar;
    { speed encoder}
    call enco;
    call Fuzzy;
    call Currentmc;
    call SINCURRENT;
    call Stcpi;

CHECK_Vabc:
    SR1 = DM(Vac_as);
    MR1 = DM(Vac_bs);
    MR2 = DM(Vac_cs);

{ Calculate the three phase-voltages. }
    DM(VrefA) = SR1;
    DM(VrefB) = MR1;
    DM(VrefC) = MR2;
    JUMP V_CW_CCW;

{ Limit the voltages before calculating the duty-cycles }
    CALL LIMIT_VrefABC;
{ Update PWM duty cycles based on calculated VrefA, B, C}
    SR1 = DM(VrefA);
    AY1 = DM(VrefB);
    MY1 = DM(VrefC);

    call WR_PWM_DUTY;

    call calc_enco;
    write_DAC(VrefA, DAC1)
    write_DAC(VrefB, DAC2)
    write_DAC(VrefC, DAC3)
    write_DAC(Fslip, DAC4)
    write_DAC(ImRef, DAC5)
    write_DAC(Ias, DAC6)
    write_DAC(Ibs, DAC7)
    write_DAC(Ics, DAC8)

    call record;

end_PWM:
    dis sec_reg;
    rti;

SINCURRENT:
        ax0 = dm(ThetaA);
        ay0 = delta * dm(Freq);
        ar = ax0 + ay0;
        dm(ThetaA) = ar;

        ax0 = dm(ThetaB);
        ay0 = delta * dm(Freq);
```

```
        ar = ax0 + ay0;
        dm(ThetaB) = ar;

        ax0 = dm(ThetaC);
        ay0 = delta * dm(Freq);
        ar = ax0 + ay0;
        dm(ThetaC) = ar;

        ax0 = dm(ThetaA);
        M5 = 1;
        L5 = 0;
        call ADMC_SIN;
        my1 = ar;
        mr = 0;
        mr = mr + my1 * dm(ImRef);
        dm(IaRef) = mr;

        ax0 = dm(ThetaB);
        M5 = 1;
        L5 = 0;
        call ADMC_SIN;
        my1 = ar;
        mr = 0;
        mr = mr + my1 * dm(ImRef);
        dm(IbRef) = mr;

        ax0 = dm(ThetaC);
        M5 = 1;
        L5 = 0;
        call ADMC_SIN;
        my1 = ar;
        mr = 0;
        mr = mr + my1 * dm(ImRef);
        dm(IcRef) = mr;

{ Subroutine : record
Description : record variables of interest in buffers using the routines
defined in capture.dsp.}
record:
    ar = dm(Fslip);
    call capture_A;

    ar = dm(ImRef);
    call capture_B;

    ar = dm(Enco);
    call capture_C;

    ar = dm(IasRef);
    call capture_D;

    rts;

{     Calc_I_Offsets - measure the three phase current offsets. This code is
only executed for the first 16 PWM cycles.}
```

```
Calc_I_Offsets:
    ay1 = ar;
    { store the measured values as the offsets }
    ay0 = dm(Ia_offset);        { load old Ia_offset }
    SR = ASHIFT sr1 BY -4 (LO); { divide latest reading by 16 }

    ar = sr0 + ay0;             { keep running total }
    dm(Ia_offset) = ar;

    ay0 = dm(Ib_offset);        { load old Ib_offset }
    SR = ASHIFT sr1 BY -4 (LO); { divide latest reading by 16 }

    ar = sr0 + ay0;             { keep running total }
    dm(Ib_offset) = ar;
    ay0 = dm(Ic_offset);        { load old Ic_offset }
    SR = ASHIFT mr1 BY -4 (LO); { divide latest reading by 16 }

    ar = sr0 + ay0;             { keep running total }
    dm(Ic_offset) = ar;

    { decrement the count value }
skip:ar = dm(Count);
    ar = ar - 1;
    dm(Count) = ar;

    rts;

