

THE UNIVERSITY OF CALGARY

**ARTIFICIAL NEURAL NETWORK FLUX ESTIMATION FOR
FIELD ORIENTED CONTROL**

BY

Allan K.P. Toh

A THESIS

**SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE**

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

July, 1994

© Allan K.P. Toh 1994



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-315-99509-2

Name ALLAN K.P. TOH

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ELECTRONICS & ELECTRICAL

SUBJECT TERM

0544

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
Ancient 0289
Linguistics 0290
Modern 0291
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce-Business 0510
Finance 0511
History 0358
Labor 0366
Theory 0351
Folklore 0366
Geography 0351
Gerontology 0578
History 0578
General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography 0628
Ethnic and Racial Studies 0629
Individual and Family Studies 0630
Industrial and Labor Relations 0700
Public and Social Welfare 0344
Social Structure and Development 0709
Theory and Methods 0999
Transportation 0453
Urban and Regional Planning 0453
Women's Studies

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and Nutrition 0476
Animal Pathology 0359
Food Science and Technology 0478
Forestry and Wildlife 0479
Plant Culture 0480
Plant Pathology 0817
Plant Physiology 0777
Range Management 0746
Wood Technology 0306
Biology 0287
General 0308
Anatomy 0309
Biostatistics 0379
Botany 0329
Cell 0353
Ecology 0369
Entomology 0793
Genetics 0410
Limnology 0307
Microbiology 0317
Molecular 0416
Neuroscience 0433
Oceanography 0821
Physiology 0778
Radiation 0472
Veterinary Science 0786
Zoology 0760
Biophysics 0425
General 0996
Medical

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology 0354
Occupational Health and Therapy 0381
Ophthalmology 0571
Pathology 0419
Pharmacology 0572
Pharmacy 0382
Physical Therapy 0573
Public Health 0574
Radiology 0575
Recreation

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
General 0486
Agricultural 0487
Analytical 0488
Biochemistry 0738
Inorganic 0490
Nuclear 0491
Organic 0494
Pharmaceutical 0495
Physical 0754
Polymer 0405
Mathematics 0605
Physics 0986
General 0606
Acoustics 0608
Astronomy and Astrophysics 0748
Atmospheric Science 0607
Atomic 0798
Electronics and Electricity 0759
Elementary Particles and High Energy 0609
Fluid and Plasma 0610
Molecular 0752
Nuclear 0756
Optics 0611
Radiation 0463
Solid State 0346
Statistics 0984
Applied Sciences

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology

PSYCHOLOGY

General 0621
Behavioral 0384
Fluid and Thermodynamics 0622
Clinical 0620
Developmental 0623
Experimental 0624
Industrial 0625
Personality 0989
Physiological 0349
Psychobiology 0632
Psychometrics 0451
Social



Nom _____

Dissertation Abstracts International est organisé en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.



SUJET

CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS

Architecture	0729
Beaux-arts	0357
Bibliothéconomie	0399
Cinéma	0900
Communication verbale	0459
Communications	0708
Danse	0378
Histoire de l'art	0377
Journalisme	0391
Musique	0413
Sciences de l'information	0723
Théâtre	0465

ÉDUCATION

Généralités	515
Administration	0514
Art	0273
Collèges communautaires	0275
Commerce	0688
Economie domestique	0278
Éducation permanente	0516
Éducation préscolaire	0518
Éducation sanitaire	0680
Enseignement agricole	0517
Enseignement bilingue et multiculturel	0282
Enseignement industriel	0521
Enseignement primaire	0524
Enseignement professionnel	0747
Enseignement religieux	0527
Enseignement secondaire	0533
Enseignement spécial	0529
Enseignement supérieur	0745
Évaluation	0288
Finances	0277
Formation des enseignants	0530
Histoire de l'éducation	0520
Langues et littérature	0279

Lecture	0535
Mathématiques	0280
Musique	0522
Orientation et consultation	0519
Philosophie de l'éducation	0998
Physique	0523
Programmes d'études et enseignement	0727
Psychologie	0525
Sciences	0714
Sciences sociales	0534
Sociologie de l'éducation	0340
Technologie	0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE

Langues	
Généralités	0679
Anciennes	0289
Linguistique	0290
Modernes	0291
Littérature	
Généralités	0401
Anciennes	0294
Comparée	0295
Médiévale	0297
Moderne	0298
Africaine	0316
Américaine	0591
Anglaise	0593
Asiatique	0305
Canadienne (Anglaise)	0352
Canadienne (Française)	0355
Germanique	0311
Latino-américaine	0312
Moyen-orientale	0315
Romane	0313
Slave et est-européenne	0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE

Philosophie	0422
Religion	
Généralités	0318
Clergé	0319
Études bibliques	0321
Histoire des religions	0320
Philosophie de la religion	0322
Théologie	0469

SCIENCES SOCIALES

Anthropologie	
Archéologie	0324
Culturelle	0326
Physique	0327
Droit	0398
Economie	
Généralités	0501
Commerce-Affaires	0505
Economie agricole	0503
Economie du travail	0510
Finances	0508
Histoire	0509
Théorie	0511
Études américaines	0323
Études canadiennes	0385
Études féministes	0453
Folklore	0358
Géographie	0366
Gérontologie	0351
Gestion des affaires	
Généralités	0310
Administration	0454
Banques	0770
Comptabilité	0272
Marketing	0338
Histoire	
Histoire générale	0578

Ancienne	0579
Médiévale	0581
Moderne	0582
Histoire des noirs	0328
Africaine	0331
Canadienne	0334
États-Unis	0337
Européenne	0335
Moyen-orientale	0333
Latino-américaine	0336
Asie, Australie et Océanie	0332
Histoire des sciences	0585
Loisirs	0814
Planification urbaine et régionale	0999
Science politique	
Généralités	0615
Administration publique	0617
Droit et relations internationales	0616
Sociologie	
Généralités	0626
Aide et bien-être social	0630
Criminologie et établissements pénitentiaires	0627
Démographie	0938
Études de l'individu et de la famille	0628
Études des relations interethniques et des relations raciales	0631
Structure et développement social	0700
Théorie et méthodes	0344
Travail et relations industrielles	0629
Transports	0709
Travail social	0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES

Agriculture	
Généralités	0473
Agronomie	0285
Alimentation et technologie alimentaire	0359
Culture	0479
Élevage et alimentation	0475
Exploitation des paturages	0777
Pathologie animale	0476
Pathologie végétale	0480
Physiologie végétale	0817
Sylviculture et faune	0478
Technologie du bois	0746
Biologie	
Généralités	0306
Anatomie	0287
Biologie (Statistiques)	0308
Biologie moléculaire	0307
Botanique	0309
Cellule	0379
Ecologie	0329
Entomologie	0353
Génétique	0369
Limnologie	0793
Microbiologie	0410
Neurologie	0317
Océanographie	0416
Physiologie	0433
Radiation	0821
Science vétérinaire	0778
Zoologie	0472
Biophysique	
Généralités	0786
Médicale	0760

SCIENCES DE LA TERRE

Biogéochimie	0425
Géochimie	0996
Géodésie	0370
Géographie physique	0368

Géologie	0372
Géophysique	0373
Hydrologie	0388
Minéralogie	0411
Océanographie physique	0415
Paléobotanique	0345
Paléoécologie	0426
Paléontologie	0418
Paléozoologie	0985
Palynologie	0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT

Économie domestique	0386
Sciences de l'environnement	0768
Sciences de la santé	
Généralités	0566
Administration des hôpitaux	0769
Alimentation et nutrition	0570
Audiologie	0300
Chimiothérapie	0992
Dentisterie	0567
Développement humain	0758
Enseignement	0350
Immunologie	0982
Loisirs	0575
Médecine du travail et thérapie	0354
Médecine et chirurgie	0564
Obstétrique et gynécologie	0380
Ophtalmologie	0381
Orthophonie	0460
Pathologie	0571
Pharmacie	0572
Pharmacologie	0419
Physiothérapie	0382
Radiologie	0574
Santé mentale	0347
Santé publique	0573
Soins infirmiers	0569
Toxicologie	0383

SCIENCES PHYSIQUES

Sciences Pures	
Chimie	
Généralités	0485
Biochimie	0487
Chimie agricole	0749
Chimie analytique	0486
Chimie minérale	0488
Chimie nucléaire	0738
Chimie organique	0490
Chimie pharmaceutique	0491
Physique	0494
Polymères	0495
Radiation	0754
Mathématiques	0405
Physique	
Généralités	0605
Acoustique	0986
Astronomie et astrophysique	0606
Électronique et électricité	0607
Fluides et plasma	0759
Météorologie	0608
Optique	0752
Particules (Physique nucléaire)	0798
Physique atomique	0748
Physique de l'état solide	0611
Physique moléculaire	0609
Physique nucléaire	0610
Radiation	0756
Statistiques	0463
Sciences Appliqués Et Technologie	
Informatique	0984
Ingénierie	
Généralités	0537
Agricole	0539
Automobile	0540

Biomédicale	0541
Chaleur et ther modynamique	0348
Conditionnement (Emballage)	0549
Génie aérospatial	0538
Génie chimique	0542
Génie civil	0543
Génie électronique et électrique	0544
Génie industriel	0546
Génie mécanique	0548
Génie nucléaire	0552
Ingénierie des systèmes	0790
Mécanique navale	0547
Métallurgie	0743
Science des matériaux	0794
Technique du pétrole	0765
Technique minière	0551
Techniques sanitaires et municipales	0554
Technologie hydraulique	0545
Mécanique appliquée	0346
Géotechnologie	0428
Matériaux plastiques (Technologie)	0795
Recherche opérationnelle	0796
Textiles et tissus (Technologie)	0794

PSYCHOLOGIE

Généralités	0621
Personnalité	0625
Psychobiologie	0349
Psychologie clinique	0622
Psychologie du comportement	0384
Psychologie du développement	0620
Psychologie expérimentale	0623
Psychologie industrielle	0624
Psychologie physiologique	0989
Psychologie sociale	0451
Psychométrie	0632



Memorandum

To: Whom it may concern
From: Allan K. P. Toh
Date: Friday, August 12, 1994
Subject: Font size in document (diagrams)

I realize that the labels and figure titles in some of my diagrams in this thesis will not turn out when microfilmed, nevertheless, I would appreciate it if you will go ahead and microfilm them anyway (since the axes labels are not important to the reader, but the shape of the diagram is). Thank you.

Yours sincerely,



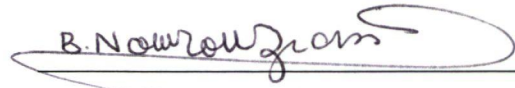
AT/AT

THE UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Artificial Neural Network Flux Estimation for Field Oriented Control", submitted by Allan K. P. Toh in partial fulfillment of the requirements for the degree of Master of Science.



Supervisor, Dr. E. P. Nowicki
Dept. of Electrical and Computer Engineering



Dr. B. Nowrouzian
Dept. of Electrical and Computer Engineering



Dr. J. Salmon
Dept. of Electrical Engineering,
University of Alberta

Date : AUG 4th 1994

Abstract

Field oriented control (FOC), sometimes referred to as vector control, is used in inverter-fed induction motor drives to obtain high performance speed response. For field oriented control it is necessary to know the instantaneous magnitude and position of the rotor flux. The magnitude and position of the rotor flux is approximated based on flux measurements in the direct FOC scheme and estimated in the indirect FOC scheme. In this thesis, a novel flux estimator, the *artificial neural network flux estimator*, is presented. The neural network is able to estimate accurately the rotor flux magnitude or position (maximum absolute error is less than 0.03 p.u.) for line-start operation of an induction motor as well as for field oriented control. A sensitivity study indicates that the neural network is quite insensitive to variations in the rotor resistance (maximum absolute error is about 0.10 p.u. if rotor resistance is increased to twice the nominal value). Its ability to estimate flux response that lies outside of the neural network training data set is another one of its strengths. This preliminary work indicates that the neural network flux estimator is a practical alternative to other flux estimation methods.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Dr. E.P. Nowicki, for his guidance, support, understanding and most of all his constant encouragement and confidence in me throughout my thesis program.

I would also like to thank Mr. Farhad Ashrafzadeh for his invaluable suggestions, constructive criticism and expert knowledge in the area of induction motors, which greatly accelerated my research work.

Many thanks are extended to the professors and supporting staff, especially the secretaries in the Department of Electrical and Computer Engineering, the University of Calgary, for all their help during my course of study here.

I wish to thank all my friends, fellow students and people whose names are impossible to be all listed here for all their support and insights.

I would also like to thank my wife, Stella, for her understanding, patience and enduring love without which I would not have finished this thesis program. Finally, I thank God for guiding me through this work.

Dedication

To my loving wife

Stella

TABLE OF CONTENTS

APPROVAL PAGE	ii
ABSTRACT	iii
ACKNOWLEDGMENTS	iv
DEDICATION	v
TABLE OF CONTENTS	vi
LIST OF FIGURES	ix
LIST OF SYMBOLS	xiii

Chapter 1

Introduction	1
1.1 Field oriented Control	2
1.2 Neural Networks	4
1.3 Thesis/Chapter Outline	7

Chapter 2

Field Oriented Control and Flux Estimation	9
2.1 Introduction	9
2.2 Field Oriented Control Theory	9
2.2.1 Principle of Variable Transformation	14
2.2.2 Synchronous rotating frame model	18
2.3 Direct and Indirect Field Oriented Control	23
2.3.1 Direct Field Oriented Control	23
2.3.2 Indirect Field Oriented Control	25
2.3.3 Generalized Field Oriented Control System	26
2.4 Flux estimation	29

Chapter 3

Artificial Neural Networks	34
-----------------------------------	----

3.1 Introduction to Artificial Neural Networks	34
3.2 History of Artificial Neural Networks	37
3.2.1 Rosenblatt's Perceptron	38
3.2.2 Minsky and Papert's Perceptron	39
3.2.3 Widrow's Adaptive Linear Element	40
3.2.4 Kohonen's Network	40
3.2.5 Back-propagation Networks	41
3.3 The Back-propagation Approach	42
3.4 The Generalized Delta Rule	47
3.4.1 Output Layer Weight Updates	50
3.4.2 Hidden Layer Weight Updates	53
3.5 Training Considerations	54
3.5.1 Network Size	54
3.5.2 Training Data	56
3.5.3 Complexity of Learning	56
3.5.4 Termination Criterion	57
Chapter 4	
Implementation of Field Oriented Control	59
4.1 Introduction	59
4.2 System Description	59
4.2.1 Rotor Flux Control	60
4.2.2 Current Controllers	63
4.2.3 PI Controller Tuning	63
Chapter 5	
Training the Neural Network	69
5.1 Introduction	69
5.2 Generating data	69
5.3 Normalization of Data	73
5.4 Selection of network architecture	73
5.5 Training the network	74
Chapter 6	
Neural Network Testing and Parameter Sensitivity	75
6.1 Neural Network Test Results	75
6.2 Parameter Sensitivity	79

Chapter 7

Discussion	86
7.1 Effects of Non-Historic Data Input	86
7.2 Alternative Inputs to the Neural Network	87
7.3 Multiple Networks versus Single Network	87

Chapter 8

Conclusion and Future Work	89
8.1 Conclusions	89
8.2 Future Work	90

Reference	91
------------------------	----

Appendix A

A.1 Induction Motor Model	95
A.2 Results of Load and Voltage Disturbances	96
A.3 Motor Parameters	102
A.4 Induction Motor Model Program Listing	103

Appendix B

B.1 Field Oriented Control of Induction Motor Program Listing	115
--	-----

List of Figures

Figure 1.1	: Schematic of a processing element (PE)	6
Figure 2.1(a)	: Separately excited dc motor	10
Figure 2.1(b)	: Field oriented control of induction motor	10
Figure 2.2	: Per-phase equivalent circuit and phasor diagram for the induction motor	12
Figure 2.3	: Axes transformations	15
Figure 2.4(a)	: D-Q equivalent circuits at synchronously rotating reference frame : q^e - axis circuit	19
Figure 2.4(b)	: D-Q equivalent circuits at synchronously rotating reference frame : d^e - axis circuit	19
Figure 2.5	: Direct field oriented control	24
Figure 2.6	: Induction machine model for indirect FOC	25
Figure 2.7	: General control scheme for indirect field oriented control of an induction motor	28
Figure 2.8	: Rotor speed feedback scheme	29
Figure 2.9	: Phasor diagram for indirect field oriented control	31
Figure 2.10	: Decoupling control	33
Figure 3.1	: Biological Nerve Cell	35
Figure 3.2	: Schematic of a processing element (PE)	36
Figure 3.3	: The perceptron	39
Figure 3.4	: A typical back-propagation neural network	44
Figure 3.5	: Three layer back-propagation network	48

Figure 4.1	: General control scheme for indirect field oriented control of an induction motor (same as Fig. 2.7)	61
Figure 4.2(a)	: FOC drive system employed to test the ANN flux estimator	62
Figure 4.2(b)	: PI rotor flux controllers	63
Figure 4.3	: Electromagnetic torque with field oriented control implemented	64
Figure 4.4	: Rotor speed with field oriented control implemented	64
Figure 4.5	: Rotor flux magnitude with field oriented control implemented	65
Figure 4.6	: $\sin(\phi)$ with field oriented control implemented	66
Figure 4.7	: $\cos(\phi)$ with field oriented control implemented	66
Figure 4.8	: I_{ds} in stationary reference frame with field oriented control implemented	67
Figure 4.9	: I_{qs} in stationary reference frame with field oriented control implemented	67
Figure 5.1(a)	: Training data set for $\sin(\phi)$	71
Figure 5.1(b)	: Training data set for $\sin(\phi)$ (continued)	72
Figure 5.2(a)	: Training data set for flux magnitude Ψ_r	72
Figure 5.2(b)	: Training data set for flux magnitude Ψ_r (continued)	73
Figure 6.1	: Neural network output for $\sin(\phi)$ vs test data for $TL = 0.25$ p.u.	76
Figure 6.2	: Neural network output for $\cos(\phi)$ vs test data for $TL = 0.25$ p.u.	77
Figure 6.3	: Neural network output for $\sin(\phi)$ vs test data for $TL = 0.25$ p.u. (expanded view)	77
Figure 6.4	: Neural network output for flux magnitude vs test data for $TL = 0.25$ p.u.	78
Figure 6.5	: Neural network output for $\sin(\phi)$ vs test data for $TL = 2.0$ p.u. (outside training range)	78

Figure 6.6	: Neural network output for flux magnitude vs test data for TL = 2.0 p.u. (outside training range)	79
Figure 6.7	: Neural network output for $\sin(\phi)$ vs test data for 50% increase in rotor resistance (PI controller adjusted)	80
Figure 6.8	: Neural network output for flux magnitude vs test data for 50% increase in rotor resistance (PI controller adjusted)	80
Figure 6.9	: Neural network output for $\sin(\phi)$ vs test data for 100% increase in rotor resistance (PI controller adjusted)	81
Figure 6.10	: Neural network output for flux magnitude vs test data for 100% increase in rotor resistance (PI controller adjusted)	81
Figure 6.11	: Neural network output for $\sin(\phi)$ vs test data for 20% increase in rotor resistance	82
Figure 6.12	: Neural network output for flux magnitude vs test data for 20% increase in rotor resistance	83
Figure 6.13	: Neural network output for $\sin(\phi)$ vs test data for 30% increase in rotor resistance	84
Figure 6.14	: Neural network output for flux magnitude vs test data for 30% increase in rotor resistance	84
Figure 7.1	: Neural network output for flux magnitude vs test data without historic data	86
Figure A.1	: Rotor speed for line start operation	96
Figure A.2	: Electromagnetic torque for line start operation	97
Figure A.3	: Flux magnitude for line start operation	97
Figure A.4	: I_{ds} vs time in the stationary reference frame	98
Figure A.5	: I_{qs} vs time in the stationary reference frame	98
Figure A.6	: I_{dr} vs time in the stationary reference frame	99
Figure A.7	: I_{qr} vs time in the stationary reference frame	99

Figure A.8	: Electromagnetic torque for various load and fault conditions (line start)	100
Figure A.9	: Stator current for various load and fault conditions (line start)	101
Figure A.10	: Rotor speed for various load and fault conditions (line start)	101
Figure A.11	: Rotor flux magnitude for various load and fault conditions (line start)	102

List of Symbols

ANN	Artificial neural network
E_p	Total error
FOC	Field oriented control
K'_t, K'_T	Proportionality constant
I_a	Armature current
I_f	Field current
I_s	Stator current (rms)
J	Moment of inertia
L_a	Armature inductance (dc motor)
L_m	Magnetizing inductance
L_r	Rotor inductance
L_s	Stator inductance
L_{lr}	Rotor leakage resistance
L_{ls}	Stator leakage resistance
P	Number of poles
PE	Processing Element
R_r	Rotor resistance
R_s	Stator resistance
T	Three phase to two axes (phase) transformation matrix
T_e	Electromagnetic Torque
T_L	Load Torque
T_R	Rotor time constant
U	Stationary to synchronously rot. ref. frame transformation matrix
U_i	Inputs to PE
V_f	Induced emf (rms)
V_t	Terminal Voltage
$V_{s^{ph}}$	Stator voltage from physical three phase
$V_{s^{s\ q}}$	Stator voltage in stationary d - q frame
$V_{s^{dq}}$	Stator voltage in synchronously rotating d - q frame
V_{sm}	Stator peak voltage
W_{ij}	Weight of PE
d	Direct axis component in field frame
q	Quadrature axis component in field frame
d^e	Direct axis component in synchronously rotating reference frame
q^e	Quadrature axis component in synchronously rotating reference frame
d^s	Direct axis component in stationary reference frame
q^s	Quadrature axis component in stationary reference frame
e_a	Armature voltage (dc motor)
f_j^h	Activation function in hidden layer j
f_k^o	Activation function in output layer k
i_{dr}	Instantaneous d -axis rotor current in field oriented reference frame

i_{ds}	Instantaneous d -axis stator current in field oriented reference frame
i_{qr}	Instantaneous q -axis rotor current in field oriented reference frame
i_{qs}	Instantaneous q -axis stator current in field oriented reference frame
i_{dr}^e	Instantaneous d -axis rotor current in synchronously rot. reference frame
i_{ds}^e	Instantaneous d -axis stator current in synchronously rot. reference frame
i_{qr}^e	Instantaneous q -axis rotor current in synchronously rot. reference frame
i_{qs}^e	Instantaneous q -axis stator current in synchronously rot. reference frame
i_{dr}^s	Instantaneous d -axis rotor current in stationary reference frame
i_{ds}^s	Instantaneous d -axis stator current in stationary reference frame
i_{qr}^s	Instantaneous q -axis rotor current in stationary reference frame
i_{qs}^s	Instantaneous q -axis stator current in stationary reference frame
i_{dr}^*	Command d -axis rotor current
i_{ds}^*	Command d -axis stator current
i_{qr}^*	Command q -axis rotor current
i_{qs}^*	Command q -axis stator current
i_{pj}^{th}	j^{th} output in hidden layer with p as the input vector
net_j^{th}	Net input to the j^{th} hidden layer
$net_{pk}^{o_{pj}}$	Net input to the k^{th} output layer
o_{pk}	k^{th} output in output layer with p as the input vector
r_a	Armature resistance (dc motor)
r_f	Field resistance (dc motor)
s	Laplace operator
v_{dr}	Instantaneous d -axis rotor voltage in field oriented reference frame
v_{ds}	Instantaneous d -axis stator voltage in field oriented reference frame
v_{qr}	Instantaneous q -axis rotor voltage in field oriented reference frame
v_{qs}	Instantaneous q -axis stator voltage in field oriented reference frame
v_{dr}^e	Instantaneous d -axis rotor voltage in synchronously rot. reference frame
v_{ds}^e	Instantaneous d -axis stator voltage in synchronously rot. reference frame
v_{qr}^e	Instantaneous q -axis rotor voltage in synchronously rot. reference frame
v_{qs}^e	Instantaneous q -axis stator voltage in synchronously rot. reference frame
v_{dr}^s	Instantaneous d -axis rotor voltage in stationary reference frame
v_{ds}^s	Instantaneous d -axis stator voltage in stationary reference frame
v_{qr}^s	Instantaneous q -axis rotor voltage in stationary reference frame
v_{qs}^s	Instantaneous q -axis stator voltage in stationary reference frame
v_{as}	Phase a stator voltage
v_{bs}	Phase b stator voltage
v_{cs}	Phase c stator voltage
w_{ji}^h	Weight between i^{th} input layer unit and j^{th} hidden layer unit
w_{kj}^o	Weight between j^{th} hidden layer unit and k^{th} output layer unit
x_{ki}	i^{th} component of the k^{th} training vector
y_{pk}	Desired output
α	Momentum coefficient
δ_{pk}	Error term of the k^{th} unit (neuron) with p as the input vector

δ_{pk}^o	Output layer (k) error term with p as the input vector
δ_{pj}^h	Hidden layer (j) error term with p as the input vector
ε_k	Difference between actual and correct output
μ	Positive constant
η	Learning coefficient
ϕ	Field angle
θ_e	Synchronously rotating frame angle ($\omega_e t$)
θ_r	Rotor angle ($\omega_r t$)
θ_{sl}	Slip angle ($\omega_{sl} t$)
θ_j^h	Bias weight in hidden layer
θ_k^o	Bias weight in output layer
ω_e	Stator (synchronously rotating frame) frequency
ω_r	Rotor electrical speed
ω_m	Rotor mechanical speed
ω_{sl}	Slip frequency
Ψ_m	Air gap flux (ac motor)
Ψ_g	Air gap flux (dc motor)
Ψ_{dr}	d -axis rotor flux linkage in field oriented reference frame
Ψ_{qr}	q -axis rotor flux linkage in field oriented reference frame
Ψ_{ds}	d -axis stator flux linkage in field oriented reference frame
Ψ_{qs}	q -axis stator flux linkage in field oriented reference frame
Ψ_{dm}	d -axis air gap flux linkage
Ψ_{qm}	q -axis air gap flux linkage
Ψ_r	Rotor flux
$\hat{\Psi}_r$	Rotor flux (estimated)
Ψ_r^*	Rotor flux (command)
Ψ_{dr}^e	d -axis rotor flux linkage in synchronously rot. reference frame
Ψ_{qr}^e	q -axis rotor flux linkage in synchronously rot. reference frame
Ψ_{dr}^s	d -axis rotor flux linkage in stationary reference frame
Ψ_{qr}^s	q -axis rotor flux linkage in stationary reference frame

Chapter 1

Introduction

In order to attain the high standards of performance obtainable by dc motor systems used in servo drive applications, field oriented control (FOC) was developed for induction motors in the early seventies. Field oriented control, is employed to keep the rotor flux constant during the normal operation of the induction motor (especially during speed transients). The squirrel cage induction motor fed from a pulse width modulation (PWM) inverter is becoming a popular choice for industrial applications in the KW to MW range. However, because of the nature of the motor's construction, the rotor windings are not accessible to extract the necessary flux information needed for FOC. This has led to two major techniques for obtaining the required flux information, namely the *direct* FOC and the *indirect* FOC. Historically, direct FOC implies a direct measurement of the air-gap flux, whereas in indirect FOC the flux quantities are estimated without the use of any magnetic field sensors.

The indirect FOC method of estimating rotor flux quantities is the focus in this thesis. There are several methods for indirect FOC, all of which require the knowledge of some combination of stator currents, stator voltages and rotor speed to determine the flux magnitude and angle. The drive system is usually modeled using differential equations which are solved in real-time to achieve indirect FOC.

In this thesis, a novel approach is adopted to calculate the rotor flux magnitude and angle. This approach requires only one set of input variables (i.e. stator currents), and

does *not* require any mathematical modeling (in the execution phase). It is based on the use of the *artificial neural network* (ANN). The ANN “learns by example” and exhibits high accuracy, good performance and robustness. An introduction of field oriented control and artificial neural networks is presented in the latter part of this chapter.

1.1 Field oriented Control

Given an induction motor with a balanced three-phase supply, the two axis or d - q theory [1] is normally used for dynamic modeling. In this theory, time-varying parameters are eliminated and the variables and parameters are expressed in orthogonal or mutually decoupled *direct* (d) and *quadrature* (q) axes. It is convenient to represent the axes in either the stationary or one of several rotating reference frames. In the stationary reference frame the d - q axis is fixed on the stator and is denoted by d^s and q^s , respectively. On the other hand, the d - q axes can be rotating at synchronous speed (referenced for example to the stator or rotor flux) or fixed on the rotor. For the synchronous rotating reference frame fixed on the stator magnetic field, the d - q axes are commonly denoted by d^e and q^e . For the field oriented frame (i.e. the synchronously rotating reference frame fixed on the rotor flux) no superscripts are used. The d - q dynamic model of a machine can be expressed in either a stationary or a rotating reference frame. In a stationary reference frame, the reference d^s and q^s axes are fixed on the stator. The advantage of a synchronously rotating frame model is that sinusoidal variables are transformed into d.c. quantities for steady-state conditions. In induction motor drive systems, the dynamic performance is a complex one because of the coupling effect between the stator and rotor phases, where

coupling coefficients vary with rotor position. In particular, in the operation of a variable speed induction motor drive system where field oriented control is not used, when an incremental torque is demanded, the values of the rotor flux linkage components on both d and q axes are changed to a new level with a slow transient (e.g. on the order of one second for a 30 HP motor). Meanwhile, the electromagnetic torque produced has a damped oscillation with a transient time depending on the rotor electrical time constant (e.g. about 0.4 s for 30 HP motor) [2].

In 1972, F. Blaschke presented a new technique that decouples motor flux and torque [3]. K. Hasse [4] together with Blaschke established the Field Oriented Control theory in which a straightforward linear control law can be established between motor primary current and torque. In order to accomplish this, they fixed the d -axis in the rotor synchronous reference frame, which indicates the orientation of the flux linkage to coincide with the total rotor flux linkage of the machine [5,6]. This is done such that all magnetization of the motor is along the d -axis. The torque components of the stator currents are along the q -axis which is orthogonal to the motor flux. The torque can be controlled in proportion to the stator current component along the q -axis while maintaining the magnetization constant.

In this way, in field oriented control, sometimes called vector control, an induction motor is controlled like a separately excited d.c. motor. Under field oriented control, one can obtain a fast torque response, just as in the case of a d.c. drive system. The final result is that the a.c. drive system has the performance of a d.c. drive system, but it employs the simple and rugged squirrel cage induction. Inherent in the field oriented control is the

problem of flux estimation in the rotor. Originally, rotor flux was calculated using Hall effect sensors in the machine. The problem is that the Hall effect sensor output drifts with temperature and it is difficult to compensate. Another method of measurement is to mount flux coils in the air gap and the corresponding induced voltage may be integrated to obtain the flux information [1]. Both methods require mounting external devices in the induction machine and this is not favored by designers.

1.2 Neural Networks

The neural network has a history that spans some five decades [7], but not until the late nineteen eighties did neural networks emerge as a practical computational technology. Today neural networks can be applied to solve problems that are difficult for conventional computers or human beings. Neural networks are being adopted for use in a wide variety of commercial, industrial and military applications. They range from applications such as pattern recognition, identification, classification, speech and vision to complex real-time adaptive control systems, and from small scale associative searching to large scale system optimization and scheduling. In most of these applications, the neural network, with its self organizing ability and its inherent nature of being an adaptive process, can be developed within a reasonable time-frame. Often such neural networks can out perform the conventional technologies (this even includes expert systems). When embedded in a hardware implementation, a neural network exhibits a high fault tolerance to system damage. More important, it also offers high overall data throughput rates due to its parallel processing capabilities. With the many different hardware options available,

including VLSI realization, the introduction of neural networks into existing and recently developed systems can be obtained at a reasonably low cost. Neural networks come in many different types, each of which has different characteristics and abilities related to their learning methods, dynamics and structure. In this thesis, we shall explore the use of neural networks in estimating the rotor flux of an induction machine.

The Neural network (or neural net), has its origins inspired by our biological neural systems and it is an implementation of an algorithm inspired by research into the brain. The neural network technology encompasses a wide variety of applications. It learns from a given set of data, thereby performing classifications, function estimation and complex control sequences. Among these neural nets, the more common and the best known ones are the MLPs (Multi-Layer Perceptron networks) [8][9], the Hopfield networks [10], the Kohonen self-organizing maps [11] and the Back-propagation networks [7][9]. In general, all neural nets share the following advantages when compared to the conventional technologies :

- Capable of real-time non-linear operations
- Inherently adaptive in learning
- Self-organizing
- Capable of generalization
- Implementable easily in existing technologies, e.g. VLSI
- Highly fault tolerant

The artificial neural network can be viewed as a dense interconnection of many non-linear computational elements called neurons, sometimes known as processing elements (PEs). This network of neurons is then capable of high speed non-linear computation due to its parallel structure. Non-linearity is built into each individual neuron (or PE) which sums N weighted inputs and passes the result through a non-linearity,

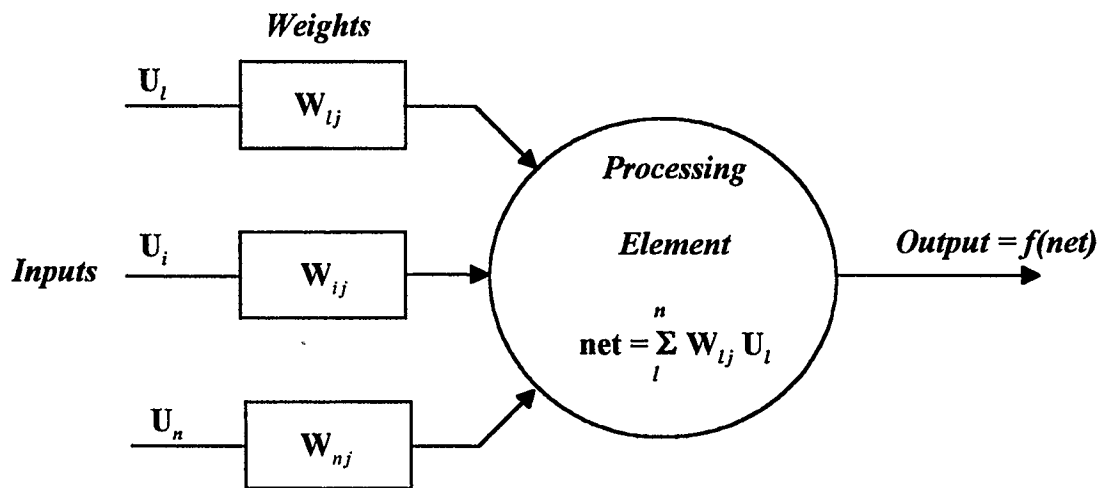


Figure 1.1 Schematic of a Processing Element (PE)

known as the activation function, to give an output. The general schematic of a PE is shown in Fig. 1.1. By Organizing the processing elements into different layers and by connecting them with proper weights, networks can be created that are capable of performing tasks that are highly non-linear. The three common types of activation function are *hard limiter*, *threshold logic element* and *sigmoidal non-linearity*. In order for the network to “learn” to perform a task, the network is required to go through a “training” process using some form of training algorithm. The input weights of each neuron are

typically adapted during the training operation to improve performance. For the network to “learn” we mean that the weights in each processing element are adapted so that the overall network is able to generate the desired output when given a valid input. Hence, neural networks are a self-learning means of emulating the input/output relationships of very non-linear systems [12]. From what has been said so far, we see that a neural net model is specified by its net topology, its neuron characteristics and its training rules.

1.3 Thesis/Chapter Outline

This thesis is composed of eight chapters. They are organized in the following manner :

Chapter 2 is dedicated to the review of the $d-q$ (two axes) theory, matrix transformations and field oriented control principles .

Chapter 3 serves as a brief review of the basic concepts of artificial neural networks. A brief history on different ANNs is given as well as a detail mathematical derivation of the generalized delta rule used to update weights.

Chapter 4 discusses the implementation of a field oriented control drive system, using models obtained in Chapter 2. The FOC is implemented with the help of very simple PI controllers.

Chapter 5 introduces the training/learning procedure of the ANN flux estimator. The multi-layer artificial neural network using error back-propagation is chosen for this thesis. A discussion on the method used for training and the selection of training parameters is also presented.

Chapter 6 presents the results of ANN flux estimator tests, including test data that is outside the training data set. Some parameter sensitivity tests are also performed here, e.g. load torque tests and rotor resistance variation tests.

Some discussions are presented in Chapter 7, followed by conclusions and suggestions for future work in Chapter 8.

Chapter 2

Field Oriented Control and Flux Estimation

2.1 Introduction

To construct a high performance induction motor drive system, it is desirable to establish a relationship between the motor electromagnetic torque, i.e. the controlled variable, and the control variables such as motor voltages or currents. This permits the motor flux to be controlled at the optimal constant level to achieve a fast speed response. This is the underlying objective of field oriented control. A review of the two-axes (d - q) theory and field oriented control principles is given in this chapter.

2.2 Field Oriented Control Theory

In field oriented control, an a.c motor is controlled like a separately excited d.c. motor [1]. Using Fig. 2.1, the analogy can be made clearer whereby Fig. 2.1(a) depicts the separately excited d.c. motor equivalent circuit, and Fig. 2.1(b) illustrates field oriented control of an induction motor. In a d.c. machine, neglecting the armature demagnetization effect and field saturation, the electromagnetic torque is given by

$$T_e = K_t \Psi_g J_a = K'_t J_a J_f \quad (2.1)$$

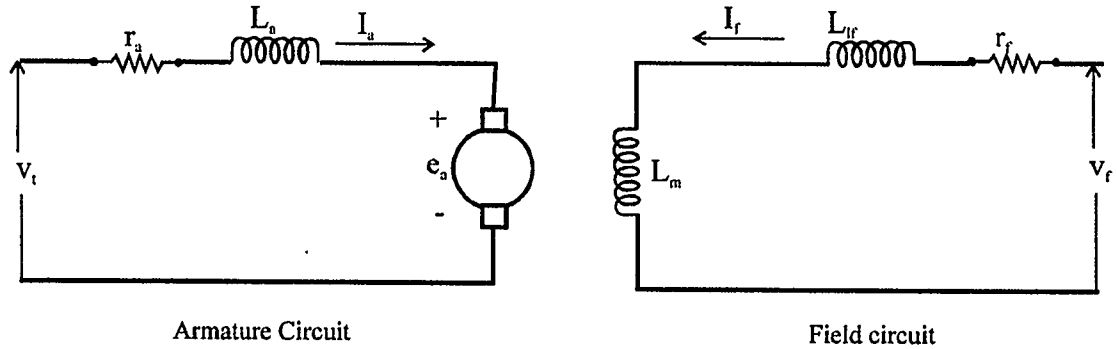
where : K'_t, K_t = Proportionality constants

J_a = Armature or torque component of current

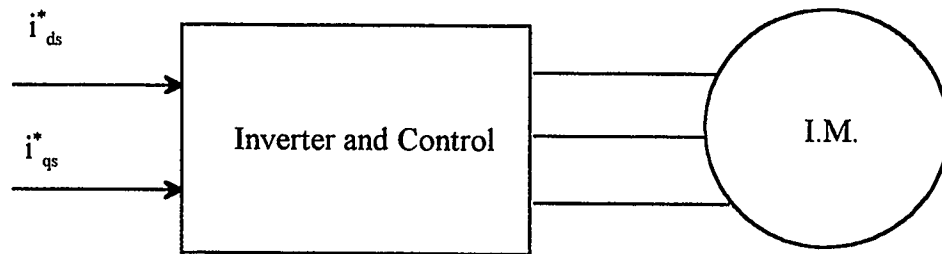
J_f = Field or flux component of current

Ψ_g = Air gap flux

In a d.c. machine, the control variables I_a and I_f can be considered as orthogonal or *decoupled* vectors. In normal operation, the field current I_f is set to maintain the rated field flux and torque is changed by changing the armature current. Since the current I_f or



(a) Separately excited d.c. motor



(b) Field oriented control induction motor

Figure 2.1 Induction motor and d.c. machine analogy

the corresponding field flux is decoupled from the armature current I_a , the torque sensitivity remains maximum in both transient and steady-state operations. This mode of control can be extended to an induction motor, if the machine operation is considered in a synchronously rotating reference frame where the sinusoidal variables appear to be d.c. quantities. In Fig. 2.1(b), the command variables to the inverter and control block are i_{ds}^*

and i_{qs}^* . The currents i_{ds} and i_{qs} are, respectively, direct-axis component and quadrature-axis component of the stator current, where both of these quantities are in the rotor synchronous rotating reference frame. In field oriented control, i_{ds} is analogous to the field current I_f and i_{qs} is analogous to the armature current I_a of a d.c. machine. Torque then can be expressed as previously given as Eqn. (2.1),

for a d.c. machine :

$$T_e = K_t \Psi_g I_a = K'_t I_d I_f \quad (2.2)$$

and for a induction motor in the rotor synchronous rotating frame :

$$T_e = K_T \hat{\Psi}_m i_{qs} = K'_T i_{qs} i_{ds} \quad (2.3)$$

where :

$K'_t, K_t, K'_T, K_T =$ proportionality constant

$\Psi_g, \hat{\Psi}_m =$ Air gap flux

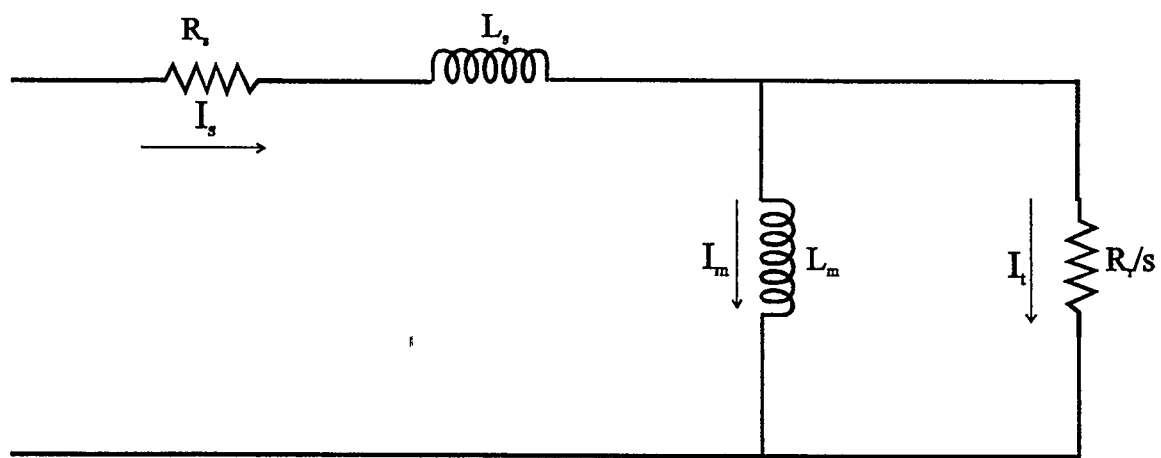
The principle of field oriented control can be illustrated by means of a phasor diagram. The equivalent circuit and phasor diagram are shown in Fig. 2.2, where I_m is the magnetizing current and I_t is the torque current corresponding to the given phasor I_s , and I_s represents the per-phase stator current of an induction machine. Hence in the complex plane,

$$I_m = I_s \cos \theta \quad (2.4)$$

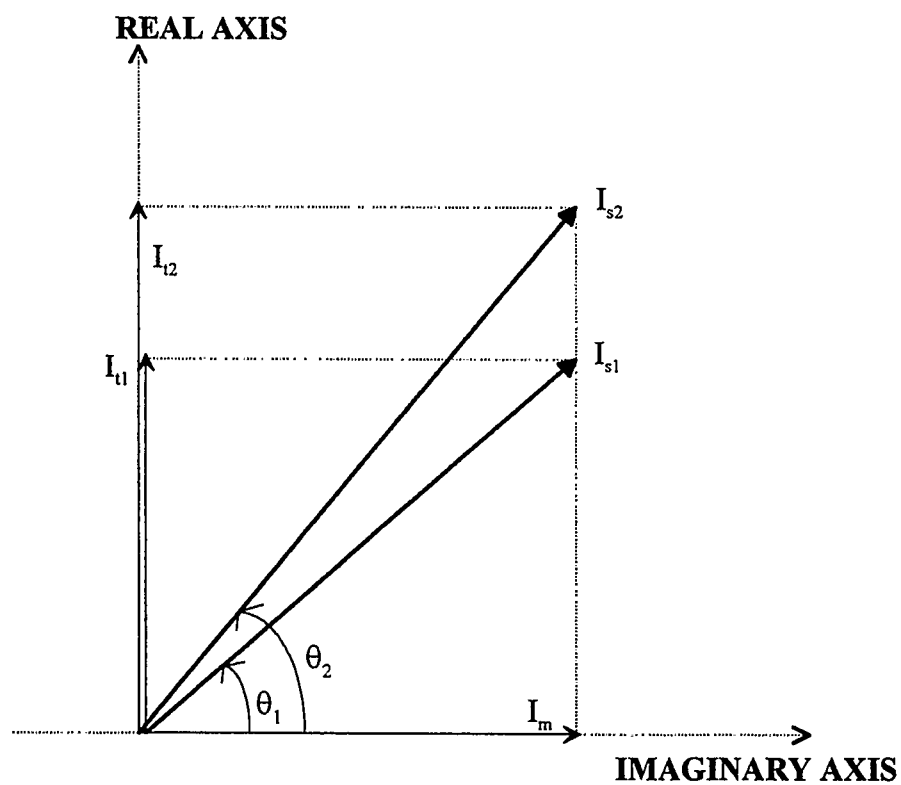
$$I_t = I_s \sin \theta \quad (2.5)$$

The rotor leakage inductance of the machine is assumed to be negligible [13].

In field oriented control theory, a control law can be established such that I_m is maintained constant, and thereby making the rotor flux constant. As for I_t , it is controlled



(a)



(b)

Figure 2.2 Per-phase equivalent circuit and phasor diagram for the induction motor

in proportion to the torque required. Under field oriented control, the stator current phasor in Fig. 2.2(b) becomes I_{s1} for a given small torque and I_{s2} when the torque is larger. The amplitude, I_s , and the phase angle, θ , of the stator current phasor is given by the expression:

$$I_s = \sqrt{I_m^2 + I_t^2} \quad (2.6)$$

$$\theta = \tan^{-1}\left(\frac{I_t}{I_m}\right) \quad (2.7)$$

From the equivalent circuit:

$$L_m I_m \omega_e = \frac{R_r}{s} I_t \quad (2.8)$$

where : $\omega_e = 2\pi f_e$; f_e is the stator excitation frequency.

The angular slip frequency is then given by:

$$\omega_{sl} = \frac{R_r I_t}{L_m I_m} \quad (2.9)$$

There are three key variables in field oriented control of induction motors, namely, the stator excitation frequency (synchronous frequency), the amplitude, and the phase angle of the stator current phasor. These quantities should be controlled so as to satisfy Eqns. (2.6), (2.7) and (2.9). Note that a drive system employing field oriented control will not necessarily solve Eqns. (2.6), (2.7) and (2.9) explicitly, but it must do so implicitly.

In a drive system where the speed is adjustable, the machine normally constitutes an element within a feedback loop [1], hence we need to take into consideration its dynamic behavior. The dynamic performance of an a.c. machine is complex because of the coupling effect between the stator and rotor phases, where the coupling coefficients vary

with rotor position. The machine can be described by differential equations with time-varying coefficients.

2.2.1 Principle of Variable Transformation

Given an induction motor with a balanced three-phase supply, the d - q axis or two axis theory is normally used for dynamic modeling. In this theory, time varying parameters are eliminated and the variables and parameters are expressed in orthogonal or mutually decoupled *direct* (d) and *quadrature* (q) axis components in the synchronous rotating frame reference. It is convenient to represent the axes in either the stationary or one of several rotating reference frames. In the stationary reference frame the d - q axis is fixed on the stator and is denoted by d^s and q^s , respectively. On the other hand, the d - q axes can be rotating at synchronous speed (referenced for example to the stator or rotor flux) or fixed on the rotor. For the synchronous rotating reference frame fixed on the stator magnetic field, the d - q axes are commonly denoted by d^e and q^e . For the field oriented frame (i.e. the synchronously rotating reference frame fixed on the rotor flux) no superscripts are used.

The equations developed in this section and the following section are based on [1]. The principle of variable transformation is shown in Fig. 2.3, where there are three physical phases as , bs , cs , (fixed relative to the stator datum), the stationary reference frame axes d^s and q^s and the rotating reference frame axes d^e and q^e . The angle x is arbitrary between the phase as -axis and the stator datum i.e. d^s -axis. The mathematical transformation for stator voltages, designated as $V_{s_{ph}}$ (or currents) from the physical three phase and d^s - q^s two phase frame fixed on the stator is given by :

$$\mathbf{Vs}_{dq}^s = T \mathbf{Vs}_{ph} \quad (2.10)$$

where :

$$\mathbf{Vs}_{dq}^s = [v_{qs}^s, v_{ds}^s]^T \quad (2.11)$$

$$\mathbf{Vs}_{ph} = [v_{as}, v_{bs}, v_{cs}]^T \quad (2.12)$$

$$T = \frac{2}{3} \begin{bmatrix} \cos(x) & \cos(x - \frac{2\pi}{3}) & \cos(x - \frac{4\pi}{3}) \\ \sin(x) & \sin(x - \frac{2\pi}{3}) & \sin(x - \frac{4\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad (2.13)$$

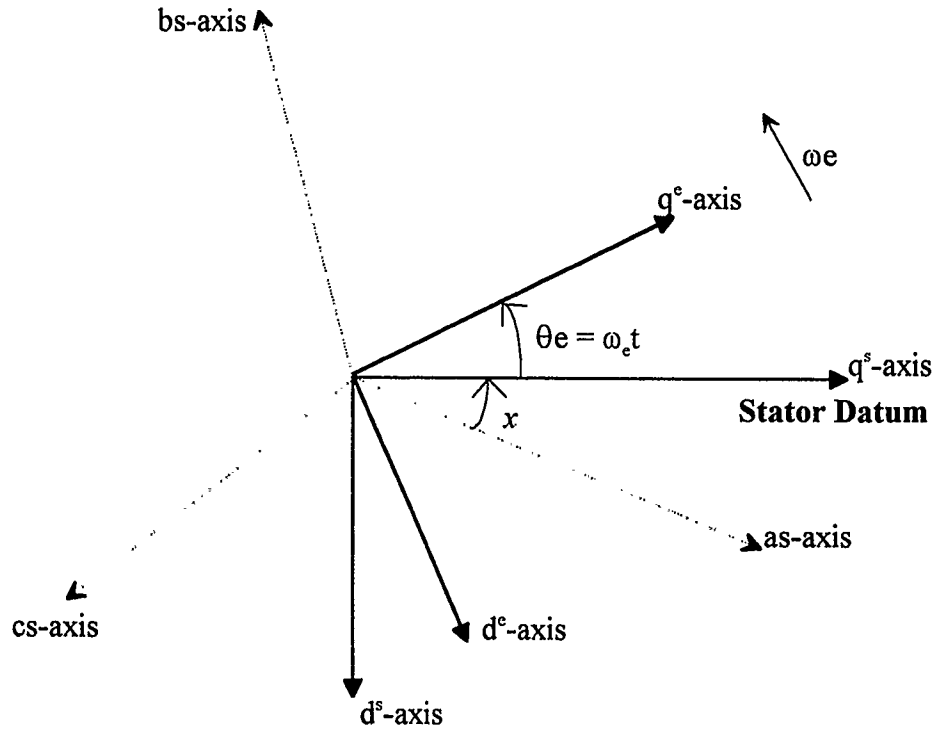


Figure 2.3 Axes transformations

conversely

$$\mathbf{V}_{s_{ph}} = \mathbf{T}^{-1} \mathbf{V}_{s_{dq}^{ss}} \quad (2.14)$$

where

$$\mathbf{T}^{-1} = \begin{bmatrix} \cos(x) & \sin(x) & 1 \\ \cos(x - \frac{2\pi}{3}) & \sin(x - \frac{2\pi}{3}) & 1 \\ \cos(x - \frac{4\pi}{3}) & \sin(x - \frac{4\pi}{3}) & 1 \end{bmatrix} \quad (2.15)$$

It is convenient to set $x = 0$, so that the d^s -axis coincides with the as -axis. With $x = 0$ we have :

$$\mathbf{T} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad (2.16)$$

and

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \end{bmatrix} \quad (2.17)$$

The transformation between the d^s - q^s stationary reference frame and the d - q rotating reference frame is given by :

$$\mathbf{V}_{s_{dq}} = \mathbf{U} \mathbf{V}_{s_{dq}^{ss}} \quad (2.18)$$

where :

$$\mathbf{V}_{s_{dq}} = [\mathbf{v}_{qs} \ \mathbf{v}_{ds}]^T \quad (2.19)$$

$$\mathbf{V}_{s_{dq}^{ss}} = [\mathbf{v}_{qs}^s \ \mathbf{v}_{ds}^s]^T \quad (2.20)$$

$$U = \begin{bmatrix} \cos \theta_e & -\sin \theta_e \\ \sin \theta_e & \cos \theta_e \end{bmatrix} \quad (2.21)$$

conversely

$$V_{s_{dq}}^{s\ s} = U^{-1} V_{s_{dq}} \quad (2.22)$$

where :

$$U^{-1} = \begin{bmatrix} \cos \theta_e & \sin \theta_e \\ -\sin \theta_e & \cos \theta_e \end{bmatrix} \quad (2.23)$$

and θ_e is the angle of the stator synchronously rotating frame (i.e. $\theta_e = \omega_e t$).

A similar type of operation can be done for the stator currents. Now, assume for example, that the phase voltages are balanced and sinusoidal. Then by simplifying Eqns. (2.10) and (2.14) in the synchronously rotating reference frame we have (zero-sequence components in T & T⁻¹ have been neglected) :

$$v_{as} = v_{qs}^s \quad (2.24)$$

$$v_{bs} = -\frac{1}{2}v_{qs}^s - \frac{\sqrt{3}}{2}v_{ds}^s \quad (2.25)$$

$$v_{cs} = \frac{1}{2}v_{qs}^s + \frac{\sqrt{3}}{2}v_{ds}^s \quad (2.26)$$

and

$$v_{qs}^s = \frac{2}{3}v_{as} - \frac{1}{3}v_{bs} - \frac{1}{3}v_{cs} = v_{as} \quad (2.27)$$

$$v_{ds}^s = -\frac{1}{\sqrt{3}}v_{bs} + \frac{1}{\sqrt{3}}v_{cs} \quad (2.28)$$

For a balanced, fixed voltage three-phase supply let

$$v_{as} = V_{sm} \cos(\omega_e t) \quad (2.29)$$

$$v_{bs} = V_{sm} \cos(\omega_e t - \frac{2\pi}{3}) \quad (2.30)$$

$$v_{cs} = V_{sm} \cos(\omega_e t + \frac{2\pi}{3}) \quad (2.31)$$

where : V_{sm} is the stator peak voltage

Substituting Eqn. (2.29), (2.30) and (2.31) into (2.27) and (2.28) we obtain:

$$v_{qs}^s = V_{sm} \cos(\omega_e t) \quad (2.32)$$

$$v_{ds}^s = -V_{sm} \sin(\omega_e t) \quad (2.33)$$

Again, substituting Eqn. (2.32) and (2.33) into (2.18) we have :

$$v_{qs}^e = V_{sm} \quad (2.34)$$

$$v_{ds}^e = 0 \quad (2.35)$$

Note that the sinusoidal variables appear as dc quantities in the stator synchronously rotating reference frame for steady-state operation (c.f. Appendix A). This is the primary advantage of using the d - q axis theory.