{ PWMTRIP - stops the PWM on the outputs.}
PWMTRIP_ISR:
    imask = 0x0E;  { unmask s/w, SPORT1 interrupts for debugger }
    { check the PWMTRIP input, if it has gone high restart the PWM block }
    AR = DM(SYSSTAT);
    AR = TSTBIT 0 of AR;
    IF NE JUMP RESTART_PWM;

    CNTR = H#3FF ;
    DO Wait0 UNTIL CE;  { wait 10us }
Wait0:  NOP;

    { check the PWMTRIP input again, if it has gone high restart the PWM block }

    AR = DM(SYSSTAT);
    AR = TSTBIT 0 of AR;
    IF NE JUMP RESTART_PWM;
    DIS SEC_REG;

    RTI;

{   After a shutdown - restart the PWM.     }
RESTART_PWM:

    CNTR = H#3FF ;
    DO Wait20 UNTIL CE;  { wait 10us }

Wait20:  NOP;
```

```
    IFC = 0X80;    { clear IRQ2 interupt }
    AR = DM(IRQFLAG);

    call INIT_PWM;

    RTI;          {Return From Interrupt}

CURRENT_HANDLING:
    AX1 = MR1;      { save Ics }
    MY0 = Vi_Scale;    { scaling factor }

    { Calculate Ias }

    AY1 = DM(Ia_offset);
    AR = SR1 - AY1;    { SR1 = Ias_ADC }
    MR = AR*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Ias) = sr1;
    AR = -SR1;
    AX0 = AR;

    { Calculate Ibs }

    AY1 = DM(Ib_offset);
    AR = SR1 - AY1;    { SR1 = Ibs_ADC }
    MR = AR*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Ibs) = sr1;
    AR = -SR1;
    AX0 = AR;

    { Calculate Ics }

    AY1 = DM(Ic_offset);
    AR = AX1 - AY1;    { AX1 = Ics_ADC }
    MR = AR*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    AY1 = SR1;
    DM(Ics) = SR1;

{   Measure the bus-voltage for correction of the phase voltages.  }
Vdc_HANDLING:

    { calculate Vdc(measured) / Vdc(max) }

    MX0 = DM(Vdc_max_inv);
    MY0 = DM(Vdc);
    MR = MX0*MY0 (SS);

    { Vac_as = VrefA * Vdc(measured) / Vdc(max) }

    MX0 = MR1;
    MY0 = DM(VrefA);
```

```
    MR = MR1*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Vac_as) = SR1;

    { Vac_bs = VrefB * Vdc(measured) / Vdc(max) }
    MY0 = DM(VrefB);
    MR = MX0*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Vac_bs) = SR1;

    { Vac_cs = VrefC * Vdc(measured) / Vdc(max) }

    MY0 = DM(VrefC);
    MR = MX0*MY0 (SS);
    SR = ASHIFT MR1 BY 2 (HI);
    DM(Vac_cs) = SR1;

    RTS;

{ Limiting of the Va,Vb and Vc, before calculating the duty cycle. }
LIMIT_VrefABC:

    AY0 = Vmax;    { limit value }
    SR1 = DM(VrefA);
    AR = ABS SR1;
    AF = AR - AY0;
    IF GT AR = PASS AY0;
    AF = PASS SR1;
    IF LT AR = -AR;
    DM(VrefA) = AR;

    SR1 = DM(VrefB);
    AR = ABS SR1;
    AF = AR - AY0;
    IF GT AR = PASS AY0;
    AF = PASS SR1;
    IF LT AR = -AR;
    DM(VrefB) = AR;

    SR1 = DM(VrefC);
    AR = ABS SR1;
    AF = AR - AY0;
    IF GT AR = PASS AY0;
    AF = PASS SR1;
    IF LT AR = -AR;
    DM(VrefC) = AR;

    RTS; {Return From Subroutine}

.ENDMOD;.ENDMOD;
```

Index

Applied Intelligent Control of Induction Motor Drives, First Edition. Tze-Fun Chan and Keli Shi.
 Published 2011 by John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82556-3