2.2.2 Synchronous rotating frame model

From the d - q equivalent circuit in Fig. 2.4, the stator voltages in the stator synchronously rotating reference frame are given by :

$$v_{qs}^e = R_s i_{qs}^e + \frac{d\Psi_{qs}^e}{dt} + \omega_e \Psi_{ds}^e \quad (2.36)$$

$$v_{ds}^e = R_s i_{ds}^e + \frac{d\Psi_{ds}^e}{dt} - \omega_e \Psi_{qs}^e \quad (2.37)$$

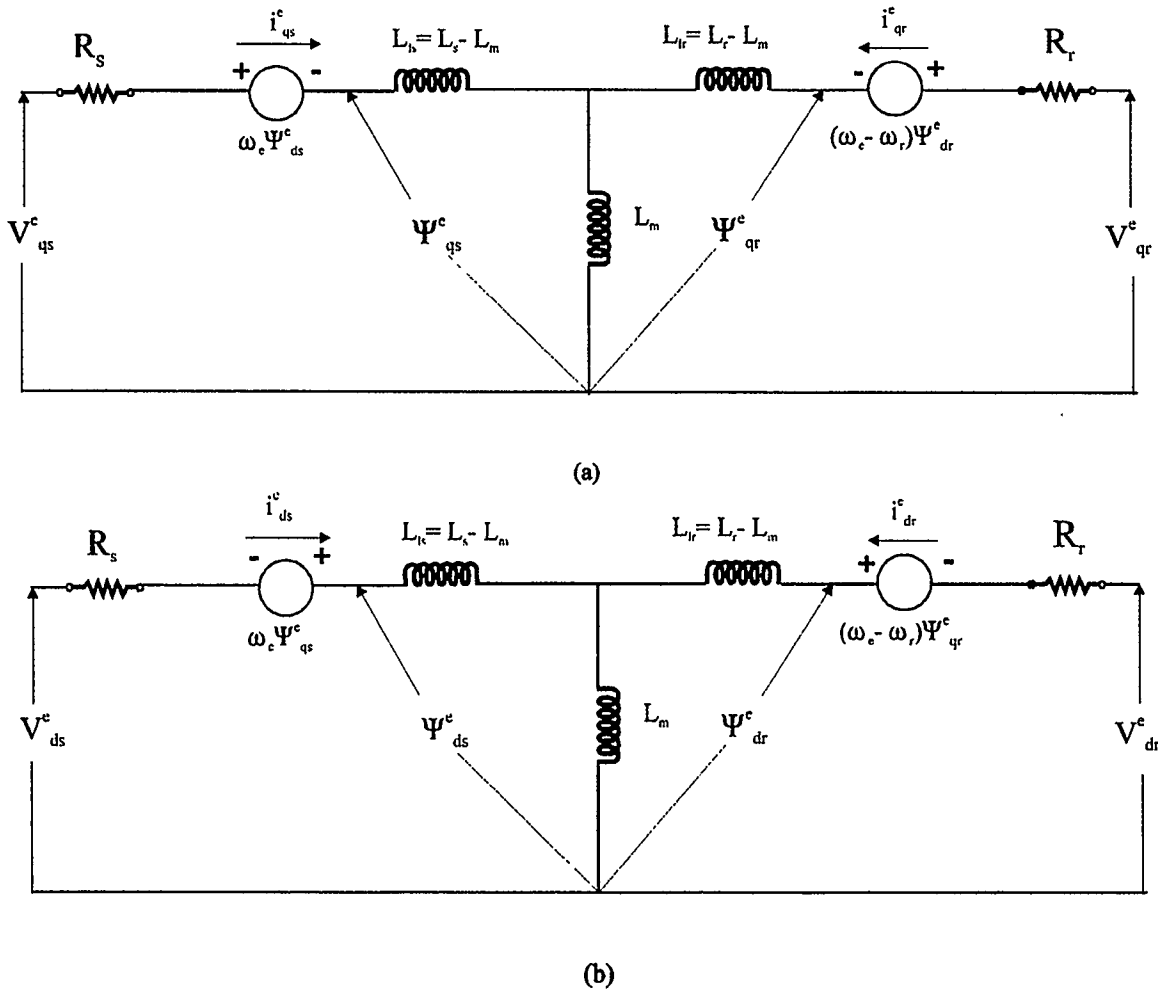


Figure 2.4 $D - Q$ equivalent circuits at synchronously rotating reference frame :

(a) q^e - axis circuit; (b) d^e - axis circuit

With the substitution of $\omega_e = 0$ in the above two equations, the stator equations in the stationary $d^s - q^s$ frame results. For a stationary rotor, the rotor voltage equations for a doubly fed machine will be similar to Eqns. (2.36) and (2.37) :

$$v_{qr}^e = R_r i_{qr}^e + \frac{d\Psi_{qr}^e}{dt} + \omega_e \Psi_{dr}^e \quad (2.38)$$

$$v_{dr}^e = R_r i_{dr}^e + \frac{d\Psi_{dr}^e}{dt} - \omega_e \Psi_{qr}^e \quad (2.39)$$

With the rotor moving at a speed of ω_r , the $d - q$ axes fixed on the rotor move at a speed of $\omega_e - \omega_r$ relative to the synchronous rotating reference frame. Hence for the synchronous rotating reference frame Eqns. (2.38) and (2.39) can be re-written as :

$$v_{qr}^e = R_r i_{qr}^e + \frac{d\Psi_{qr}^e}{dt} + (\omega_e - \omega_r) \Psi_{dr}^e \quad (2.40)$$

$$v_{dr}^e = R_r i_{dr}^e + \frac{d\Psi_{dr}^e}{dt} - (\omega_e - \omega_r) \Psi_{qr}^e \quad (2.41)$$

The following equations are the flux linkage expressions derived from the equivalent circuit.

$$\Psi_{qs}^e = L_{ls} i_{qs}^e + L_m (i_{qs}^e + i_{qr}^e) \quad (2.42)$$

$$\Psi_{qr}^e = L_{lr} i_{qr}^e + L_m (i_{qs}^e + i_{qr}^e) \quad (2.43)$$

$$\Psi_{ds}^e = L_{ls} i_{ds}^e + L_m (i_{ds}^e + i_{dr}^e) \quad (2.44)$$

$$\Psi_{dr}^e = L_{lr} i_{dr}^e + L_m (i_{ds}^e + i_{dr}^e) \quad (2.45)$$

where :

Ψ_{dr}^e = d^e -Axis rotor flux linkage

Ψ_{ds}^e = d^e -Axis stator flux linkage

Ψ_{qr}^e = q^e -Axis rotor flux linkage

Ψ_{qs}^e = q^e -Axis stator flux linkage

L_{ls} = Stator leakage inductance

L_{lr} = Rotor leakage inductance

L_m = Magnetizing inductance

i_{dr}^e = Instantaneous d^e -axis rotor current

i_{ds}^e = Instantaneous d^e -axis stator current

i_{qr}^e = Instantaneous q^e -axis rotor current

i_{qs}^e = Instantaneous q^e -axis stator current

Using Eqns. (2.36), (2.37), (2.40) and (2.41), the model of electrical dynamics in terms of voltages and currents in matrix form is as follows :

$$\begin{bmatrix} v_{qs}^e \\ v_{ds}^e \\ v_{qr}^e \\ v_{dr}^e \end{bmatrix} = \begin{bmatrix} R_s + sL_s & \omega_e L_s & sL_m & \omega_e L_m \\ -\omega_e L_s & R_s + sL_s & -\omega_e L_m & sL_m \\ sL_m & (\omega_e - \omega_r)L_m & R_r + sL_r & (\omega_e - \omega_r)L_r \\ -(\omega_e - \omega_r)L_m & sL_m & -(\omega_e - \omega_r)L_r & R_r + sL_r \end{bmatrix} \begin{bmatrix} i_{qs}^e \\ i_{ds}^e \\ i_{qr}^e \\ i_{dr}^e \end{bmatrix} \quad (2.46)$$

where :

s = the Laplace operator

v_{ds}^e = Instantaneous d^e -axis stator voltage

v_{dr}^e = Instantaneous d^e -axis rotor voltage

v_{qs}^e = Instantaneous q^e -axis stator voltage

v_{qr}^e = Instantaneous q^e -axis rotor voltage

R_r = Rotor resistance

R_s = Stator resistance

ω_e = Stator frequency (rad/s)

ω_r = Rotor electrical speed (rad/s)

L_s = Stator inductance

L_r = Rotor inductance

For a squirrel cage motor or single fed machine, the voltages v_{qr}^e and v_{dr}^e should be zero.

For a steady-state solution of the above equation, all s -related terms should be zero.

During steady-state, all variables in the synchronously rotating reference frame appear as d.c. quantities with sinusoidal excitation.

Electrical rotor speed can be related to torque as follows :

$$T_e - T_L = J \frac{d\omega_m}{dt} = \frac{2}{P} J \frac{d\omega_r}{dt} \quad (2.47)$$

where :

$$T_L = \text{Load torque}$$

$$J = \text{System inertia}$$

Developed torque in terms of d^e - q^e components is given by :

$$T_e = \frac{3}{2} \left(\frac{P}{2} \right) (\Psi_{dm}^e i_{qr}^e - \Psi_{qm}^e i_{dr}^e) \quad (2.48)$$

In terms of fluxes and currents using relations found in Eqn. (2.10) to (2.13) :

$$T_e = \frac{3}{2} \left(\frac{P}{2} \right) (\Psi_{dm}^e i_{qs}^e - \Psi_{qm}^e i_{ds}^e) \quad (2.49)$$

$$= \frac{3}{2} \left(\frac{P}{2} \right) (\Psi_{ds}^e i_{qs}^e - \Psi_{qs}^e i_{ds}^e) \quad (2.50)$$

$$= \frac{3}{2} \left(\frac{P}{2} \right) L_m (i_{qs}^e i_{dr}^e - i_{ds}^e i_{qr}^e) \quad (2.51)$$

Combining Eqns. (2.46), (2.47) and (2.51) we have the complete model of the electromechanical dynamics of an induction machine.

It should be mentioned that the dc model obtained is based on the assumptions noted above. Non-sinusoidal and /or unbalanced excitation wave forms will not yield a dc model! However, in the case of “square-wave” Current Source Inverter-Induction Motor (CSI-IM) drives, the currents of the inverter are balanced with a modified rectangular waveform. This kind of non-sinusoidal waveform can be analyzed into its Fourier components and a reference speed can be chosen for each component. Regarding the issue of harmonics, these harmonics are only weakly coupled to the motor shaft via

electromagnetic interactions. Hence, the study of the machine's electromechanical properties is only slightly affected by neglecting these harmonics. Moreover, modern switch-mode induction motor drives, employing bipolar junction transistor (BJT), or insulated gate bipolar transistor (IGBT), or gate turn-off (GTO) devices, apply nearly sinusoidal three-phase currents to the induction motor. Therefore this model is quite valid for most modern drive application.

2.3 Direct and Indirect Field Oriented Control

As noted in Chapter 1, there are two methods to obtain the magnitude and position of the rotor flux, namely :

- 1) the direct method, where the rotor flux is determined, based on *direct* measurement of air-gap or stator flux; and
- 2) the indirect method, where the rotor flux is *calculated* using motor flux models.

2.3.1 Direct Field Oriented Control

Direct method flux sensing consists of two sensing devices that are placed orthogonal to each other as shown in Fig. 2.5. Two methods are discussed here namely, the Hall sensing and the sensing coils methods.

In the Hall sensing method, the Hall effect sensors are placed in the air gap of the stator of the induction motor, and signals representing the local flux density are produced by integrating the measurements of substantial number of suitably placed transducers. One disadvantage of this method is that a complex adjustable filter is needed since strong slot

harmonics with speed dependent frequency are superimposed on the fundamental signal [14]. Another disadvantage with Hall effect sensors is due to its semiconductor nature, i.e. they are subject to severe thermal and mechanical stress and requires a modified motor. For the above reasons the Hall sensing technique is not an economical solution for many general applications.

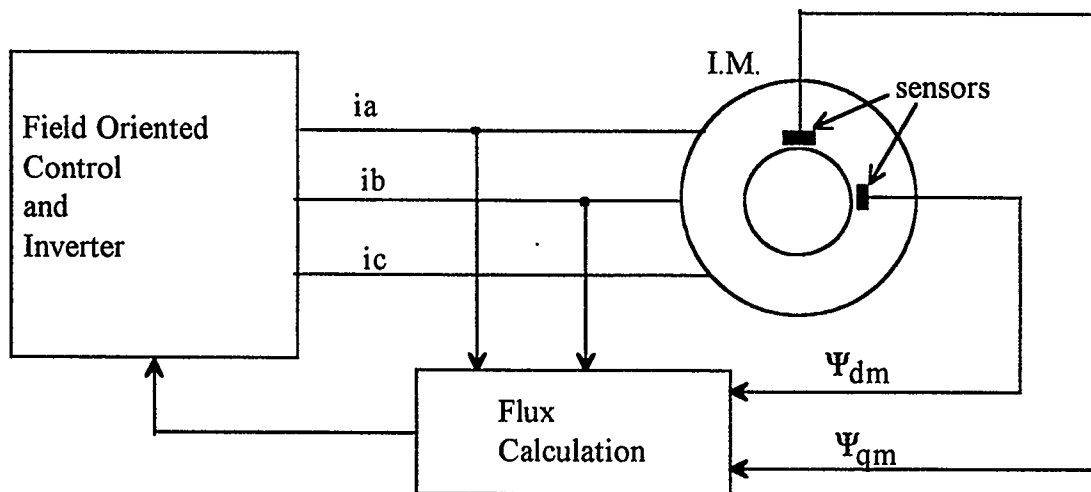


Figure 2.5 Direct field oriented control

In the sensing coils method, sensing coils are physically installed in the induction machine's stator, the voltages across the sensing coils are proportional to the flux change which is then integrated to represent the main flux of the motor. The sensing coils have certain qualities that eradicate the problems faced by the Hall sensors, that being, the sensing coils have avoided using active semiconductor components within the motor and at the same time since the sensing coils behave like low pass filters, undesirable slot

harmonics are filtered out. One major disadvantage of this technique is that the flux cannot be sensed at zero speed, hence making it unsuitable for position control.

Both the above methods require special modifications to be made to the motor therefore making standard production induction motors expensive to retrofit.

2.3.2 Indirect Field Oriented Control

To circumvent the limitations imposed by the direct method, the indirect method is used. To acquire the rotor flux of a standard induction motor, the equations of the motor flux model are solved in real time using measurable stator currents and rotor speed as driving functions. Fig. 2.6 shows a block diagram on how this is to be done.

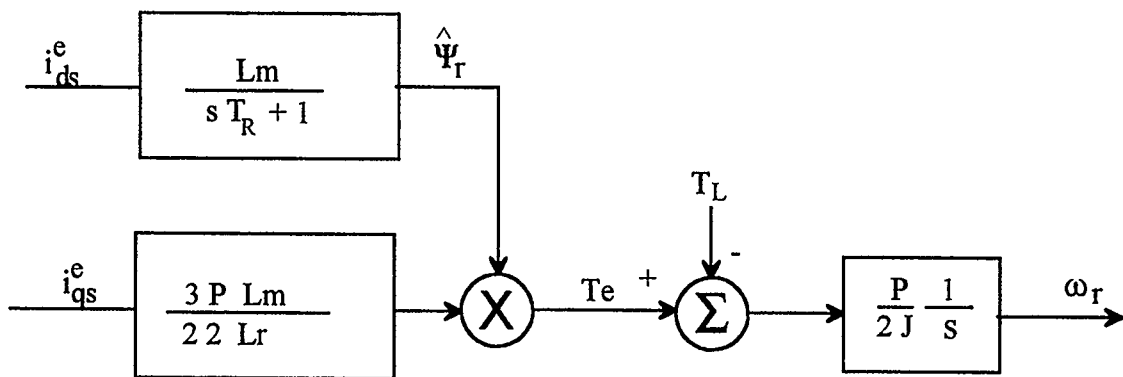


Figure 2.6 Induction machine model for indirect FOC

The advantage of this technique is that it does not require any form of flux sensing device to be placed inside the induction motor. Moreover, flux can be obtained even down to zero frequency making it attractive for position control. However, the major setback of

indirect field oriented control is the variations of rotor time constant T_R where $T_R = L_r/R_r$. Since the rotor time constant T_R is dependent on rotor resistance and rotor resistance varies substantially due to temperature variations and skin effect, the rotor time constant will affect the accuracy of the flux magnitude and angle estimation. The result will be a deterioration in the quality of control and system performance.

2.3.3 Generalized Field Oriented Control System

In Fig. 2.7, the generalized field oriented control system is presented. Note that the dashed lines indicates a signal that may or may not be required whereas a solid line indicates a signal that is required for field oriented control. Note also that the d - q axes without a superscript ' e ' refer to the field oriented frame (i.e. the synchronously rotating reference frame fixed on the rotor magnetic field). In the diagram there are two inputs, the torque command T_e^* (established by speed or position feedback) and flux command, ψ_r^* (usually constant). The torque reference, v_{qs}^* or i_{qs}^* and magnetizing reference input, v_{ds}^* or i_{ds}^* are produced by separate controllers. These voltages or currents require the matrix U^{-1} , which is the matrix used to convert the synchronously rotating reference frame to a stationary reference frame. Their output is consequently used to convert the two-phase quantities to the physical three-phase quantities via the matrix T^{-1} . For a similar reason, in the feedback path the physical three-phase variables are transformed back into two-phase quantities from a stationary to synchronously rotating reference frame. The above mentioned process is essentially a very complex one, due to the number of coordinate changes from one type of phase to another and from a stationary to a rotating reference

frame and vice versa. It has been made practical by the advent of microprocessors and digital signal processors.

Note that a number of systems are represented in Fig. 2.7. To begin with, the flux magnitude, field angle and torque calculation block may require phase voltages or phase currents or rotor speed or almost any combination of these variables [15]. Second, the controllers block can generate field oriented d - q voltages, or alternatively d - q currents. Following rotating to stationary frame and two-phase to three-phase transformations, these become the voltage, or alternatively current, phase commands for the inverter. Though it may be convenient to use a voltage source inverter (VSI) for voltage phase commands and a current source inverter (CSI) for current phase commands, this is not necessarily the case. For example, current phase commands can be applied to a VSI with current feedback implemented for the inverter itself.

With the introduction of a rotor speed feedback signal, one has a wide range of choices for the speed controller implementation. A typical scheme is shown in Fig. 2.8 [1]. In the diagram, the magnitude of rotor flux linkage, ψ_r and slip frequency, ω_{sl} can be calculated using Eqns. (2.64) and (2.65), respectively. The slip frequency angle, θ_{sl} can be obtained by integrating ω_{sl} . Angle θ_{sl} is the slip angle which is required to adjust the inclination of the d -axis, so that the magnetization of the motor is along this axis. The real rotated electrical rotor angle, θ_r is then added to θ_{sl} to give the instantaneous rotor flux position angle.

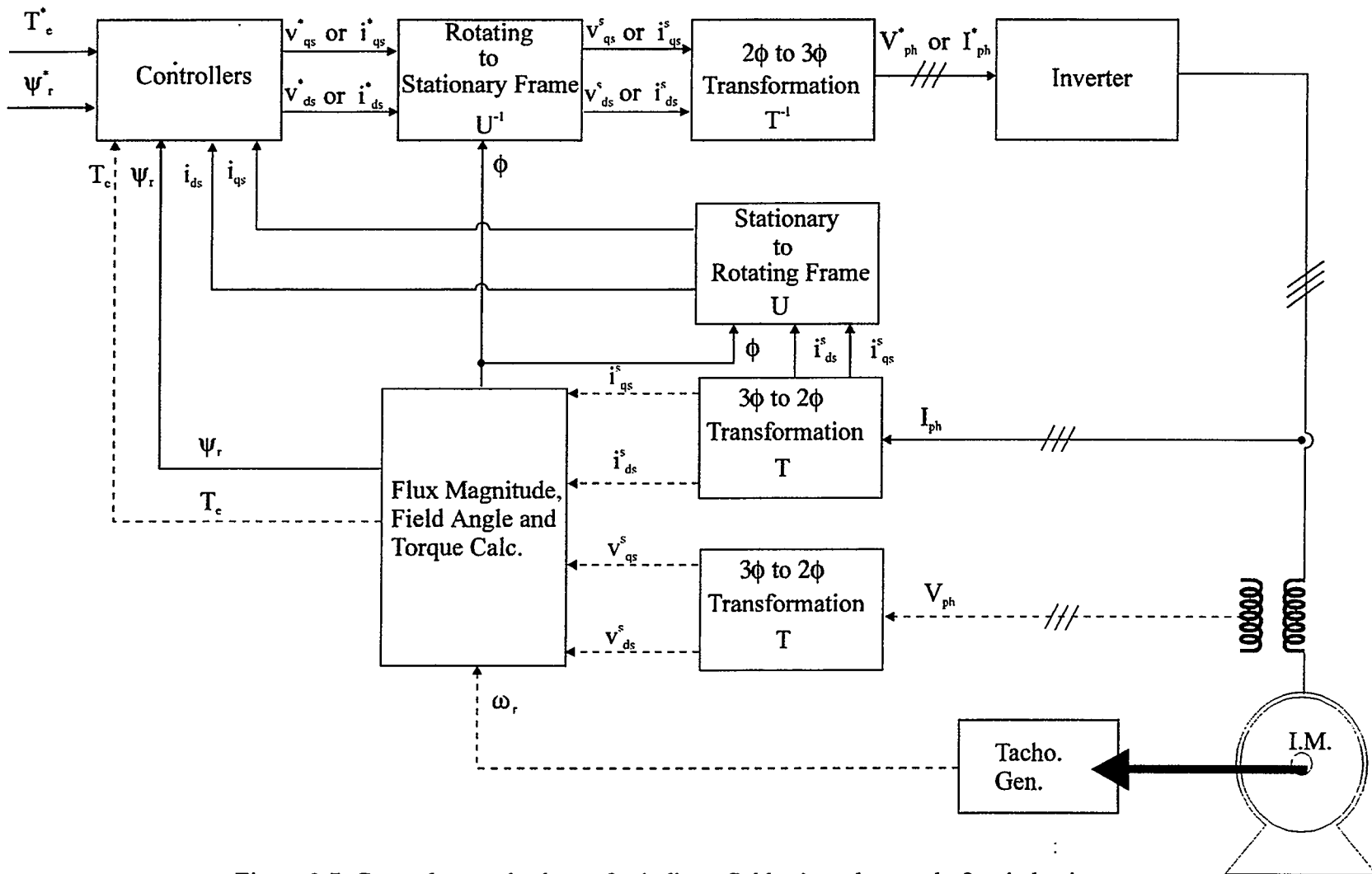


Figure 2.7 General control scheme for indirect field oriented control of an induction motor.

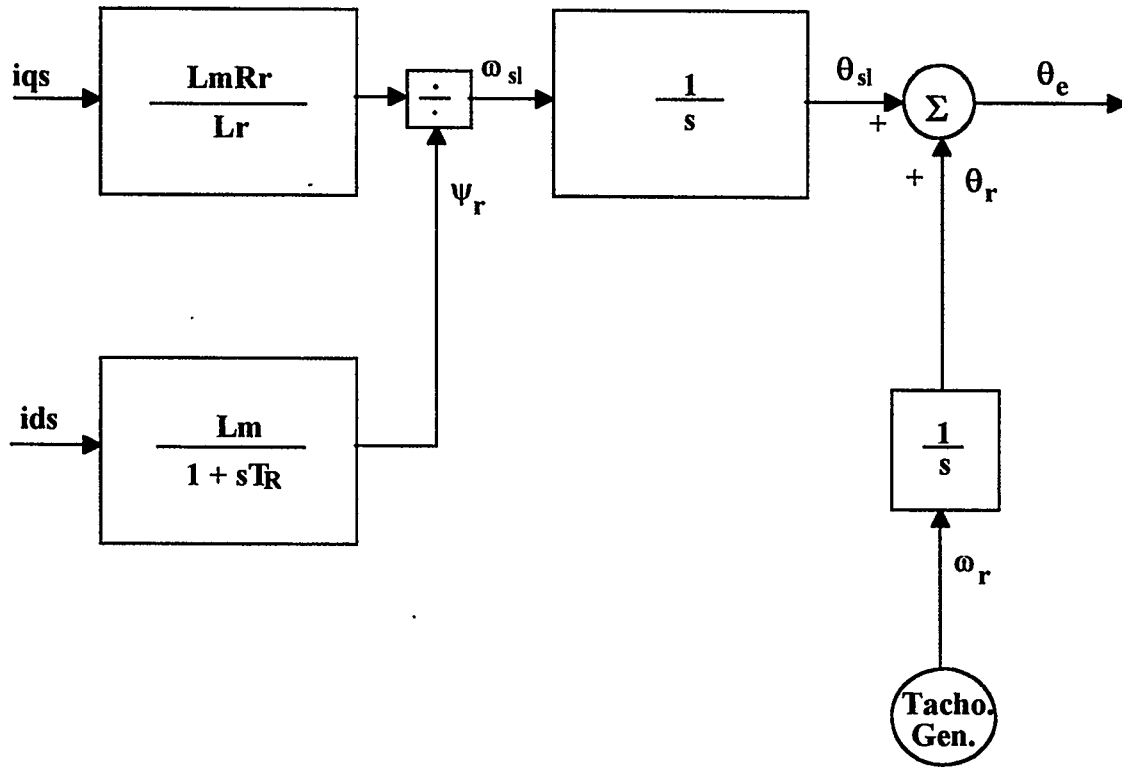


Figure 2.8 Rotor speed feedback scheme

2.4 Flux estimation

The objective of this thesis is to be able to estimate the rotor flux magnitude and angular position as accurately and as quickly as possible (at least as accurately and as quickly as required for field oriented control). Only the indirect method of field oriented control will be discussed in this thesis. With the help of the phasor diagram given in Fig. 2.9 the procedure of obtaining a conventional rotor flux estimation and the principle of indirect field oriented control is explained. In Fig. 2.9, note that the d^s - q^s axes are fixed on the stator while the d^r - q^r axes rotate at synchronous angular velocity ω_e as shown. The

q^e -axis is at the angular position θ_e with respect to the q^s -axis at any instance. The angle θ_e is given by :

$$\begin{aligned}\theta_e &= \theta_r + \theta_{sl} \\ &= (\omega_r + \omega_{sl})t \\ &= \omega_e t\end{aligned}\tag{2.52}$$

where : θ_r = Rotor angular position, $\omega_r t$

θ_{sl} = Slip angular position, $\omega_{sl} t$

$$\omega_e = \omega_r + \omega_{sl}$$

The rotor flux Ψ_r is made up of the air gap flux and the rotor leakage flux and it is aligned to the d -axis. The following equations are written based on the synchronously rotating reference frame $d^e - q^e$ equivalent circuits (i.e. c.f. Fig. 2.4) :

$$\frac{d\Psi_{qr}^e}{dt} + R_r i_{qr}^e + (\omega_e - \omega_r) \Psi_{dr}^e = 0\tag{2.53}$$

$$\frac{d\Psi_{dr}^e}{dt} + R_r i_{dr}^e - (\omega_e - \omega_r) \Psi_{qr}^e = 0\tag{2.54}$$

from Eqns. (2.43) and (2.45) :

$$\Psi_{qr}^e = L_r i_{qr}^e + L_m i_{qs}^e\tag{2.55}$$

$$\Psi_{dr}^e = L_r i_{dr}^e + L_m i_{ds}^e\tag{2.56}$$

Rearranging we have :

$$i_{qr}^e = \frac{1}{L_r} \Psi_{qr}^e - \frac{L_m}{L_r} i_{qs}^e\tag{2.57}$$

$$i_{dr}^e = \frac{1}{L_r} \Psi_{dr}^e - \frac{L_m}{L_r} i_{ds}^e\tag{2.58}$$

By substituting (2.57) and (2.58) into (2.53) and (2.54) the rotor currents can be removed:

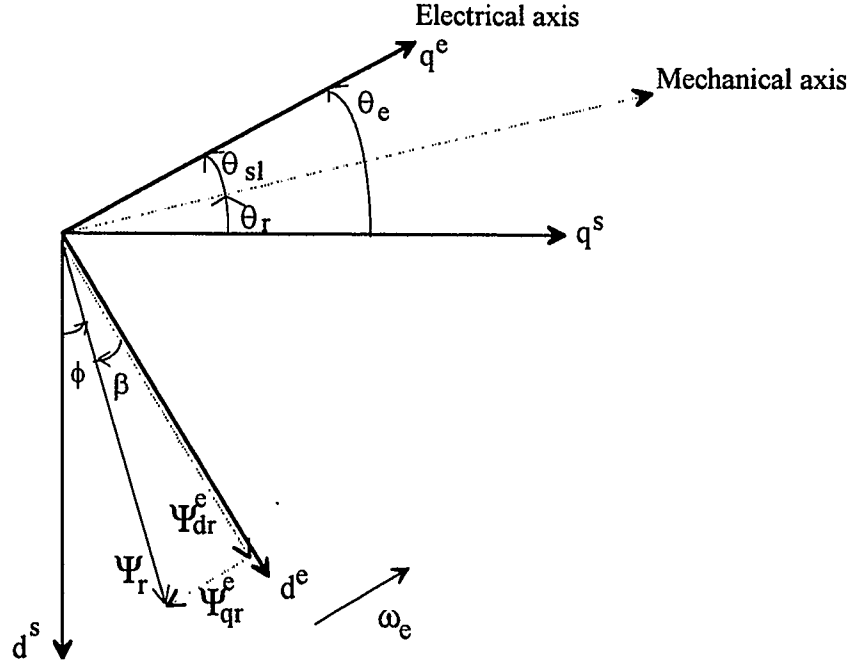


Figure 2.9 Phasor diagram for Indirect Field Oriented Control

$$\frac{d\Psi_{qr}^e}{dt} + \frac{R_r}{L_r}\Psi_{qr}^e - \frac{L_m}{L_r}R_r i_{qs}^e + \omega_{sl}\Psi_{dr}^e = 0 \quad (2.59)$$

$$\frac{d\Psi_{dr}^e}{dt} + \frac{R_r}{L_r}\Psi_{dr}^e - \frac{L_m}{L_r}R_r i_{ds}^e - \omega_{sl}\Psi_{qr}^e = 0 \quad (2.60)$$

where : $\omega_{sl} = \omega_e - \omega_r$

When $\Psi_{qr}^e = 0$, Ψ_r will be aligned with the d^r -axis in the synchronously rotating reference frame (i.e. angle $\beta = 0$, hence the field angle ϕ now equals the synchronous angle θ_e). Let this be the case for steady-state operation with zero torque load, neglecting stator

and rotor losses. This, therefore is a definition for the position of the d^* axis. This is shown in Fig. 2.10.

This figure illustrates the critical difference between θ_e , the position of the stator flux in the stationary frame and ϕ , the position of the rotor flux in the stationary frame. In the literature, there is occasionally some confusion between these two quantities. One purpose of field oriented control is to determine the value of ϕ , the field angle, at all times, as required for coordinate transformations (i.e. for U and U^{-1} matrix in Fig. 2.7). Note that $\phi = \theta_e + \beta$. To re-emphasize, during steady-state, zero load torque operation, $\beta = 0$ neglecting any losses. During steady-state, constant load torque operation, β is a small negative constant quantity (typically between 0° and -30°). Let us now define the d axis by the following conditions (note no superscript is used to denote the field oriented axes) :

$$\Psi_{qr} = \frac{d\Psi_{qr}}{dt} = 0$$

$$\Psi_{dr} = \Psi_r = \text{const}$$

$$\frac{d\Psi_{dr}}{dt} = 0$$

With decoupling control Eqn. (2.59) and (2.60) becomes :

$$\omega_{sl} = \frac{L_m}{\Psi_r} \left(\frac{R_r}{L_r} \right) i_{qs} \quad (2.61)$$

$$\frac{L_r}{R_r} \frac{d\Psi_r}{dt} + \Psi_r = L_m i_{ds} \quad (2.62)$$

or

$$\frac{d\Psi_r}{dt} = \frac{R_r}{L_r} L_m i_{ds} - \frac{R_r}{L_r} \Psi_r \quad (2.63)$$

or

$$i_{ds} = \frac{1 + sT_R}{L_m} \Psi_r \quad (2.64)$$

Substituting (2.64) into (2.61) we have :

$$\omega_{sl} = \frac{1 + sT_R}{T_R} \frac{i_{qs}}{i_{ds}} \quad (2.65)$$

where :

$$T_R = L_r/R_r$$

$$s = d/dt$$

From Eqn. (2.63), we see that rotor flux is a simple first order differential equation which is dependent on the d -axis current of the stator in the synchronously rotating reference frame .

Note that one can derive equations similar to Eqns. 2.55 and 2.56 in the field oriented frame. They are simply :

$$\Psi_{qr} = L_r i_{qr} + L_m i_{qs} \quad (2.66)$$

$$\Psi_{dr} = L_r i_{dr} + L_m i_{ds} \quad (2.67)$$

These last two equations are very relevant to the approach that we have chosen to obtain field oriented control, as is discussed in Chapter 4.

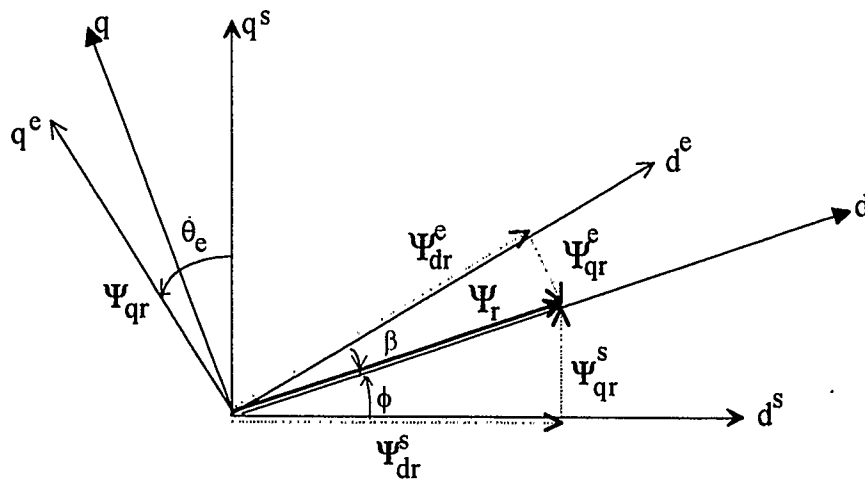


Figure 2.10 Decoupling control

Chapter 3

Artificial Neural Networks

3.1 Introduction to Artificial Neural Networks

The human brain is the most complex computing device known to man. The brain's powerful thinking, remembering and problem solving capabilities have inspired many scientists to attempt computer modeling of its operations.

The neuron is the fundamental cellular unit of the entire nervous system and in particular of the brain. Each neuron is a simple processing unit which receives and combines signals from many other neurons through input elements called dendrites. If the combined signal is strong enough, it activates the firing of the neuron, which produces an output signal; the path of the output signal is along a component of a cell called the axon. The signal arrives at the synapse of the neuron for distribution to the dendrites of other neurons. Note that the signals are transferred by an electrochemical process (i.e. some of the steps are electrical in nature and some are chemical in nature).

The brain consists of tens of billions of neurons densely interconnected. The axon (output path) of a neuron splits up and connects to dendrites (input paths) of other neurons through a junction referred to as a synaptic cleft. The transmission across this junction is chemical in nature and the amount of signal transferred depends on the presence of certain chemicals (neuro-transmitters) released by the axon and received by the dendrites. This synaptic efficiency (or "strength") is what is modified when the brain

learns. The synapse combined with the processing of the information in the neuron form the basic memory mechanism of the brain.

The major structures of a typical nerve cell is shown in Fig. 3.1. The cell consists of dendrites, the cell body, and a single axon. The axon of many neurons is surrounded by a membrane called myelin sheath. Nodes of Ranvier interrupt the myelin sheath periodically along the length of the axon. Synapses connect the axons of one neuron to various parts of other neurons.

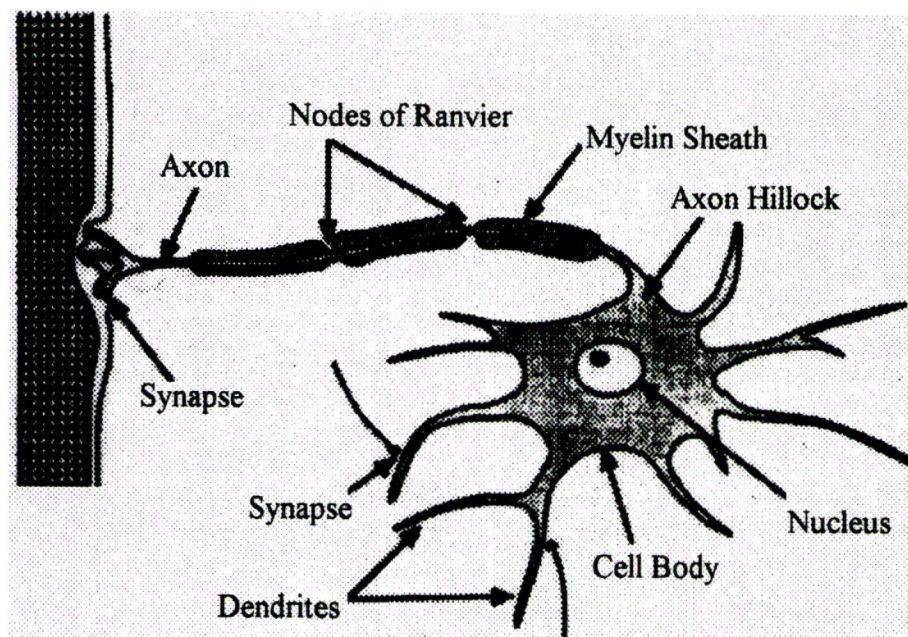


Figure 3.1 Biological nerve cell

In an artificial neural network, the unit analogous to the biological neuron is referred to as the “processing element.” A processing element (PE) has many input paths (dendrites) and combines, usually by simple weighted summation, the values of these input paths. The result is an internal activity level for the processing element. The combined input is then modified by a transfer function. This function can be a threshold function

which passes information only if the combined activity level reaches a certain level, or it can be a continuous function of the combined input. The output value of the transfer function is generally passed directly to the output path of the processing element. The output paths of a processing element can be connected to the input paths of other processing elements through connection weights which correspond to the synaptic

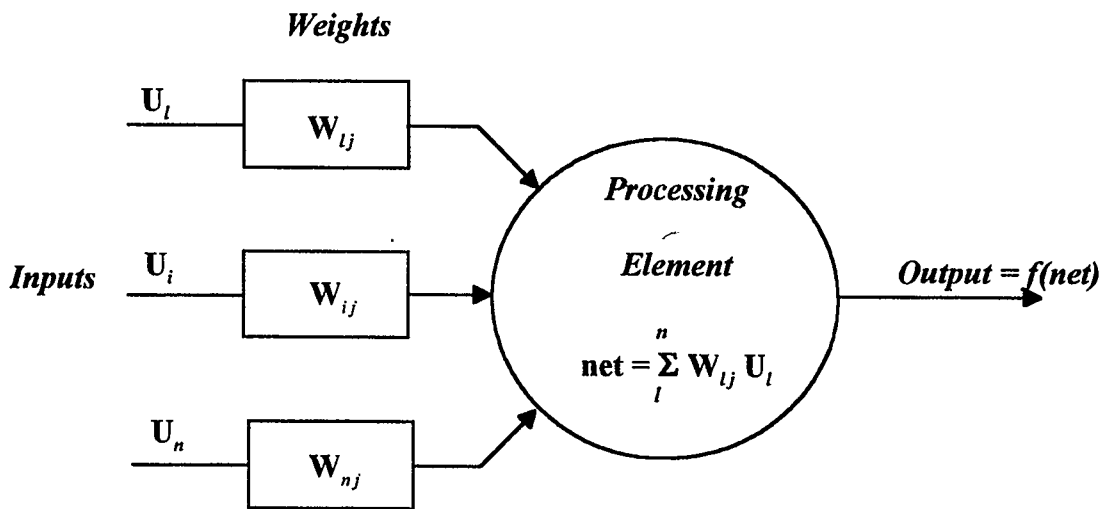


Figure 3.2 Schematic of a processing element (PE).

strength of neural connections. Since each connection has a corresponding weight, the signals on the input lines to a processing elements are modified by these weights prior to being summed. Thus, the summation function is a weight summation. In itself, this simplified model of a neuron is not very interesting; however the interesting effects result from the ways neurons are interconnected. In this chapter, a thorough review and background of artificial neural networks is provided.

3.2 History of Artificial Neural Networks

In 1943, a neurobiologist, Warren McCulloch and a statistician, Walter Pitts, published a watershed paper titled “A Logical Calculus of Ideas Imminent in Nervous Activity.” This paper was an inspiration that helped to launch three diverse fields. One of these was early digital computers, as John von Neumann saw the paper as a blueprint for “electronic brains.” Marvin Minsky, one of the most prominent researchers in Artificial Intelligence, became enthralled with the idea of macroscopic intelligence from this paper, which later led to his interest in black-box macroscopic intelligence, the birthplace of *expert systems*. Frank Rosenblatt, a compatriot of Minsky at the Bronx High School of Science, became intrigued with the computations of the eye and this interest led him to invent the *perceptron* [16].

In 1956, artificial intelligence pioneers Marvin Minsky, John McCarthy, Nathaniel Rochester and Claude Shannon organized the first conference on artificial intelligence. It was at this conference that researchers from all over the world gathered to discuss the potential use of computers in simulation of “every aspect of learning or any other feature of intelligence.” It was at this landmark conference that the fields of neural computing and artificial intelligence were launched. Nathaniel Rochester of IBM Research, presented a neural network model that he had been building. Using several hundred simulated neurons and interconnections, Rochester constructed a system to explore how such a network would respond to environmental stimuli. The results of this model consisted of piles of network generated numerical data, which Rochester did not know how to interpret. This was the first known software simulation of neural computing networks.

3.2.1 Rosenblatt's Perceptron

In 1957, Frank Rosenblatt at Cornell, published the first major research project in neural computing. The development of an element called a "*perceptron*." Rosenblatt's perceptron sparked a great amount of research interest in neural computing.

The perceptron is a pattern classification system which could identify both abstract and geometric patterns. The first perceptron was capable of learning and was robust in that its operation was only degraded after damage to component parts. In addition, the perceptron was capable of making limited generalizations and could properly categorize patterns despite noise in the input.

The perceptron was primarily aimed at optical pattern recognition. The grid of 400 photocells, corresponding to light sensitive neurons in the retina, received the primary optical stimuli. These photocells were connected to associator units as shown in Fig. 3.3, that collected electrical impulses from the photocells. Connections between the associator units and the photocells were made by randomly wiring the associators to the cells. If the input from the cells exceeded a certain threshold, the associator units signaled response units to produce output.

Since it was a developmental device, the perceptron also had certain limitations. One of these would be emphasized by Minsky and Papert, who discovered that the perceptron was unable to represent the basic Exclusive OR (XOR) function. This is the result of the linear nature of the perceptron. One of the most significant changes made since Rosenblatt's work in the 60's has been the development of multi-layer systems which can learn and categorize complex class categories. This is typically achieved by using a

non-linear transfer function, but can also arise from normalization and competition.

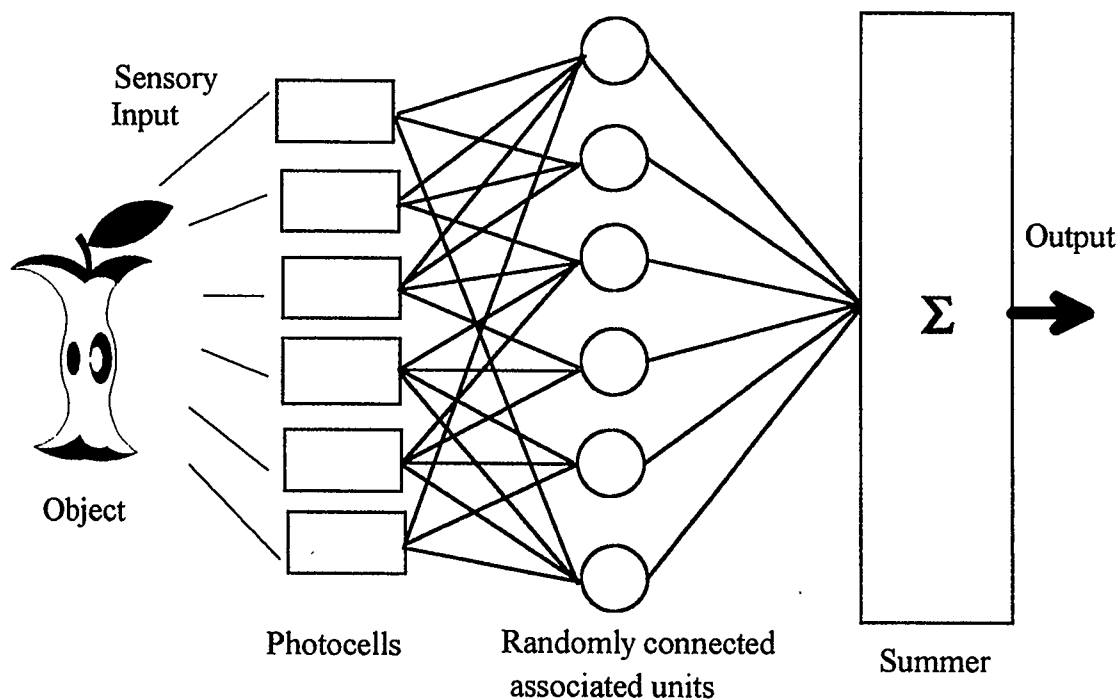


Figure 3.3 The perceptron

3.2.2 Minsky and Papert's Perceptron

In the mid 1960's, Marvin Minsky and Seymour Papert, both of MIT's Research Laboratory of Electronics, began work on an in-depth critique of the perceptron. The book *Perceptrons*, published in 1969 (and re-published recently), is a detailed mathematical analysis of an abstract version of Rosenblatt's perceptron. The main result of this book was Minsky and Papert's conclusion that the perceptron cannot handle inputs that are visually non-local. The conclusion this work reported to the world—that the perceptron and neural computing were basically “not interesting” subjects to

study—drastically decreased the amount of funds and therefore research in neural computing at the time.

3.2.3 Widrow's Adaptive Linear Element

In 1959, Bernard Widrow, at Stanford, developed an adaptive linear element called, “adaline” (*Adaptive Linear Neuron*), based on simple neuron-like elements. The Adaline and a two layer variant, the “madaline” (Multiple Adaline) were used for a variety of applications including speech recognition, weather prediction, character recognition and adaptive control. Widrow used the adaptive linear element algorithm to develop adaptive filters that eliminated echoes on phone lines. This was the first time a neural computing system was applied to a major real-world problem.

3.2.4 Kohonen's Network

Teuvo Kohonen of Helsinki Technical University in Finland has been doing fundamental work in adaptive learning and associative memories since the early 1970's. He is responsible for the description and analysis of a large scale of adaptation rules: rules in which weights are modified in a manner only dependent on the previous weight value and the post and pre-synaptic values.

Another contribution that Kohonen made is the principle of competitive learning in which processing elements compete to respond to an input stimulus and the winner adapts itself to respond more strongly to that stimulus. Such learning is unsupervised in

that the internal organization of the network is governed only by the input stimuli. The competitive learning paradigm was the result of a general study of self-organizing maps which was motivated by various physiological observations about how information received at the sensory organs is mapped topologically onto one and two dimensional areas of the brain.

3.2.5 Back-propagation Networks

The back-propagation network, the most popular network for current applications of artificial neural networks was first formalized in 1974 by Werbos [17], and later by Parker [18] and Rumelhart and McClelland [9]. One of the major research groups of recent years has been the PDP (Parallel Distributed Processing) group, started by Rumelhart, McClelland and Hinton in 1982.

Back-propagation is a technique for solving the credit assignment problem posed by the Minsky and Papert in *Perceptrons* [16]. A perceptron network is able to train the output units to learn to classify patterns of inputs, provided that the classes are “linearly separable.” More complex non-linearly separable classes can be separated with a multi-layer network. However, if the output is in error, how does one determine which processing element or inter-connection to adjust? This is the *credit assignment* problem. Back-propagation solves this problem by assuming all processing elements and connections are somewhat to blame for the erroneous response.

3.3 The Back-propagation Approach

Conceptually, a back-propagation network is made up of interconnected nodes arranged in at least three layers. The input layer is passive; it merely receives the input vectors (data patterns) passing into the network. The number of input nodes consequently equals the number of measured data values (vector components) presented to the network. In contrast, both the hidden and output layer actively process data. The output layer, produces the network result as its name suggests. In a back-propagation network, the result is a set of output vectors, one value per output node.

The hidden layer has no direct connection to the input or the output. Introducing this intermediate layer permits back-propagation to model non-linear functions with greater complexity. Choosing the number of hidden layer nodes almost invariably involves experimentation.

A single node has many input values but only one output value. Each input is a single data value presented to the node, usually through a connection from a preceding layer. An extra input known as the bias for the processing element acts as the reference level. On a serial computer, the microprocessor emulates one node at a time. On a parallel computer, each such element typically maps to a single physical processor. Associated with each connection is an adjustable parameter called a weight. Basically, a node calculates the weighted sum of its input, then passes the sum through a function to produce a result. The transfer function is typically a Sigmoid, a monotonic S-shaped curve. The attenuation at the upper and lower limbs of the sigmoid constrains the sum with fixed limits.

The back-propagation algorithm's strength is its ability to change the values of its weights in response to errors. It does this automatically during training, hence, training requires a series of input patterns tagged with their desired output patterns. During training, the network passes each input pattern through the hidden layer to the output layer to generate a result for each output node (see Figs. 3.4(a) to 3.4(f)). It then compares the desired and the actual results. The differences are the output layer errors, which the network passes back to the hidden layer using the same weighted connections. This backward propagation of errors gives the algorithm its name. Subsequently, each hidden node calculates the weighted sum of the back-propagated errors to find its contribution to the known output errors.

After each output and each hidden node find its error value, it adjusts its weight to reduce its error. The equation that changes the weights—called the *delta rule*—is designed to minimize the network's sum-squared error. The network's overall accuracy is improved by the aggregate corrections during training. When the network can process input patterns with sufficient accuracy, the weights are saved to preserve what it has learned.

After training, the network should be tested with known data that was not used in the training data set. The network's accuracy with patterns outside the training set is called generalization and indicates its reliability in an application.

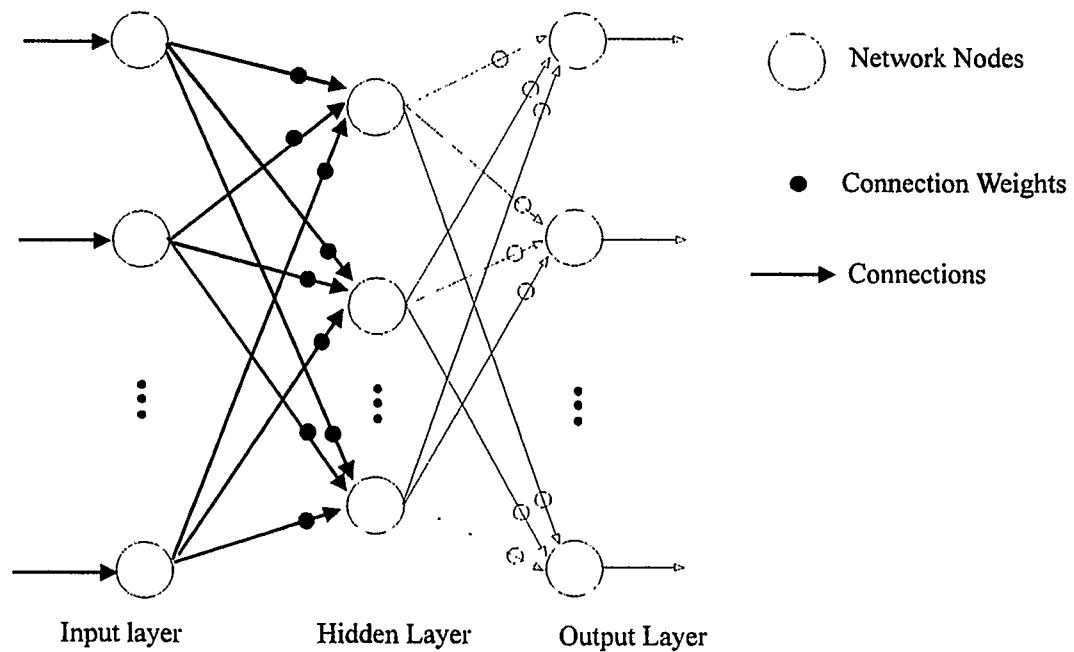


Figure 3.4(a) A typical Back-propagation neural network. During training, the input layer propagates a pattern to all hidden nodes. These calculate a weighted sum of inputs and are passed through a transfer function.

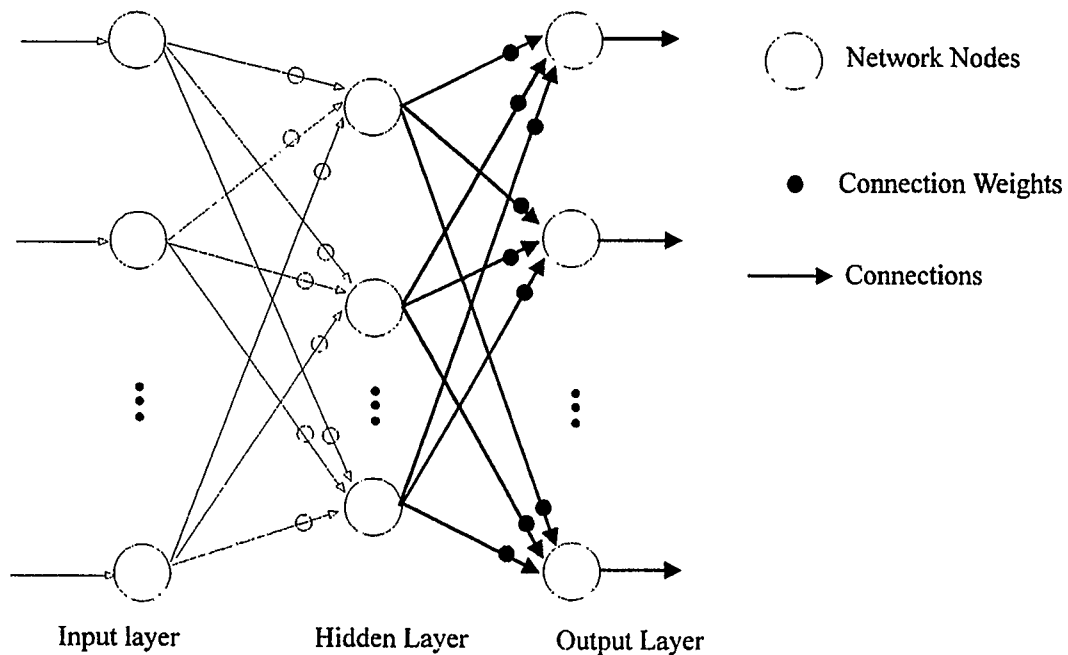


Figure 3.4(b) The hidden nodes propagate their results to all output nodes. Each output node then calculates a weighted sum and passes it typically through the same transfer function to generate an actual result.

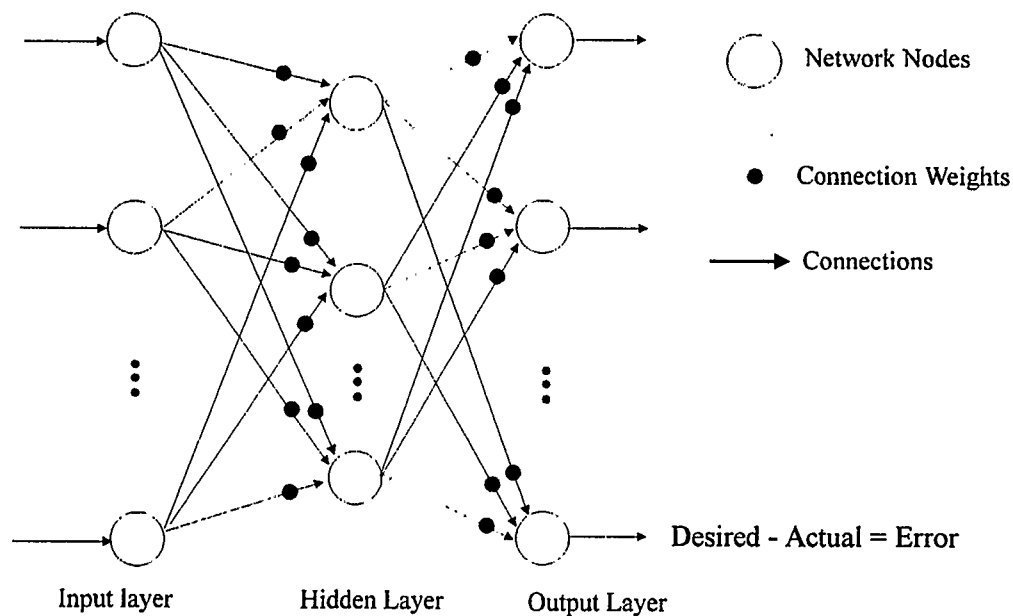


Figure 3.4(c) Each output node subsequently subtracts its actual result from its desired result, which yields the output error.

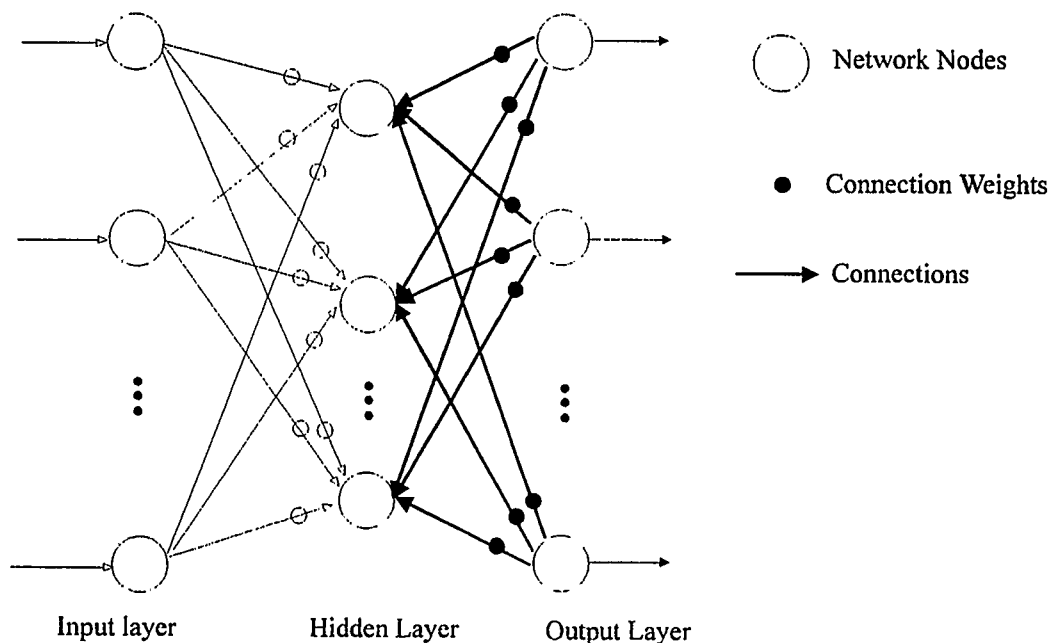


Figure 3.4(d) The output nodes calculate the derivatives of the error vector components with respect to the weights, subsequently passing these derivatives back to the hidden layer. The back-propagation of error gives this neural network its name.

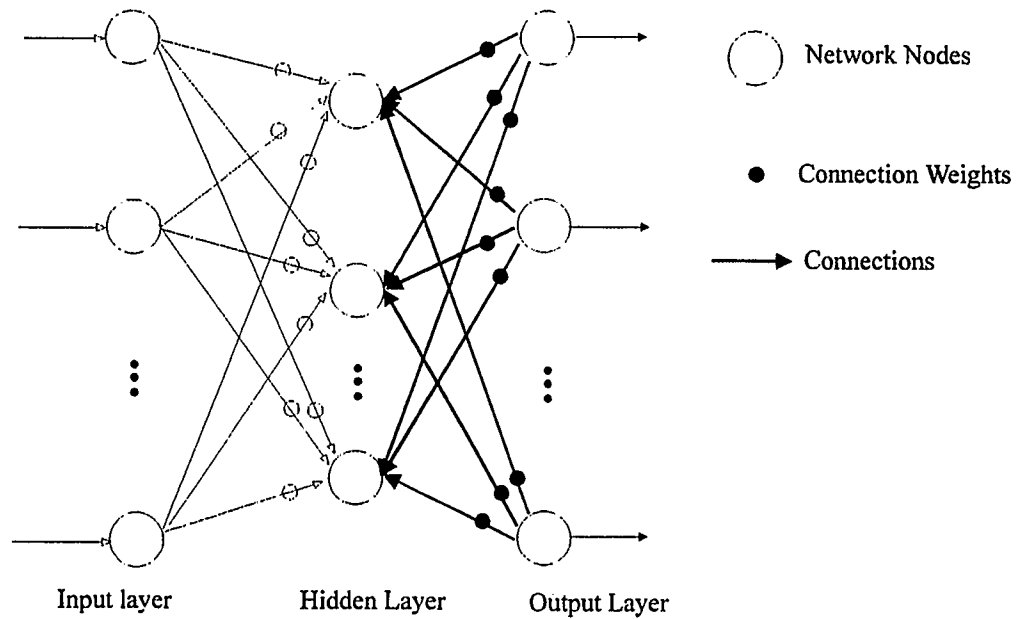


Figure 3.4(e) Each hidden node calculates the weighted sum of the error derivatives to find its contribution to the output error.

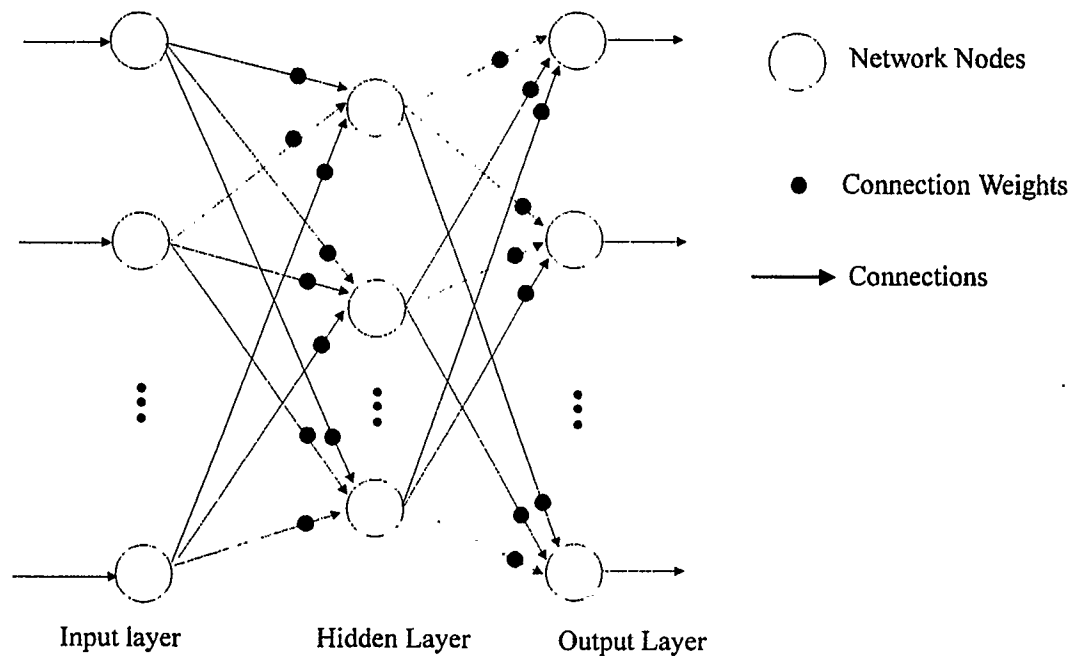


Figure 3.4(f) Each hidden layer node and output layer node changes its weight according to a predetermined mathematical criterion, i.e. the *least mean squares* algorithm, to reduce its error.

When training and testing are completed, the network is ready to process unknown data. Applying a pattern to the input produces a corresponding pattern at the output. The network therefore acts as a *model of a function*, matching input patterns to output patterns. It learns this association solely from the training data, even if the equation describing the function is non-linear, unknown or both.

3.4 The Generalized Delta Rule

In this section, a formal mathematical description of the back-propagation operation is presented. A detailed derivation of the *generalized delta rule* (GDR), which is the learning algorithm for the network is also discussed [19]. Fig. 3.5 serves as a reference diagram for most of the discussions. The back-propagation network is a layered, feedforward network that is fully interconnected by layers. Therefore, there are no feedback connections and no connections that bypass one layer to go to another directly.

A mapping network is defined as one that is able to compute some functional relationship between its input and its output. For simple functions like the mapping of α to $\sin(\alpha)$ where the functional relationship is known, a neural network is not required. However, to perform a complicated mapping where the functional relationship is not known in advance but where some correct mappings are known, a neural network is most applicable. In this situation the ability of neural network to discover its own algorithms is extremely useful.

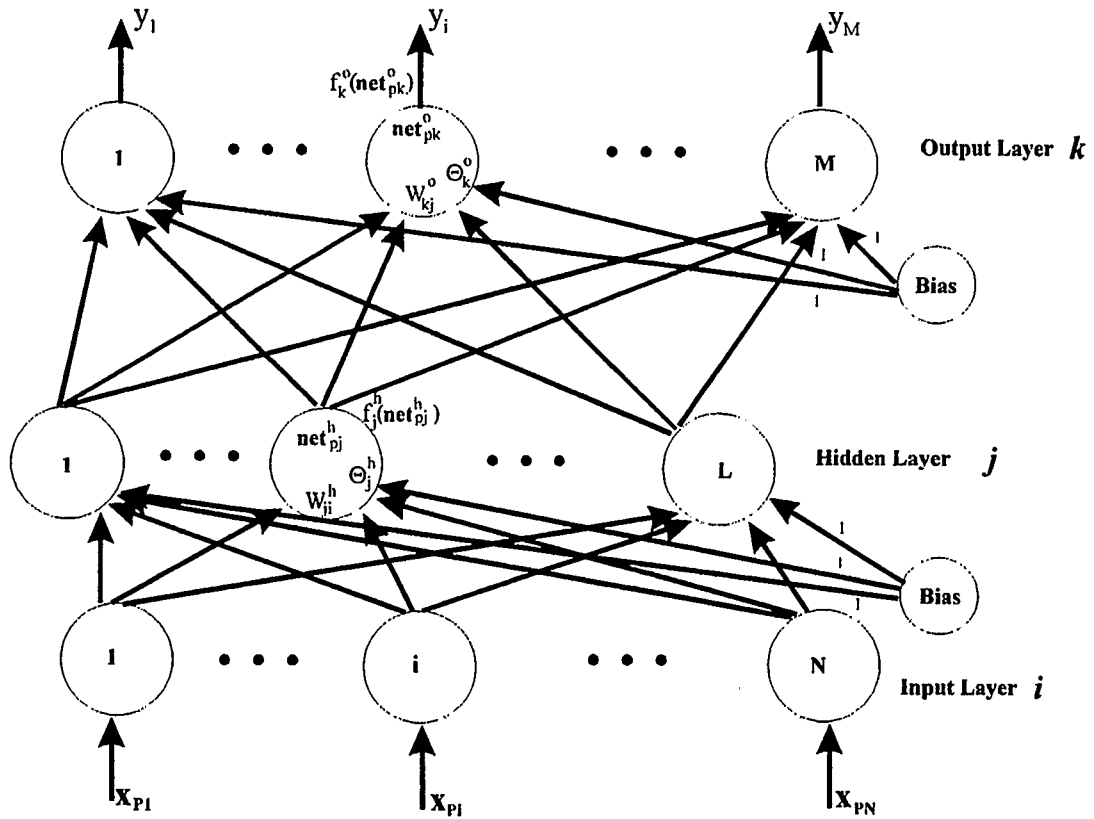


Figure 3.5 Three layer back-propagation network. The bias weights, Θ_j^h and Θ_k^o and the bias units are optional. The bias units provide a fictitious input value of 1 on a connection to the bias weight. The bias weight can be treated like any other weight.

Input vectors pairs P are as follows: $(x_1, y_1), (x_2, y_2), \dots, (x_p, y_p)$, which are examples of a functional mapping $y = \phi(x)$. To train the network so that it will learn the approximation o , where $o = y' = \phi'(x)$, where the “'” denotes the differential operator. Learning in a neural network means finding an appropriate set of weights. The learning technique used here will be a generalization of the *least mean squares* (LMS) rule. Due to possible non-linearity mapping function, as well as its multidimensional nature, the

iterative version of the simple least-squares method called a *steepest-descent technique* is employed.

If an input vector $x_p = (x_{p1}, x_{p2}, \dots, x_{pN})^t$ is applied to input layer (Fig. 3.5), the input units propagate the values to the hidden layers units. The net input to the j^{th} hidden unit is

$$net_{pj}^h = \sum_{i=1}^N w_{ji}^h x_{pi} + \theta_j^h \quad (3.1)$$

where w_{ji}^h is the weight on the connection from the i^{th} input unit, and θ_j^h is the bias term. The h superscript refers to quantities on the hidden layer. Assuming that the activation of this node is equal to the net input; then the output of this node is

$$i_{pj} = f_j^h(net_{pj}^h) \quad (3.2)$$

The equations for the output nodes are

$$net_{pk}^o = \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o \quad (3.3)$$

$$o_{pk} = f_k^o(net_{pk}^o) \quad (3.4)$$

where the “o” superscript refers to quantities on the output layer.

The procedure for training the network is as follows:

- 1) Apply an input vector to the network and calculate the corresponding output values.
- 2) Compare the actual outputs with the correct outputs and determine a measure of the error.
- 3) Determine in which direction, positive or negative, to change each weight in order to reduce the error.

- 4) Determine the amount by which to change each weight.
- 5) Apply the corrections to the weights.
- 6) Repeat items 1 through 5 with all the training vectors until the error for all vectors in the training set is reduced to an acceptable value.

For a network with no hidden layers and a linear output, the weight-change law is called the LMS rule or Delta rule given by :

$$w(t+1)_i = w(t)_i + 2\mu\epsilon_k x_{ki} \quad (3.5)$$

where μ = is a positive constant

x_{ki} = is the i^{th} component of the k^{th} training vector

ϵ_k = is the difference between the actual output and the correct value, $\epsilon_k = (d_k - y_k)$

A similar equation results when the network has more than two layers, or the output functions are non-linear.

3.4.1 Output Layer Weight Updates

Error at a single output unit is defined as $\delta_{pk} = (y_{pk} - o_{pk})$, where the subscript “ p ” refers to the p^{th} training vector, and “ k ” refers to the k^{th} output unit. The actual output is o_{pk} and desired output is y_{pk} . The total squared error is minimized by the gradient delta rule and is given by :

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2 \quad (3.6)$$

For convenience in calculating derivatives later, the factor of 1/2 in Eqn. (3.6) has been added. To determine the direction in which to change the weights, the negative of the gradient of E_p , ∇E_p , with respect to the weights, w_{kj} is calculated. Equation (3.6) becomes :

$$E_p = \frac{1}{2} \sum_k (y_{pk} - o_{pk})^2 \quad (3.7)$$

and

$$\frac{\partial E_p}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k^o}{\partial (net_{pk}^o)} \frac{\partial (net_{pk}^o)}{\partial w_{kj}^o} \quad (3.8)$$

the partial derivatives are determined by the chain rule. The derivative of f_k^o will not be evaluated, but instead will be written as $f_k^{o'}(net_{pk}^o)$. The last term in Eqn. (3.8) is :

$$\frac{\partial (net_{pk}^o)}{\partial w_{kj}^o} = \left(\frac{\partial}{\partial w_{kj}^o} \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o \right) = i_{pj} \quad (3.9)$$

for negative gradient from Eqns. (3.8) and (3.9) :

$$-\frac{\partial E_p}{\partial w_{kj}^o} = (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^o) i_{pj} \quad (3.10)$$

The update of output layer weights are then determined by the following :

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \Delta_p w_{kj}^o(t) \quad (3.11)$$

such that

$$\Delta_p w_{kj}^o(t) = \eta (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^o) i_{pj} \quad (3.12)$$

where η is called the *learning-rate* parameter. In general it is difficult to determine the best value of η . A general rule is to make the learning rate for each node inversely proportional to the average magnitude of vectors feeding into the node. There are several methods to adapt the learning rate as a function of the local curvature of the weight space surface [9, 20, 21, 22]. The value of η is usually a positive number and less than one.

The function $f_k^{o'}$ is obtained by differentiating f_k^o . Thus the function f_k^o must be differentiable. With that in mind, there are two forms of output functions that are of interest here. They are :

$$f_k^o(net_{jk}^o) = net_{jk}^o \quad (3.13)$$

$$f_k^o(net_{jk}^o) = (1 + e^{-net_{jk}^o})^{-1} \quad (3.14)$$

Eqn. (3.13) defines a linear output unit and equation (3.14) defines a *sigmoid*, or logistic function. To determine which output function to use depends on the type of output representation required. If the output units are to be binary, then the sigmoid output function would be appropriate. In addition to being output limiting the sigmoid is also quasi-bistable but differentiable.

In Eqn. (3.13), $f_k^{o'} = 1$; in Eqn. (3.14), $f_k^{o'} = f_k^o(1 - f_k^o) = o_{pk}(1 - o_{pk})$ For the two cases, we have :

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})i_{pj} \quad (3.15)$$

for the linear output, and

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})i_{pj} \quad (3.16)$$

for the sigmoidal output.

The weight update equations are summarized by defining :

$$\begin{aligned} \delta_{pk}^o &= (y_{pk} - o_{pk})f_k^{o'}(net_{pk}^o) \\ &= \delta_{pk}f_k^{o'}(net_{pk}^o) \end{aligned} \quad (3.17)$$

and hence the weight update equation :

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta\delta_{pk}^o i_{pj} \quad (3.18)$$

3.4.2 Hidden Layer Weight Updates

A similar type of calculation is performed for hidden layer weight updates as in the output layer weight updates. A major difference between the output weight updates and the hidden layer weight updates is that in the output layer weight updates, the actual output is known whereas in the hidden layer weight updates the correct outputs of the hidden layer is not known in advance. To verify that the total error E_p is related to the output values on the hidden layer we need to look at Eqn. (3.7) :

$$\begin{aligned} E_p &= \frac{1}{2} \sum_k (y_{pk} - o_{pk})^2 \\ &= \frac{1}{2} \sum_k (y_{pk} - f_k^o(net_{pk}^o))^2 \\ &= \frac{1}{2} \sum_k (y_{pk} - f_k^o(\sum_j w_{kj}^o i_{pj} + \theta_k^o))^2 \end{aligned}$$

From Eqns. (3.1) and (3.2), it is shown that i_{pj} depends on the weights on the hidden layer.

The gradient of E_p calculated with respect to the hidden layer weights are as follows :

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ji}^h} &= \frac{1}{2} \sum_k \frac{\partial}{\partial w_{ji}^h} (y_{pk} - o_{pk})^2 \\ &= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial (net_{pk}^o)} \frac{\partial (net_{pk}^o)}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial (net_{pj}^h)} \frac{\partial (net_{pj}^h)}{\partial w_{ji}^h} \end{aligned} \quad (3.19)$$

by using Eqn. (3.6)-(3.18) in the above we have :

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^o) w_{kj}^o f_j^{h'}(net_{pj}^h) x_{pi} \quad (3.20)$$

Updating the hidden layer weights in proportion to the negative of Eqn. (3.20) we get :

$$\Delta_p w_{ji}^h = \eta f_j^{h'}(net_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{o'}(net_{pk}^o) w_{kj}^o \quad (3.21)$$

where η again is the learning rate.

By using Eqn. (3.17) i.e. the definition of δ_{pk}^o we can write :

$$\Delta_p w_{ji}^h = \eta f_j^{h'}(net_{pj}^h) x_{pi} \sum_k \delta_{pk}^o w_{kj}^o \quad (3.22)$$

It is important to note that every weight update on the hidden layer depends on *all* the error terms, δ_{pk}^o , on the output layer. In other words, the known errors on the output layer are propagated backwards to the hidden layer to determine the appropriate weight changes on that layer. The hidden layer error term can be defined as :

$$\delta_{pj}^h = f_j^{h'}(net_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \quad (3.23)$$

to become analogous to those for the output layer :

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \eta \delta_{pj}^h x_i \quad (3.24)$$

3.5 Training Considerations

The neural network is capable of forming arbitrarily close approximations to any continuous non-linear mapping using the back-propagation algorithm. However, there are a few practical considerations. The first is choosing the network size. The second is the training data selection and, lastly, the complexity of learning.

3.5.1 Network Size

The neural network is able to map any non-linear continuous function provided that the size of the network grows in proportion to the complexity of the function [23]. In general, it is not known what network size works best for a given problem, since each problem demands different capabilities from the network. Nevertheless, choosing a correct network size is important, too small a network size will not be able to form a good model of the problem. On the other hand, too large a network may be “overly capable” [24], i.e. the network may be able to give several solutions that are consistent with the training data

but are likely to be poor approximations to the actual problem. Ideally, we would like a network size that best matches the capability of the network to the structure of the underlying problem, in other words we would like a network whose size best captures the structure (or intricacies) of the underlying problem. Generally, because there is no prior knowledge of the problem, a methodical trial and error approach is adopted to find the optimal network size of the problem. The following guidelines are used in determining network size,

- 1) Start with the smallest possible network and then gradually increase the size until performance levels off. Alternatively, start with a large network and then apply a pruning technique that destroys weights and/or nodes that do not contribute to the solution [25].

- 2) For a fully connected neural network no more than four-layers are required to perform a particular task, i.e. one input layer, one output layer and two hidden layers. It has been suggested that the number of hidden layer nodes should not be more than the number of training samples, it is almost always the case that the number of hidden layer nodes are much less than the number of training samples [26]. Otherwise, the network will simply “memorize” the training samples resulting in poor generalization of the problem.

The main idea is to use as few hidden layers nodes (units) as possible. This is because each additional node adds to the load on the central processing unit (CPU) during simulations.

3.5.2 Training Data

Generally, the larger the number of training data samples the better the data is capable of describing the underlying problem. The training data's objective is to be able to generalize the entire underlying problem with the given amount of data samples. Generalization is a measure of how well the network performs on the actual problem once training (learning) is complete. The back-propagation neural network is good at generalization. Given several different input vectors, all belonging to the same class, a back-propagation neural network is capable of learning key significant similarities in the input vectors, while irrelevant data is ignored. In order for the network to perform adequately, the training data provided must cover the entire input range of operation. During the training process, the training vector pairs selected should be selected randomly from the data space if possible.

3.5.3 Complexity of Learning

After determining the correct network size and proper training data set, it turns out that finding the correct weights for a network is an inherently difficult problem. The learning algorithm employed for this thesis work (back-propagation) is based on gradient search techniques, and is slow in finding local weight solutions to a problem. To explain its slow speed in learning (not to be confused with the speed of execution) we need to characterize the error surface which is being searched. For multilayered networks the surfaces can be quite severe [27], these surfaces tend to have large areas of flatness as well as extreme steepness and not much in between. It is difficult to determine if a solution has

been reached since the transient flat areas “look” very much like minima, i.e. gradient is very small. The selection of the value of η (the learning rate) has a significant effect on system performance. Usually, η is a small value of 0.05 to 0.25—to ensure that the network settles to a solution. A small value of η will cause the network to have a large number of iteration to achieve a solution. Increasing the value of η as network error decreases will often speed up convergence. Too large of an increase in η may cause instabilities when it reaches the steeper portions of the error surface.

Another way to speed up convergence is to add a momentum term in the weight change Eqn. (3.18). When calculating the weight change value $\Delta_p w$, a fraction of the previous change is added to the weight change :

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} + \alpha \Delta_p w_{kj}^o(t-1) \quad (3.25)$$

where $0 < \alpha < 1$. This term makes the current search direction an exponentially weighted average of past directions, and helps keep weights moving across flat sections of the error surface after they have descended from steep portions.

3.5.4 Termination Criterion

The termination criterion is determined firstly by the magnitude of the gradient, i.e. the learning algorithm can be terminated when the gradient is sufficiently small, since by definition the gradient will be zero at the minimum. Secondly, termination may come as output error falls below a certain threshold. Finally, the learning process may stop at the

completion of a fixed number of iterations, which will not guarantee that a minimum is reached.

Chapter 4

Implementation of Field Oriented Control

4.1 Introduction

In this chapter, the design of a field oriented control drive system is discussed. The control scheme for the FOC chosen is based on the availability of a voltage source inverter in the power electronics lab of the University of Calgary and thus would facilitate the future implementation of this drive system.

With the successful computer simulation of an induction machine model (see appendix A), a field oriented control system can be designed and simulated. For this thesis work the major blocks involved in this process are the induction motor model, a PWM voltage source inverter, a single proportional-integral (P.I.) controller for rotor flux, two P.I. controllers for d - q field currents and the possible use of a P.I. controller for speed control (many of the initial neural network flux estimator tests were carried out without the need for speed feedback). Note that the P.I. controllers are chosen only because of the simplicity of their simulation. This is quite sufficient for testing of the artificial neural network flux estimator, as discussed in the next chapter.

4.2 System Description

Recall the discussion on the generalized field oriented control system in (section 2.3.3). For convenience, Fig. 2.7 is reproduced here in Fig. 4.1. With the objective of one day implementing a high performance speed controlled field oriented

control induction motor drive, a block diagram of a drive system is given in Fig. 4.2(a) (limiters not shown). The entire system depicted in Fig. 4.2(a), has been coded in C (see Appendix B). This program was used to generate all training data (c.f. Chapter 5) and test data (c.f. Chapter 6) for the artificial neural network flux estimator. Note that we have chosen to generate the d - q voltages in the field oriented frame (with simple P.I. controllers) from the field oriented i_{ds} and i_{qs} feedback quantities. These voltages are then rotated (using U^{-1}) and then transformed (using T^{-1}) for direct application to a voltage source inverter.

4.2.1 Rotor Flux Control

As seen in Fig. 4.2(a), the coordinate transformation can be obtained only with an accurate knowledge of the field angle ϕ . Further, in field oriented control, as the motor speed starts to increase, it is desirable for the rotor flux to reach maximum flux as fast as possible and be maintained at that flux magnitude throughout its operating range (field weakening is not considered here). In order to achieve this required fast response, a PI rotor flux controller has been employed shown in Fig. 4.2(b), which is a subsection of Fig. 4.2(a).

In this chapter all results presented are based on flux magnitude and field angle calculations as determined using the Eqns. 2.46, 2.66 and 2.67. Note that this will be done by the neural network flux estimator in the eventual implementation of the system depicted in Fig. 4.2(a).

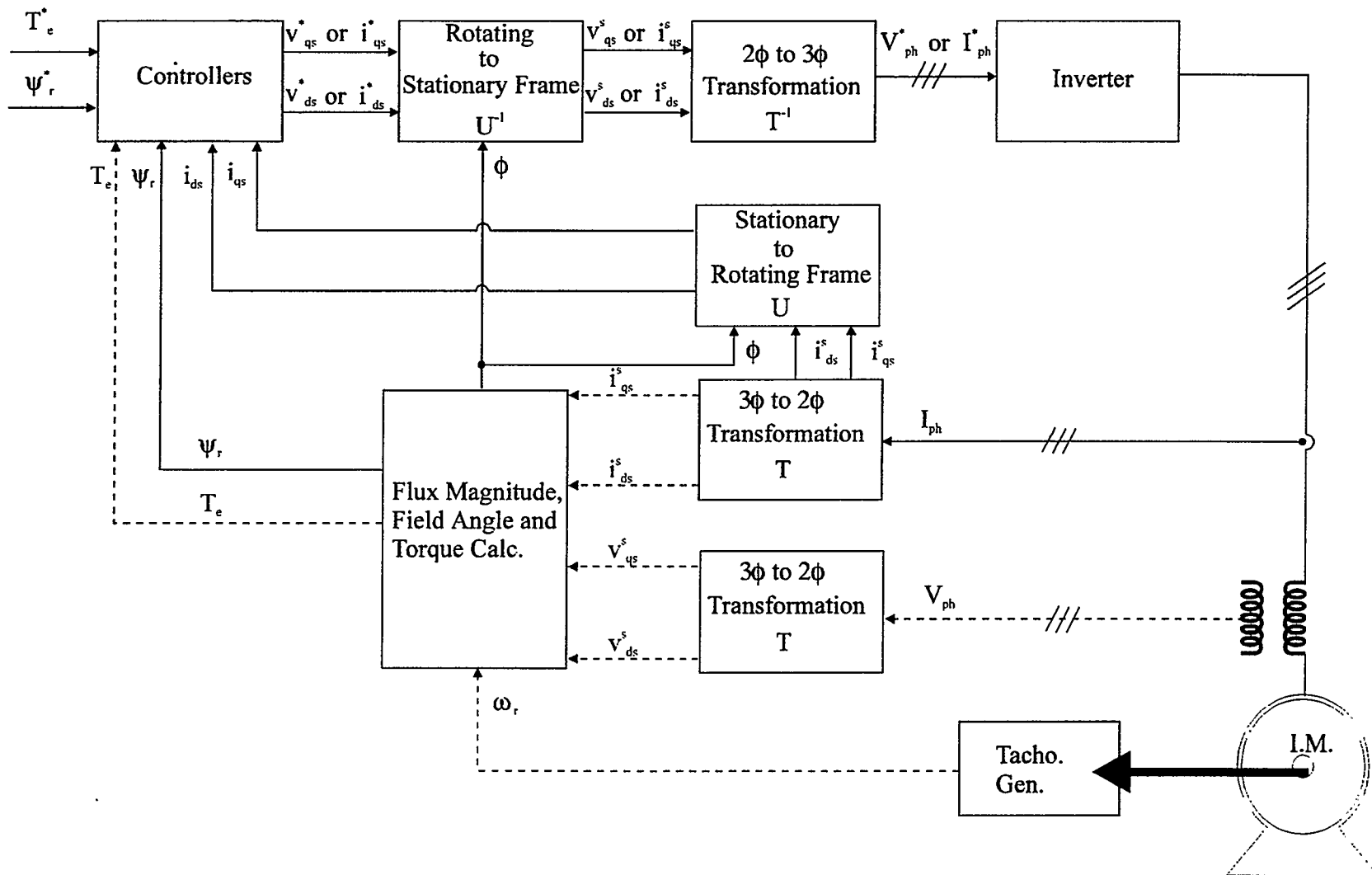


Figure 4.1 General control scheme for indirect field oriented control of an induction motor. (same as Fig. 2.7)

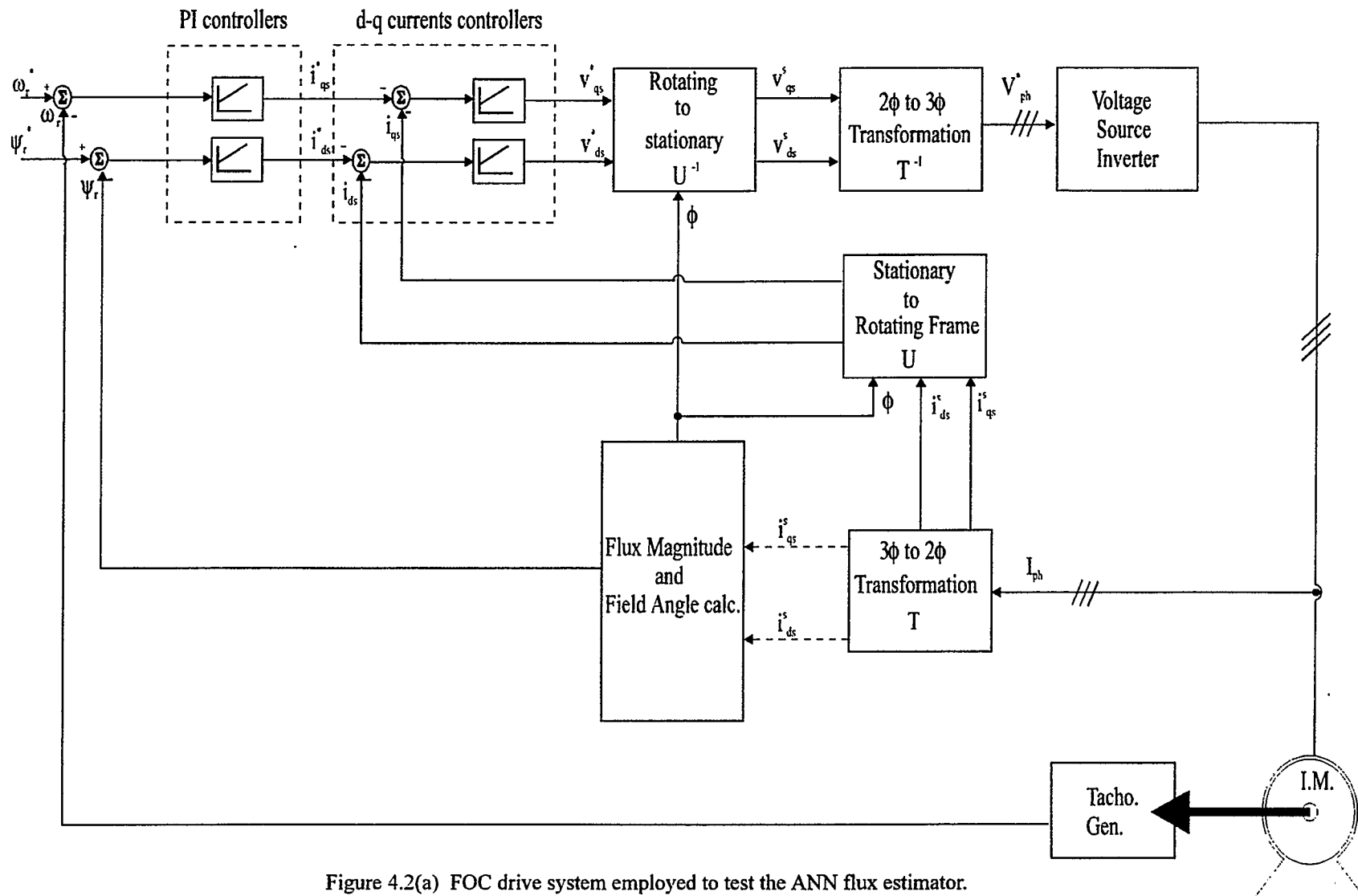


Figure 4.2(a) FOC drive system employed to test the ANN flux estimator.

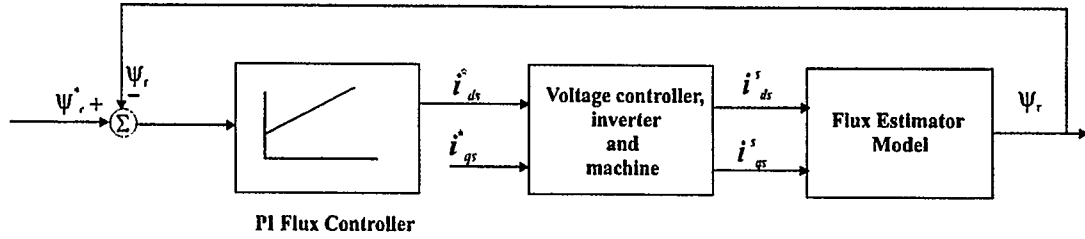


Figure 4.2(b) PI rotor flux controller

4.2.2 Current Controllers

For FOC to be successful, the correct stator currents must be produced, i.e. the d - q axes are decoupled. Therefore, for drive systems using a voltage source inverter (as in this case) it is necessary to have current controllers to regulate the stator voltages in order to impress the desired currents. In the proposed FOC, current control is performed in the field oriented reference frame, where i_{ds}^* is obtained from the rotor flux controller and where i_{qs}^* is controlled independently (c.f. Fig. 4.2(a)).

4.2.3 PI Controller Tuning

The procedure to tune the individual PI controllers is straight forward. Firstly, the current controllers are tuned and then the rotor flux PI parameters are tuned. In the case of the current controllers, the proportional control coefficient is tuned first until the drive system just begins to become unstable (to determine a conservative choice), then the integral coefficient is tuned to have the best steady state error. The same procedure is adopted for the rotor flux controller tuning. The results of optimal tuning for these PI controllers can be seen in Figs. 4.3-4.5 (for 30 hp motor).

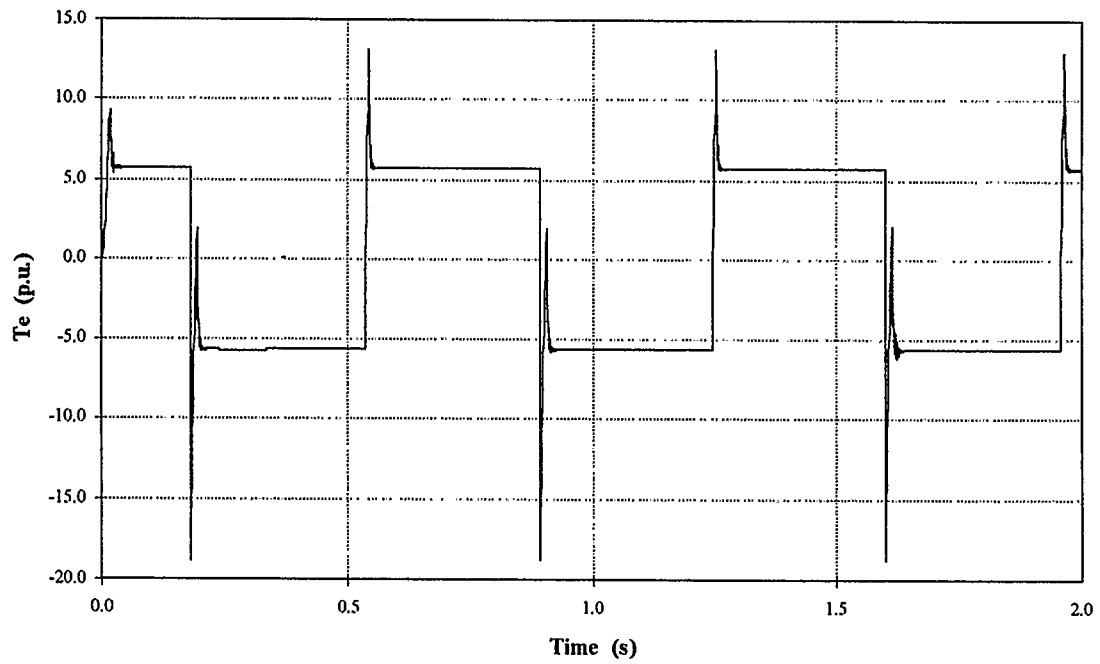


Figure 4.3 Electromagnetic torque with field oriented control implemented.

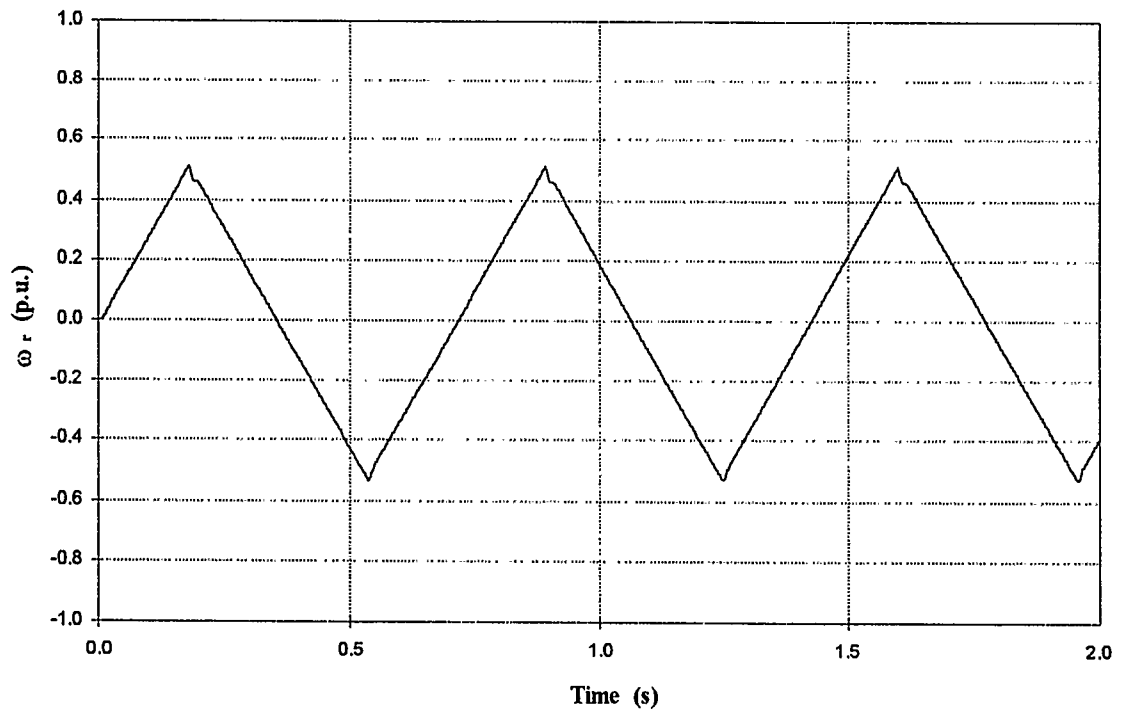


Figure 4.4 Rotor speed with field oriented control implemented.

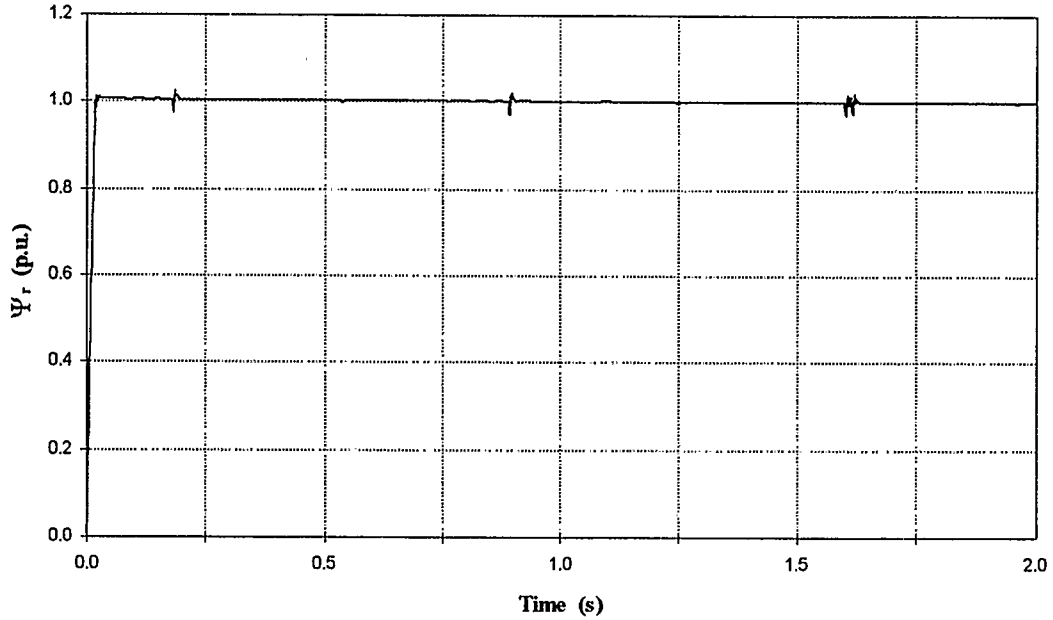


Figure 4.5 Rotor flux magnitude with field oriented control implemented.

As seen from Fig. 4.3, the electromagnetic torque (T_e) reaches steady-state quite quickly (in less than 0.03 s), although, there is an oscillatory transients during switching. The rotor speed curve seen in Fig. 4.4 is produced by letting $I_{qs}^* = 6.0$ p.u for 0.2 s, and then letting $I_{qs}^* = -6.0$ p.u. for 0.4 s and so on. As seen from Fig. 4.5, the rotor flux magnitude is able to reach 1.0 p.u. in less than 0.01 s with FOC implemented, whereas without FOC it took 1.5 s to reach a value of 1.0 p.u. (c.f. Appendix A).

The field angle ϕ is responsible for the precise location of the rotor flux field in field oriented control of the induction motor. This is crucial to the success of FOC, as this field angle is used in the rotation matrix to convert variables in the synchronously rotating frame into their counterparts in the stationary reference frame and vice versa. Figs. 4.6 and 4.7 show $\sin(\phi)$ and $\cos(\phi)$ for the induction motor, respectively.

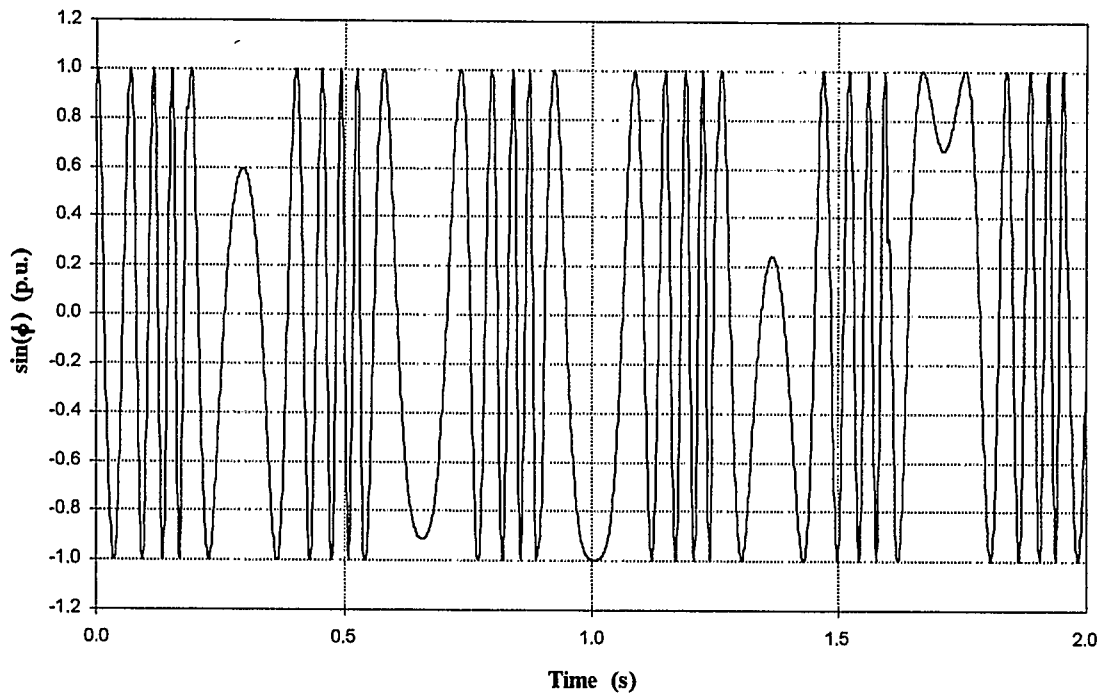


Figure 4.6 $\sin(\phi)$ with field oriented control implemented.

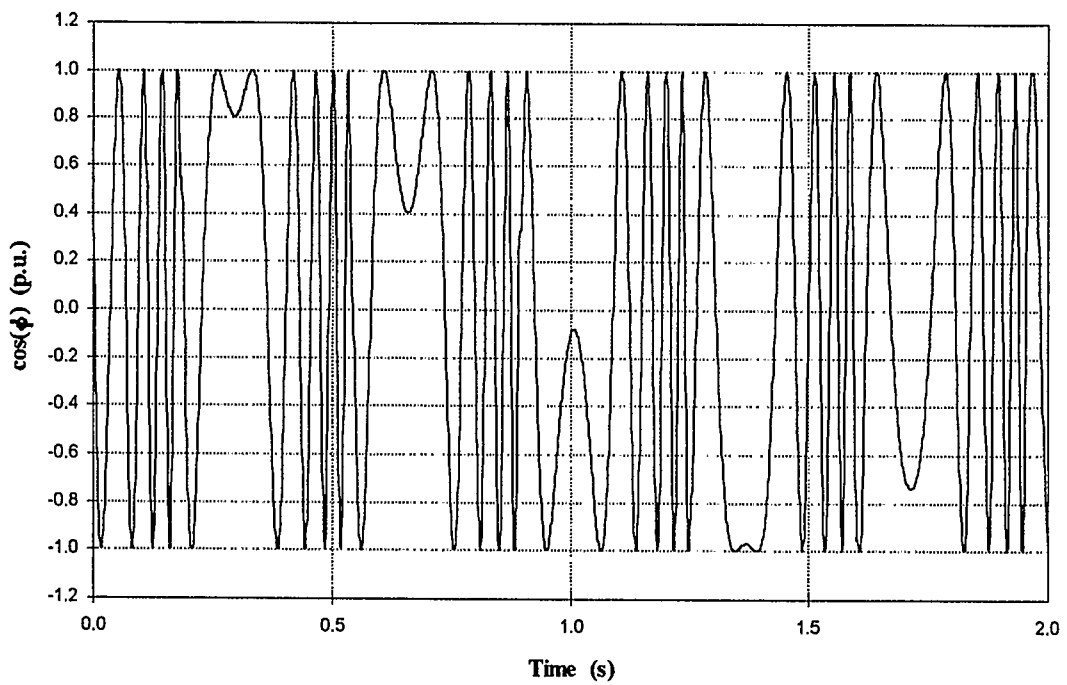


Figure 4.7 $\cos(\phi)$ with field oriented control implemented.

The d-q axes currents in the stationary reference frame are shown in Figs. 4.8 and 4.9.

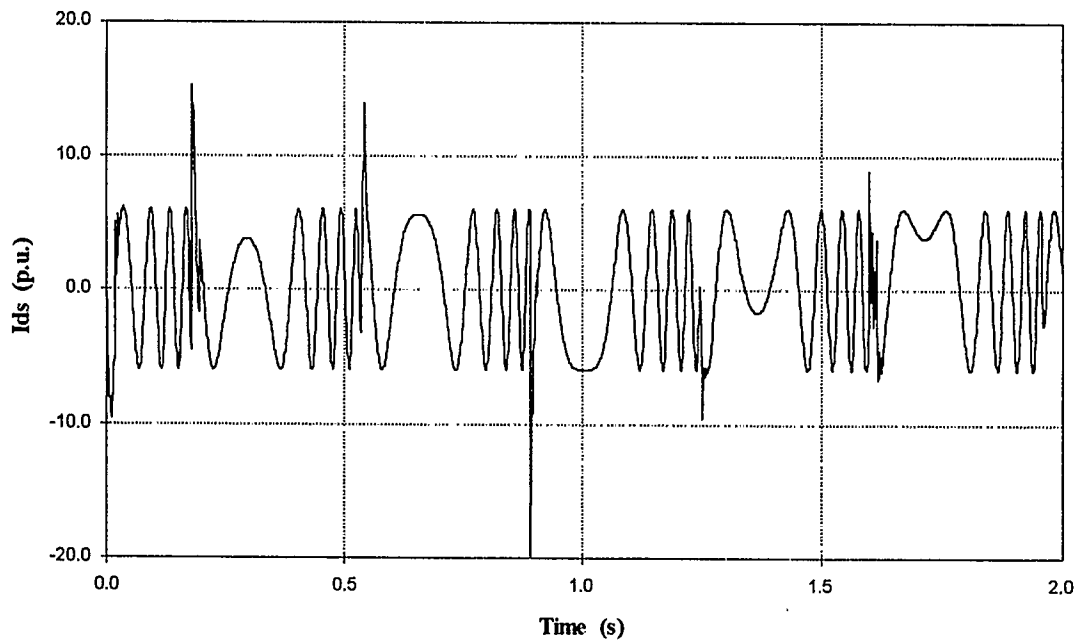


Figure 4.8 I_{ds} in stationary ref. frame with field oriented control implemented.

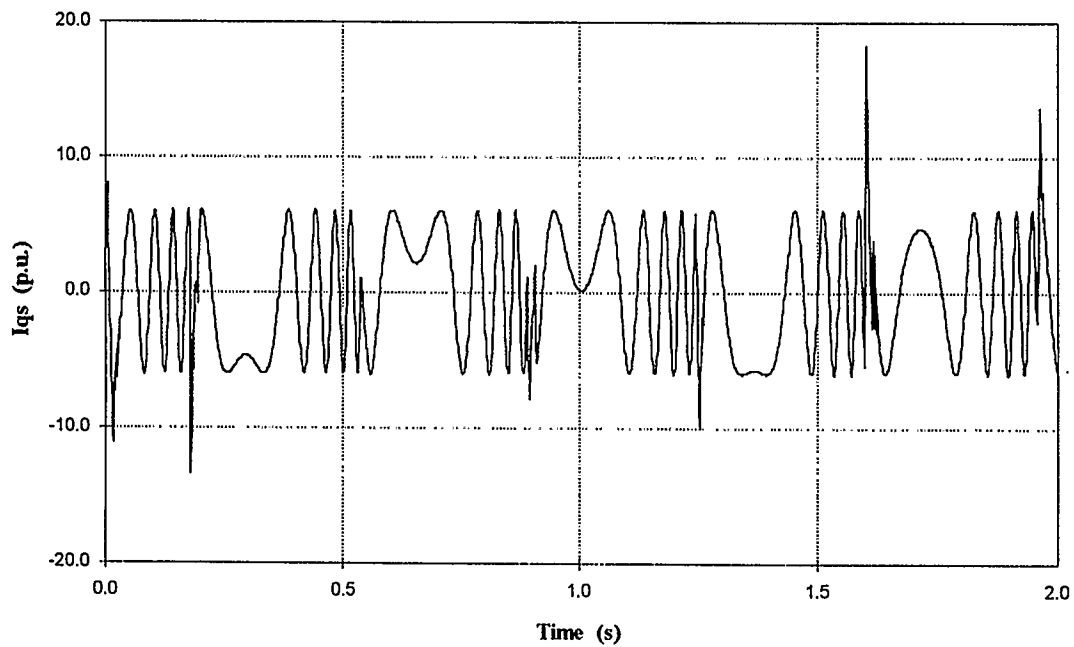


Figure 4.9 I_{qs} in stationary ref. frame with field oriented control implemented.

As shown in this chapter, the dynamic performance of the drive system is greatly improved with the introduction of field oriented control (c.f. Appendix A).

Chapter 5

Training the Neural Network

5.1 Introduction

Before the proposed neural network can be used as a flux estimator in the field oriented control of an induction motor, it first must be trained. In essence, the neural network must learn the correct mapping from input to output under a wide range of operating conditions. The training process can be broken down into these five steps :

- 1) Generate data
- 2) Normalize data
- 3) Select the neural network architecture
- 4) Train the network
- 5) Test the network

Steps (1)-(4) are discussed in this chapter as applied to one particular neural network used to generate $\sin(\phi)$, $\cos(\phi)$ and Ψ_r . Step (5) is discussed in Chapter 6.

5.2 Generating data

There are essentially two types of input data namely, the input *training* data set and input *test* data set. The input training data set is the data set that the neural network learns by, and the test data set is the data set by which the neural net is judged. The input training data is an important part of the training (learning) process because the data set is the only information it will see in order to learn its task for a given set of inputs. It also has

to encompass the entire operating range (space) in order for the neural network to properly estimate the desired output. The neural network will not give good estimates for unknown input data that lies outside of the input data training set, as it is not good at extrapolating outside of the operating space.

In addition, historical input training data is also very important during training. This is not always the case, as it only applies to input data that are periodic (oscillatory). An example would be a sinusoid where at different time instances the sinusoidal output returns the same magnitude, posing a problem for the neural network to differentiate between points on the curve.

The input vector of the neural network consists of the d - q axis stator currents, i_{ds} and i_{qs} , and their delayed values $i_{ds}(t - 1)$, $i_{ds}(t - 2)$, ..., $i_{ds}(t - 5)$, $i_{qs}(t - 1)$, $i_{qs}(t - 2)$, ..., $i_{qs}(t - 5)$, and the delayed values of $\sin(\phi)$, $\cos(\phi)$ and flux magnitude Ψ_r , $\sin(\phi - 1)$, $\sin(\phi - 2)$, ..., $\sin(\phi - 4)$, $\cos(\phi - 1)$, $\cos(\phi - 2)$, ..., $\cos(\phi - 4)$, $\Psi_r(t - 1)$, $\Psi_r(t - 2)$, ..., $\Psi_r(t - 4)$. Where one unit delay equals the sampling period which we have set to 1 ms. The output (target) vector is $\sin(\phi)$, $\cos(\phi)$ and Ψ_r . The number of inputs to the neural network is determined by the number of delays (historic data). For the network studied in this and the next chapter, there are 24 inputs and 3 outputs. Note that in the proposed neural net, delays have been set to a maximum of five unit delays. It has been observed that higher values of delays did not improve the accuracy of the estimation of the desired values significantly. The number of hidden layers and the neurons in each hidden layer are subjected to the complexity of the mapping, computer memory and computation time. For the present neural network, 20 neurons are used for the first hidden layer, 15 neurons for

the second hidden layer, and three output neurons. The training data sets for $\sin(\phi)$ and Ψ_r are shown in Figs. 5.1 and 5.2, respectively ($\cos(\phi)$ is similar in nature to $\sin(\phi)$, hence not shown). The data sets have been generated using our FOC program listed in appendix B. A total of 8,000 samples for each training data set covering load torque values of 0.0, 0.5, 1.0 and 1.5 p.u. is presented to the neural network for training. The training sequence starts with a load torque of 0.0 p.u., and a rotor speed of 0.0 p.u. The speed ramps up linearly to 0.5 p.u. (it takes approximately 200 samples (0.2 s) to do this). The speed then ramps down linearly from 0.5 p.u. speed to -0.5 p.u. (taking approximately 400 samples (0.4 s) to reach -0.5 p.u. speed). The ramp up and then ramp down action (see Fig. 4.4) is repeated for 2000 sample points (i.e. 2.0 s). The load torque is then increased to 0.5 p.u., 1.0 p.u. and 1.5 p.u. and the procedure is repeated for the remaining 6,000 points.

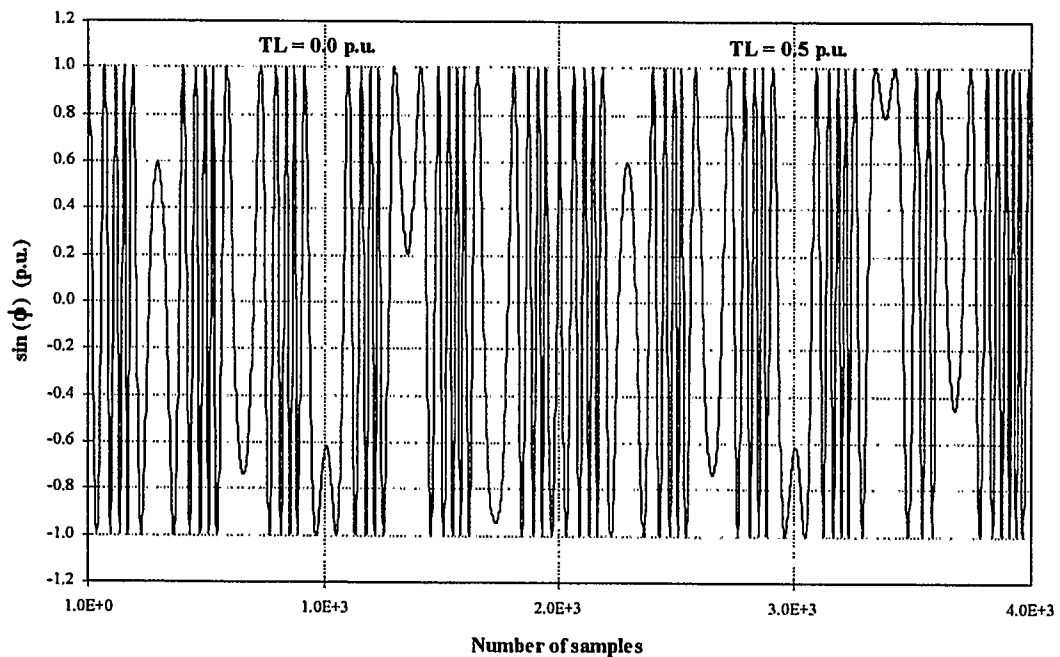


Figure 5.1(a) Training data set for $\sin(\phi)$

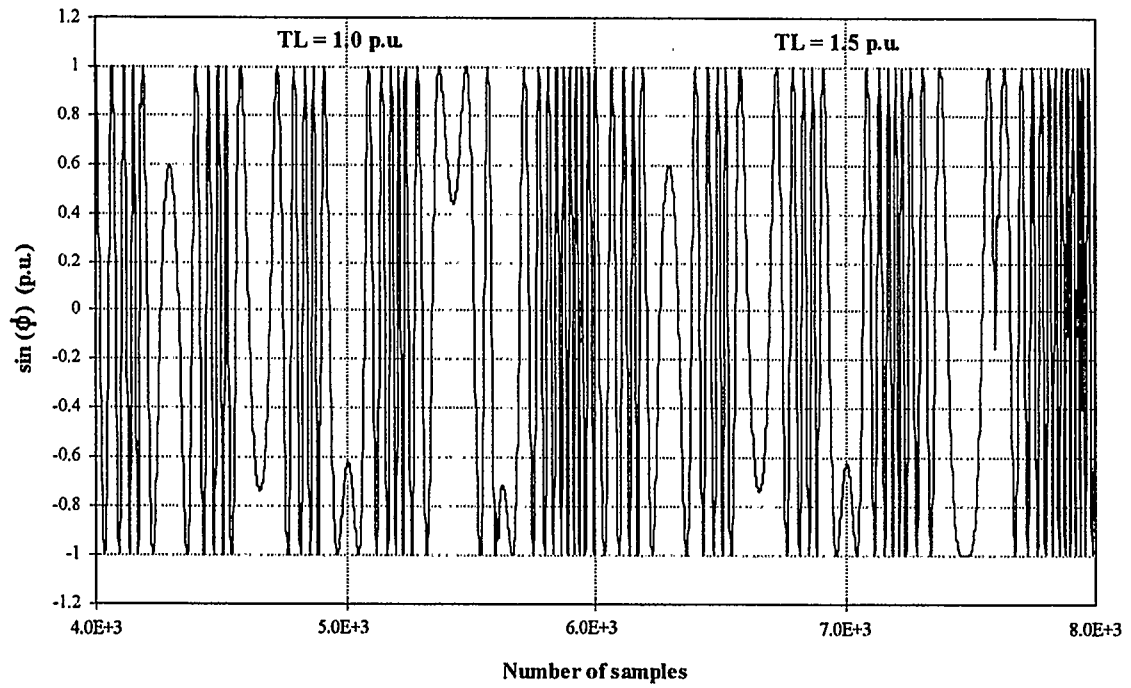


Figure 5.1(b) Training data set for $\sin(\phi)$ (continued)

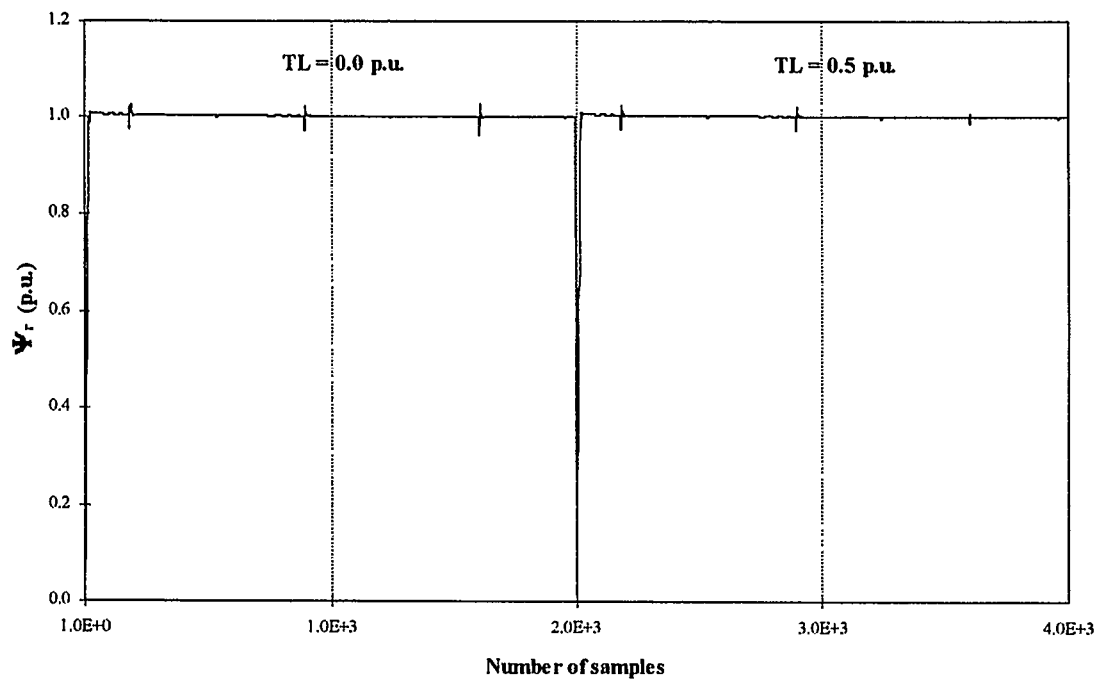


Figure 5.2(a) Training data set for flux magnitude

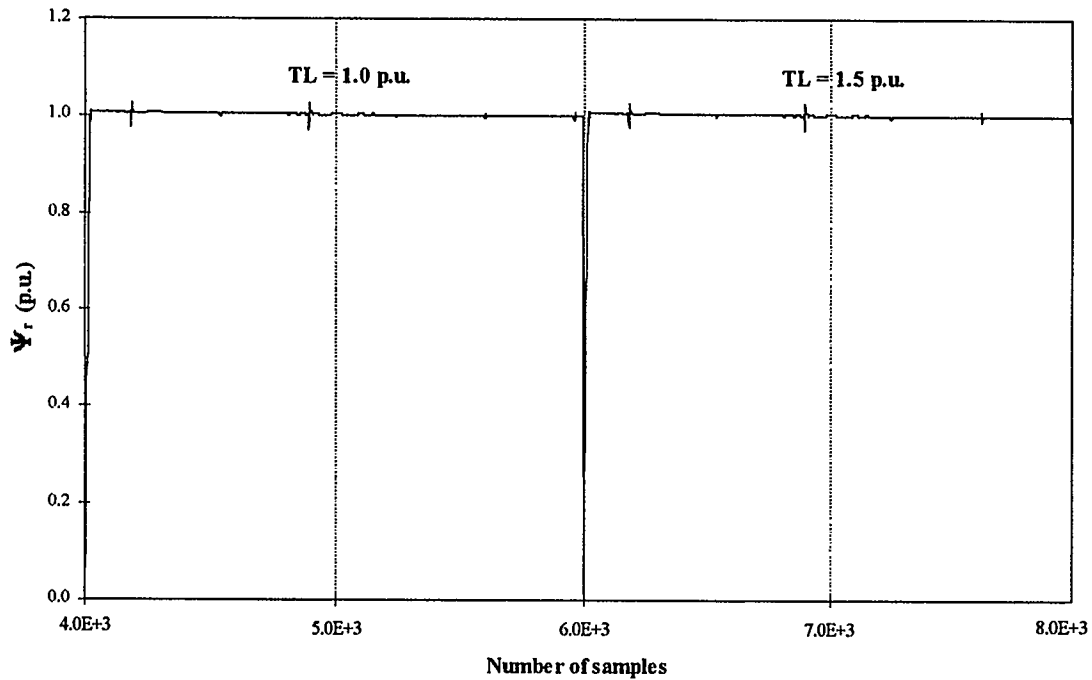


Figure 5.2(b) Training data set for flux magnitude (continued)

5.3 Normalization of Data

Once the input data are generated, the next step is to normalize all the data so that each value falls within the range from -1 to +1. This is to prevent the neurons from being driven too far into saturation. Once saturation is reached, changes in the input value result in little or no change in the output. Hence limits the performance of the neural network. The software used for training (Neural Works) will perform the required normalization automatically.

5.4 Selection of network architecture

The architecture used for the present case is a four-layer network, i.e. one input layer, two hidden layers and an output layer. The input layer simply acts as a buffer,

feeding information from the input vector through the interconnection weights to the first hidden layer. This layer (i.e. the input layer) consist of 24 neurons made up of the stationary $d-q$ currents with their delayed values, followed by the fed-back values of flux magnitude, $\sin(\phi)$ and $\cos(\phi)$ with their respective delayed values. The first hidden layer contains 20 neurons and the second 15. The choice of 20 neurons for the first hidden layer and 15 neurons for the second hidden layer are obtained through trial and error. The output layer consisting of three neurons made up of flux magnitude, $\sin(\phi)$ and $\cos(\phi)$ where they are recovered and denormalized.

5.5 Training the network

To train the network, the historical input data and output training patterns are shown to the network repeatedly until the root-mean-squared (RMS) error tolerance is met (i.e. the input data goes through the entire data set repeatedly until the RMS error is met). The algorithm used in training has been discussed in Chapter 3. In short this process involves the presentation of input data, passing it forward through the network, and back-propagating the error for each observation in the historical training set until the output values converges to a solution.

The training parameters used for the proposed ANN flux estimator are as follows: learning rate $\eta = 0.01$, momentum term $\alpha = 0.5$, the learning tolerance RMS error (termination criterion) = 0.005, maximum number of iterations = 1,000,000. Initial weights were randomly selected.

Chapter 6

Neural Network Testing and Parameter Sensitivity

6.1 Neural Network Test Results

After training the neural network with the appropriate data sets, it is time to test the network to see if indeed it is able to estimate the flux magnitude as well as the sine and cosine of the field angle ϕ . In this chapter, the neural network is put through a series of tests that involves different load torque conditions and variation in rotor resistance.

Some test results are shown in Figs. 6.1-6.6. Fig. 6.3 shows an expanded (zoomed in) view of Fig. 6.1, illustrating the high accuracy of the neural network in estimating the field angle. The other test results presented (i.e. Figs. 6.2 and 6.4-6.10) achieve similar accuracy. The neural network output plots are invariably always plotted coincident with the test data (hence hard to see), because the network can estimate the desired result so well.

The results shown in Figs. 6.1-6.4 are for test data generated with a constant load torque of 0.25 p.u., which is not part of the training data set, although, it is still within the scope of the training data range (recall from the previous chapter that the neural network was trained for torque load conditions of 0.0, 0.5, 1.0 and 1.5 p.u.). The test results for a constant load torque of 0.75 p.u. (which is also different from the training data set but still within the training range) are similar to that of the 0.25 p.u. load torque case (not shown here). Results shown in Figs. 6.5 and 6.6 are for test data that are outside the range of the training data, i.e. the torque load is 2.0 p.u. Note in these two figures that the ANN flux

estimator (somewhat surprisingly) estimates very accurately the field angle and flux magnitude despite a load condition outside the training data set.

For Figs. 6.5 onwards, $\cos(\phi)$ is not plotted since it is similar in nature to $\sin(\phi)$. In addition the range of these plots (i.e. Figs. 6.5-6.14.) have been expanded to aid in distinguishing the neural network output from the test data.

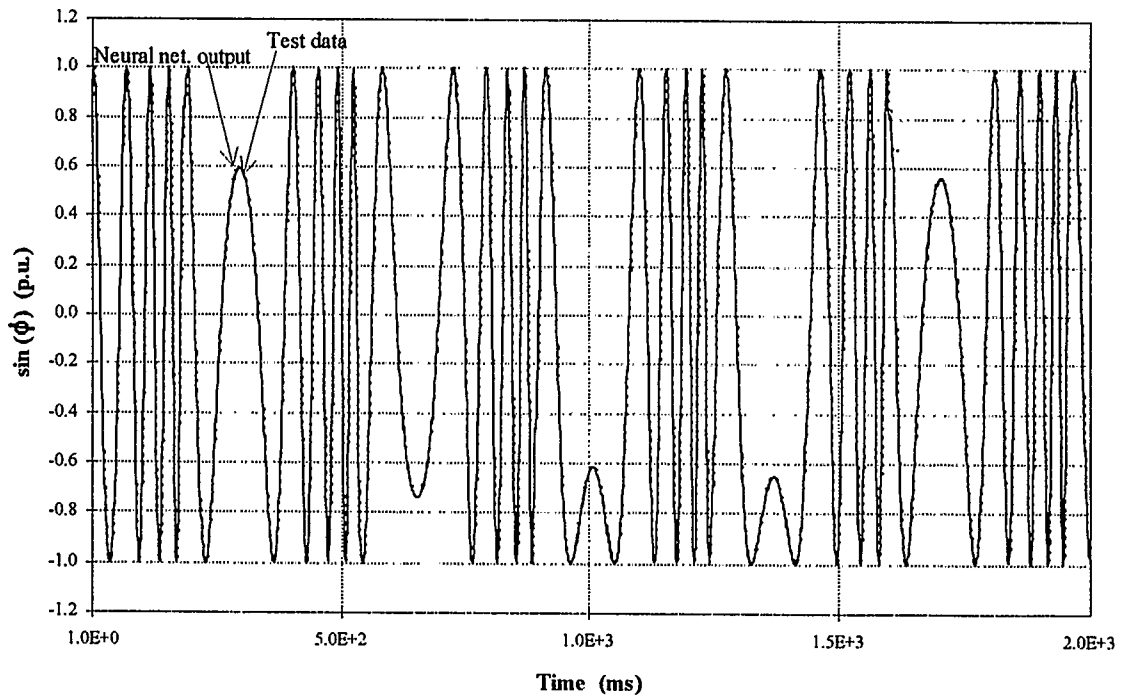


Figure 6.1 Neural network output for $\sin(\phi)$ vs test data for TL=0.25 p.u.

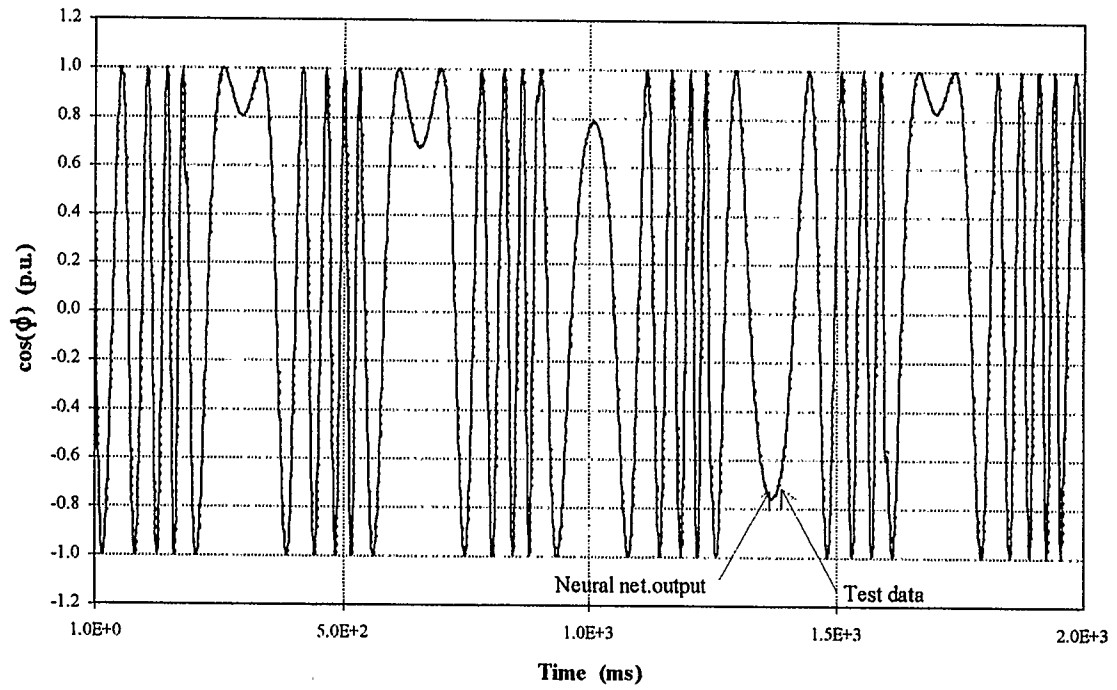


Figure 6.2 Neural network output for $\cos(\phi)$ vs test data for $TL=0.25$ p.u.

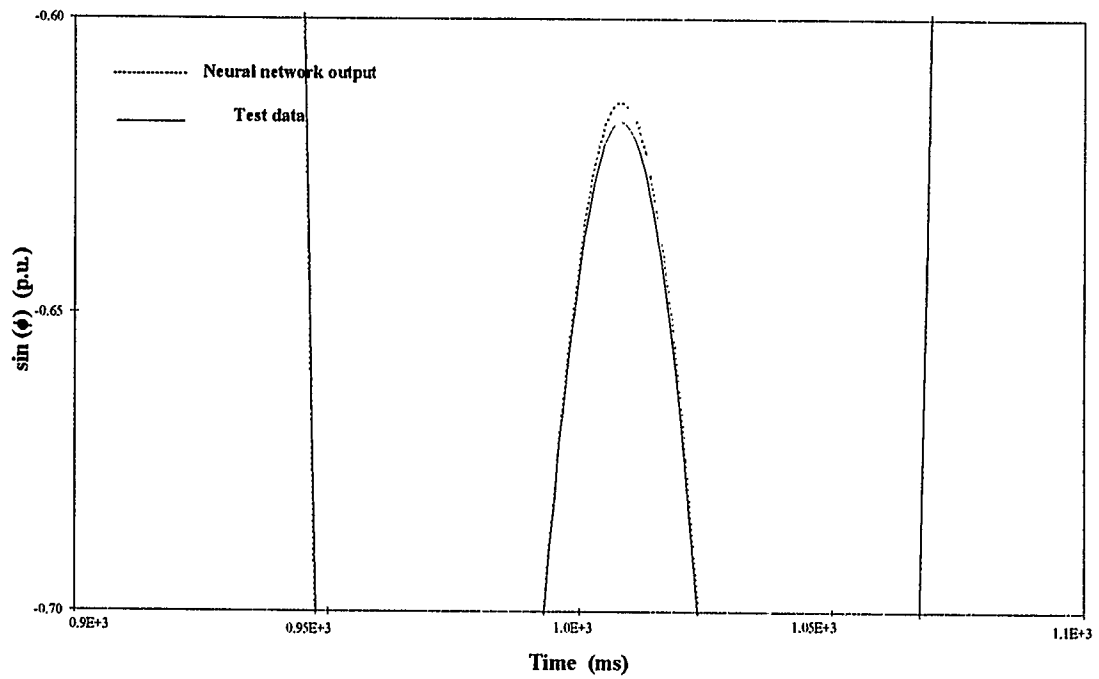


Figure 6.3 Neural network output for $\sin(\phi)$ vs test data for $TL = 0.25$ p.u.
(expanded view)

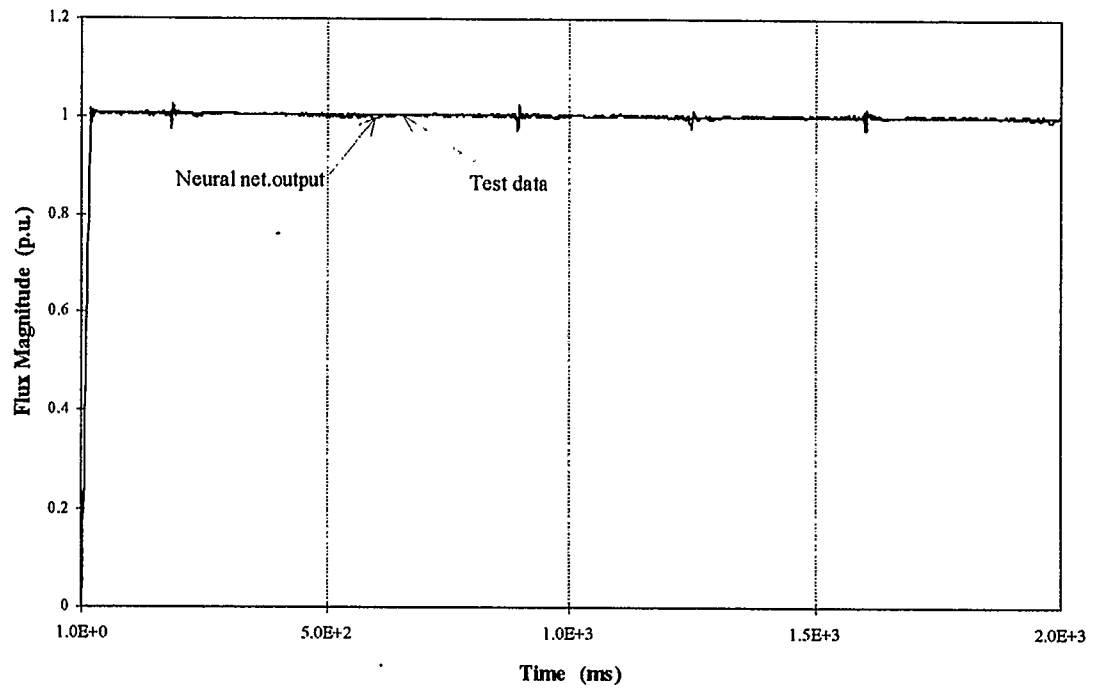


Figure 6.4 Neural network output for flux magnitude vs test data for TL = 0.25 p.u.

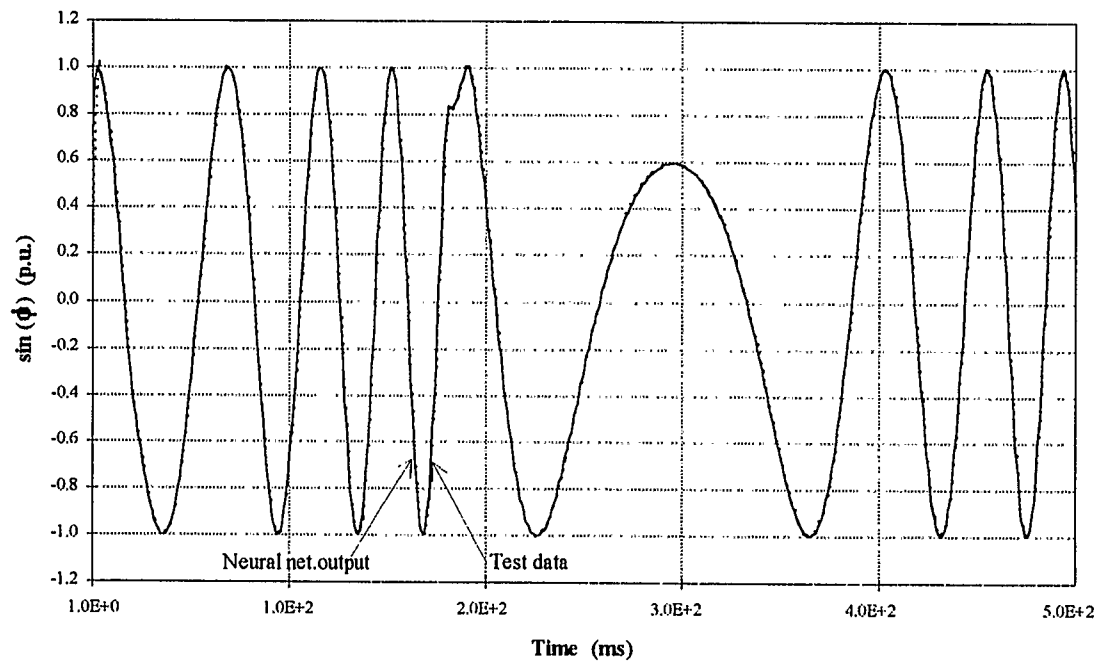


Figure 6.5 Neural network output for $\sin(\phi)$ vs test data for TL = 2.0 p.u. (outside training range)

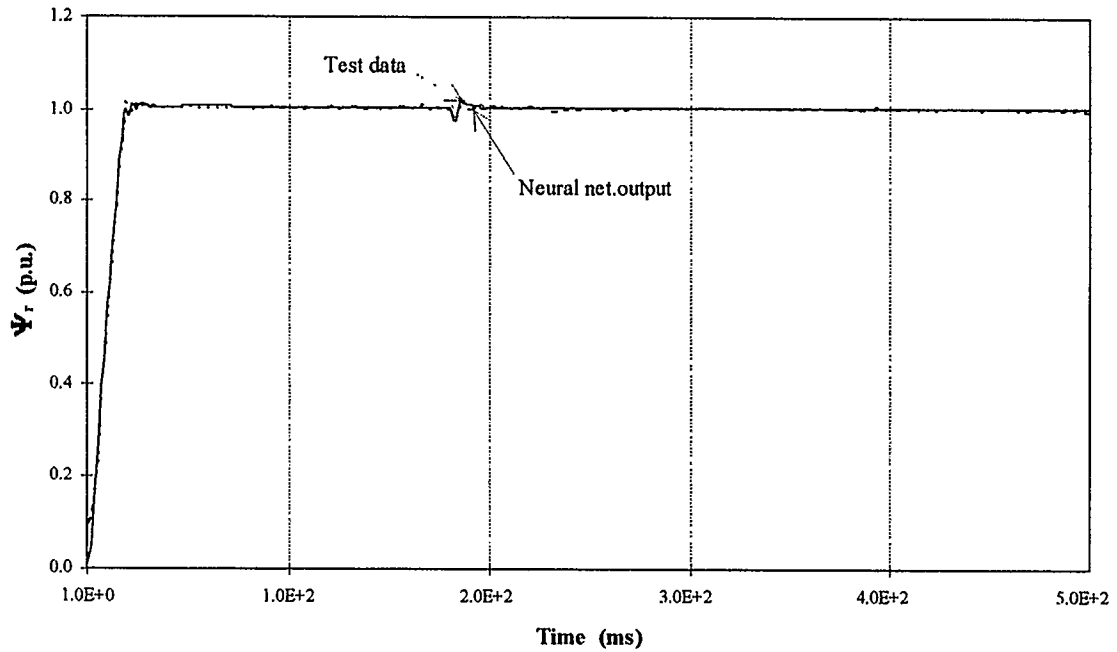


Figure 6.6 Neural network output for flux magnitude vs test data for TL=2.0 p.u. (outside training range).

6.2 Parameter Sensitivity

Note that estimation of the rotor flux magnitude and the field angle depends on the value of machine parameters (naturally the network must be trained for a particular machine). The rotor resistance variation, especially, becomes dominant due to temperature variation and the skin effect.

The following results verify that the neural network is capable of achieving accurate results even with rotor resistance variations present. The rotor resistance was increased 50% and 100% (i.e. $R_r=1.5$ p.u. and $R_r=2.0$ p.u.) above the nominal value and the results of $\sin(\phi)$ and flux magnitude are shown in Figs. 6.7-6.10, respectively. The results of $\cos(\phi)$ are similar to that of $\sin(\phi)$ and are not shown here. In achieving these results, the flux PI controller had to be adjusted to give the desired results. To achieve

these results in practice, some type of adaptive control would be used with or in place of the PI controllers.

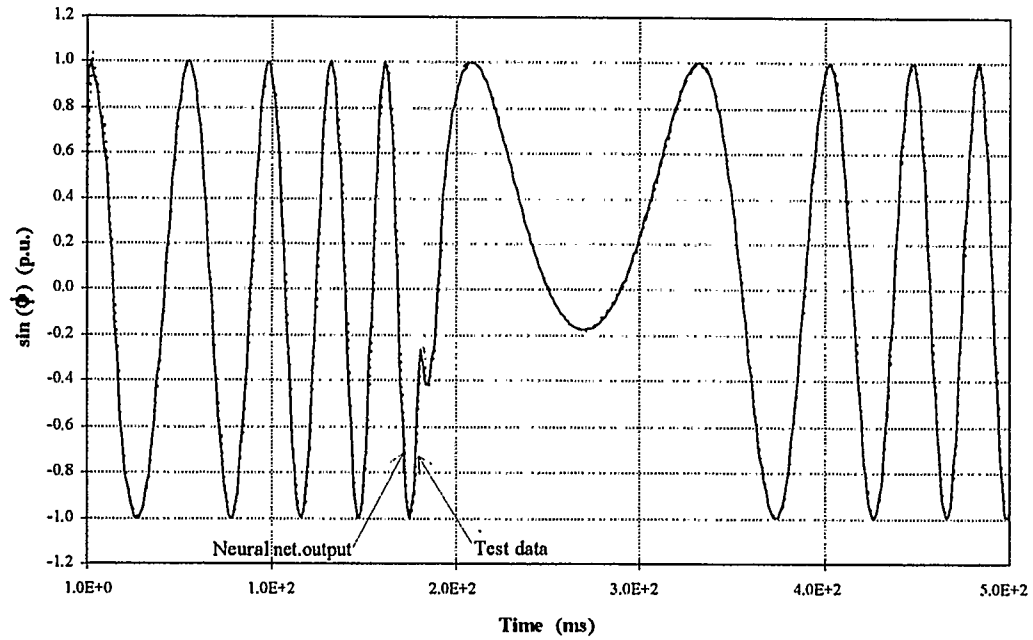


Figure 6.7 Neural network output for $\sin(\phi)$ vs test data for 50% increase in rotor resistance (PI controller adjusted).

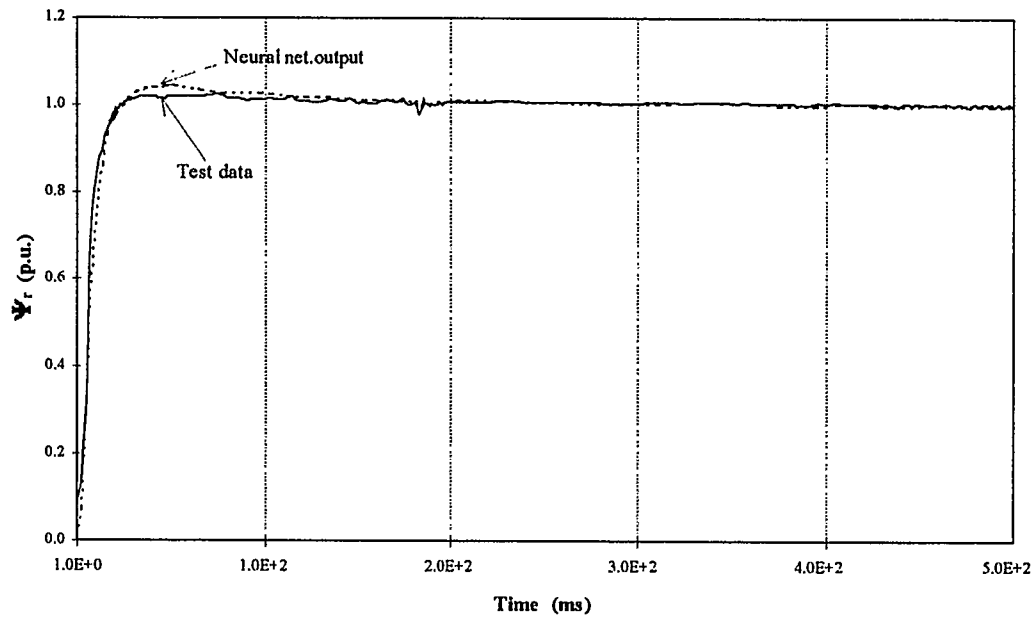


Figure 6.8 Neural network output for flux magnitude vs test data for 50% increase in rotor resistance (PI controller adjusted).

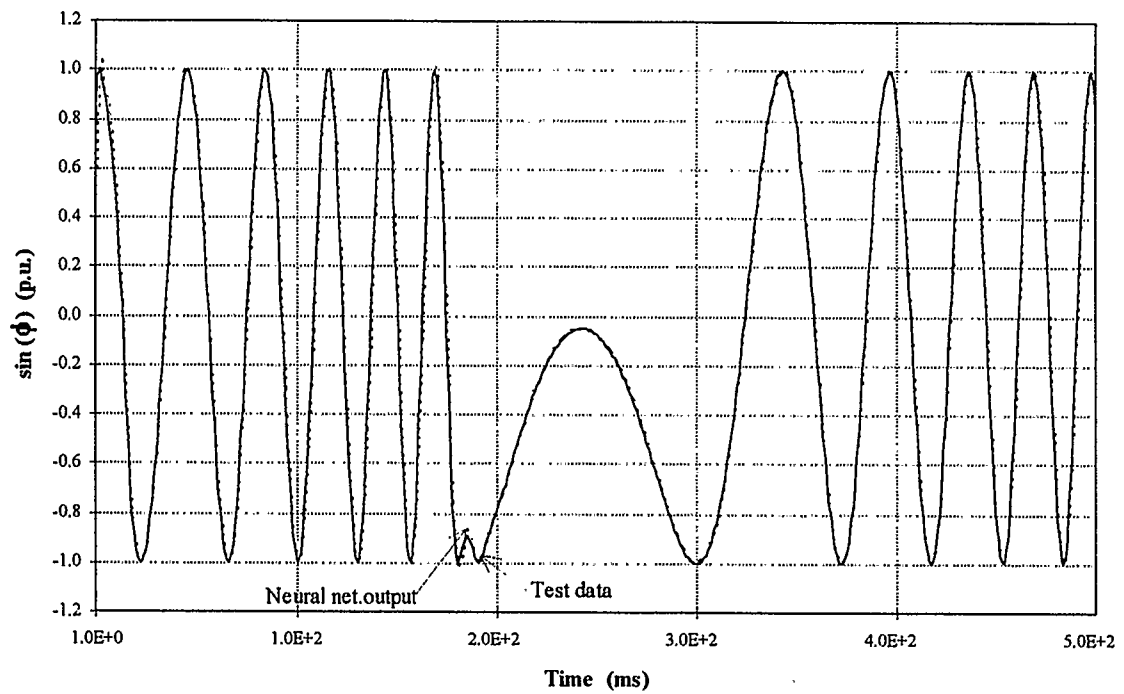


Figure 6.9. Neural network output for $\sin(\phi)$ vs test data for 100% increase in rotor resistance (PI controller adjusted).

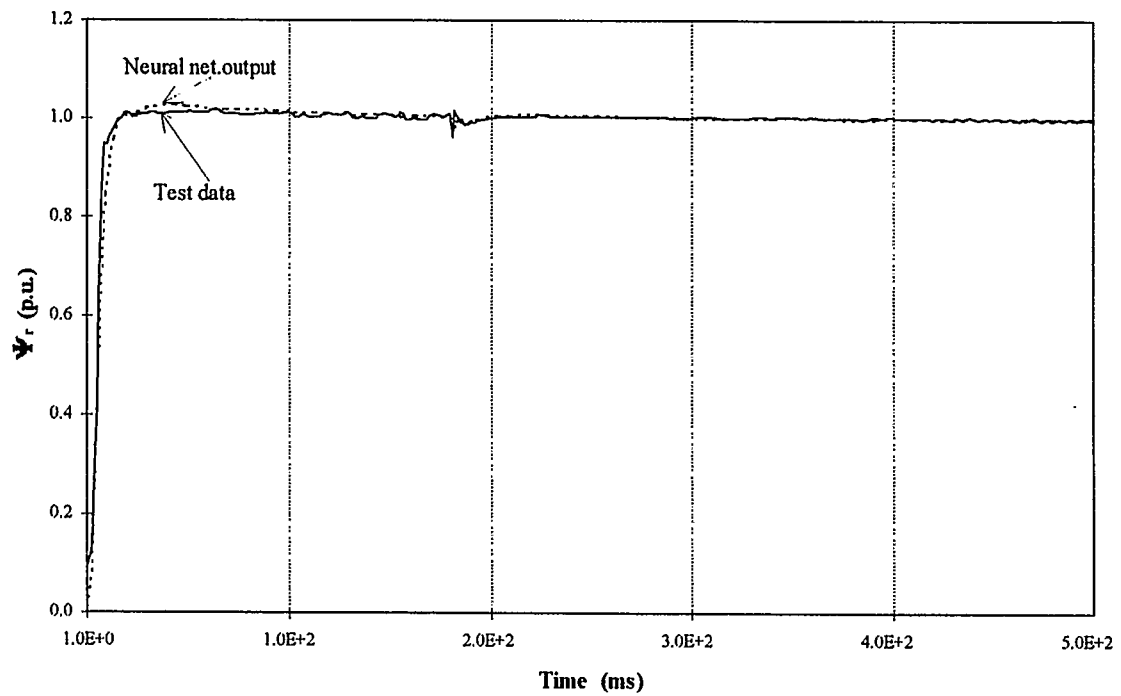


Figure 6.10 Neural network output for flux magnitude vs test data for 100% increase in rotor resistance (PI controller adjusted).

As can be seen from Figs.6.7-6.10, the neural network is able to accurately track the test data even with a 100% rotor resistance variation. The maximum absolute error is about 0.10 p.u. for each case.

It is worthwhile to determine under what conditions the neural network flux estimator will not perform well. The following tests are done with fixed PI controller coefficient values, optimized for the nominal rotor resistance (i.e. $R_r = 1.0$ p.u.). The rotor resistance is then changed by 20% and 30% (PI controller coefficients unchanged). It is observed in Figs. 6.11 and 6.12 that during the flux transient, the neural network is still able to track the test data for a rotor resistance variation of 20%. However, as seen in Fig. 6.12 the neural network is able to track the steady-state test data (checked to 2.0 s of operation) but with an oscillatory error (at about the synchronous frequency)

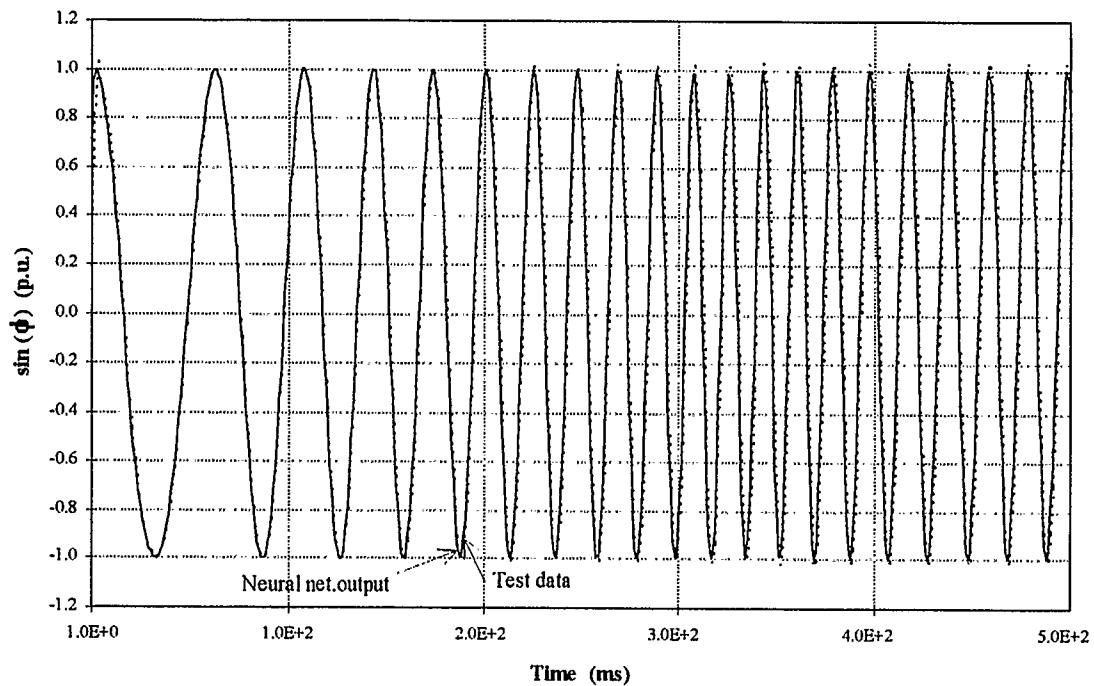


Figure 6.11 Neural network output for $\sin(\phi)$ vs test data for 20% increase in rotor resistance.

of approximately 0.02 p.u. peak absolute error. As seen in Figs. 6.13 and 6.14, for a 30% increase in rotor resistance, the PI controllers with fixed coefficients can no longer regulate the flux magnitude. As seen, despite this very abnormal operation, the neural network can still estimate the flux magnitude with a 0.2 p.u. maximum absolute error.

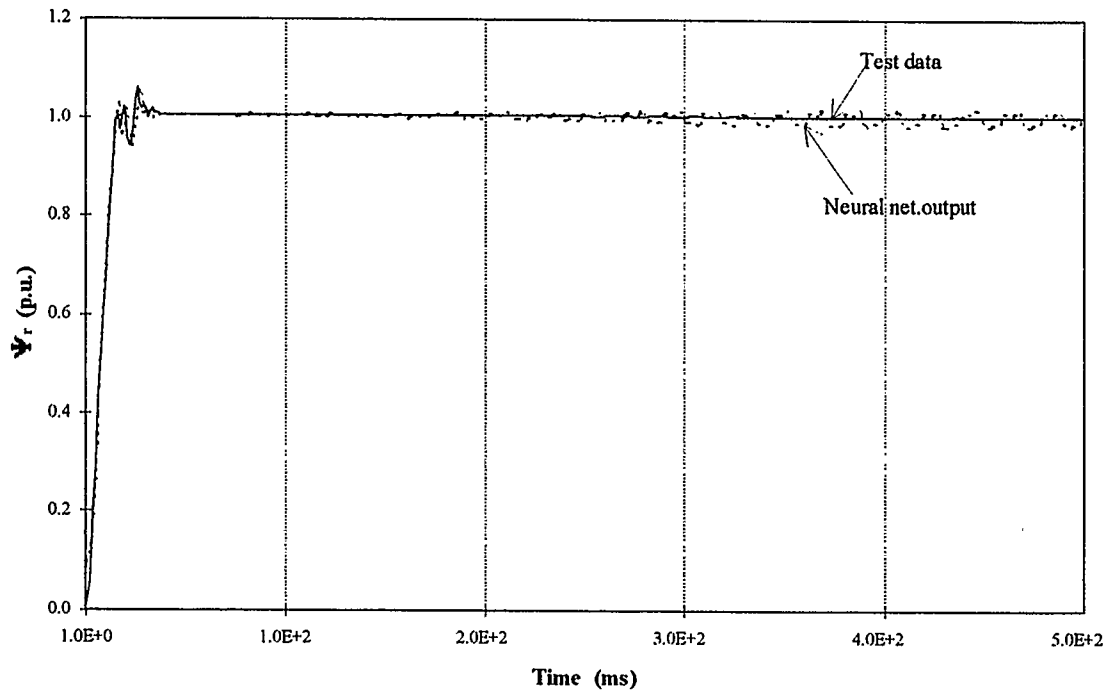


Figure 6.12 Neural network output for flux magnitude vs test data for 20% increase in rotor resistance.

Note that we have not yet placed the flux estimator into the field oriented control feedback loop. We have only applied test data generated with exact calculations of the flux magnitude and the field angle (c.f. Chapter 4). So behavior of the field oriented controller with fixed PI coefficient values and neural network flux estimation is unknown at this time for large variations in rotor resistance (i.e. greater than 20%). Nonetheless, we have shown that the robustness (i.e. load condition and parameter insensitivity) of the neural network is very good, even though the neural network was trained for a fixed value

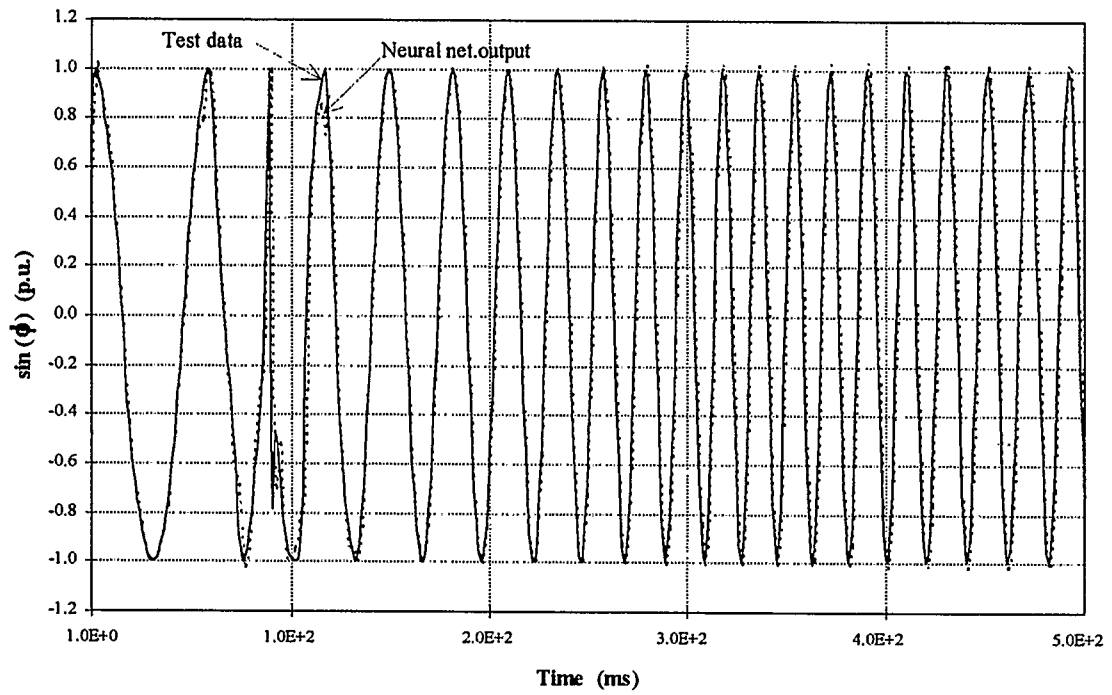


Figure 6.13 Neural network output for $\sin(\phi)$ vs test data for 30% increase in rotor resistance.

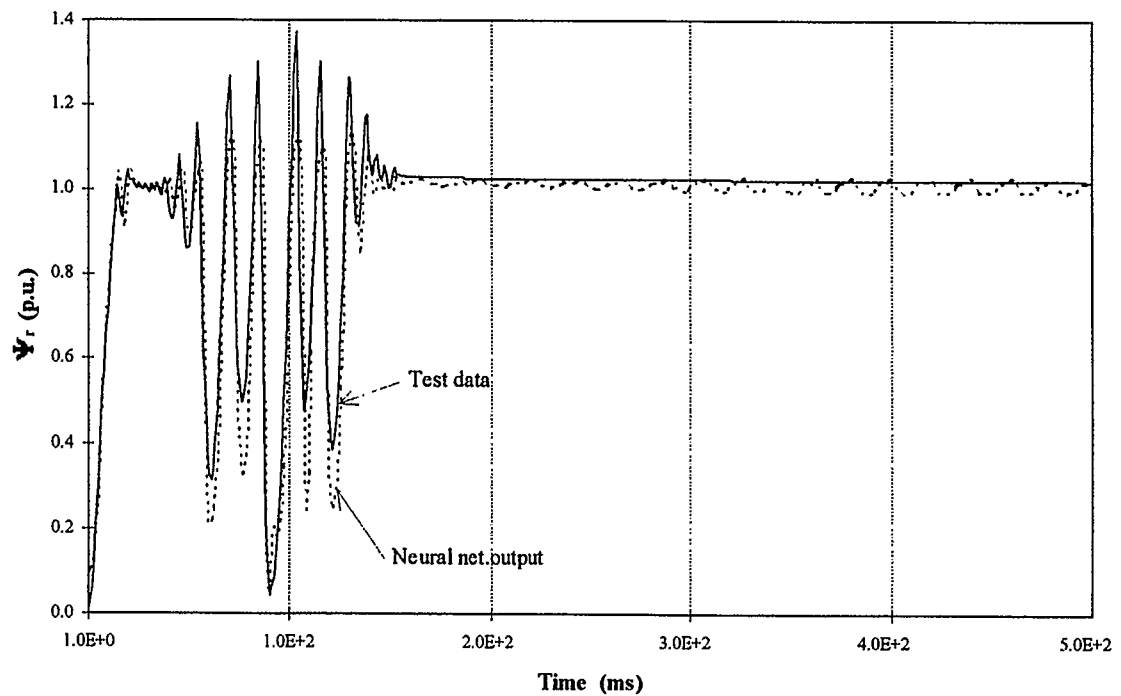


Figure 6.14 Neural network output for flux magnitude vs test data for 30% increase in rotor resistance.

of rotor resistance. The ability for the neural network to generalize the given input data to generate desired outputs is indeed one of its strengths. Further, the results presented in this chapter indicates that the artificial neural network flux estimator should work very well under all normal conditions, provided that the controller has some adaptive ability for rotor resistance and other parameter variations.

Chapter 7

Discussion

7.1 Effects of Non-Historic Data Input

Employing a training data set without historic (past/delayed) values has an adverse effect on the outcome of the neural network's ability to generalize data. As can be seen in Fig. 7.1, the neural network has great difficulty in tracking the desired flux magnitude without the use of historic inputs. The more historic data the better the neural network will perform, although, there are limitations to how many historic (delayed) steps the neural network requires until no significant improvements are observable. For the network studied in Chapters 5 and 6, delays ranging from one to five sample periods proved to be optimum.

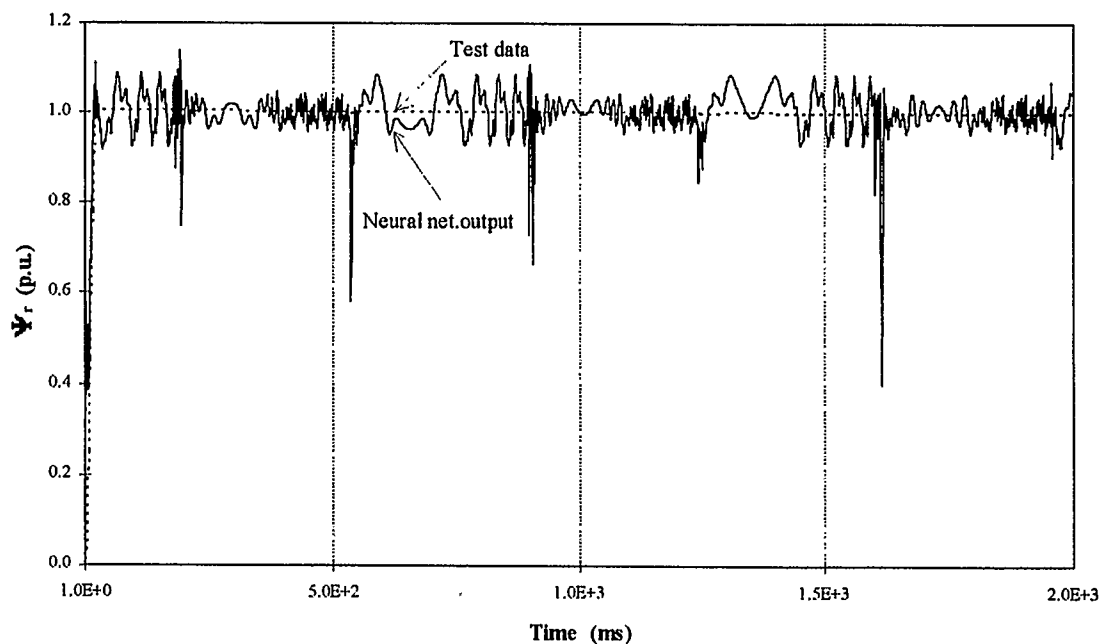


Figure 7.1 Neural network output for flux magnitude vs test data without historic data.

7.2 Alternative Inputs to the Neural Network

In this thesis, the inputs to the neural network come in the form of stator d^s - q^s currents (i.e. the motor phase currents i_a , i_b and i_c are just transformed into the stationary frame i_{ds}^s and i_{qs}^s form prior to application to the neural network). Alternatively, one might wish to apply the three-phase motor currents i_a , i_b and i_c directly to the neural network. When using the three-phase currents as inputs (with historic data), the neural network is still able to estimate the flux magnitude and field angle accurately, however the results are slightly less accurate than those for which i_{ds}^s and i_{qs}^s are inputs [29]. One explanation for this is that the “information” (i.e. changes in the operation of the induction motor) reflected in the phase currents i_a , i_b and i_c is not as “detailed/accurate” (i.e. not able to identify subtle changes in the operation of the induction motor) as is contained in the stator d^s - q^s currents i_{ds}^s and i_{qs}^s .

7.3 Multiple Networks versus Single Network

A single neural network has been used for all three outputs ($\sin(\phi)$, $\cos(\phi)$ and flux magnitude Ψ_r). Although the nature of the outputs $\sin(\phi)$ and $\cos(\phi)$ (Figs. 4.6 and 4.7) are similar in nature, they are quite different from the nature of the flux magnitude Ψ_r output (Fig. 4.5). Nonetheless, the network is still able to estimate these outputs without difficulty. However, when a separate network is used to estimate $\sin(\phi)$ and $\cos(\phi)$ and another network is used to estimate flux magnitude Ψ_r , the accuracy of the neural network flux estimator for each case is improved. The accuracy improves only slightly when compared to the single network case, hence, this does not warrant the use of two

separate networks which would require more resources and computational power to perform the work that a single network can do.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

For this thesis, the work of implementing field oriented control (FOC) for an induction motor using an artificial neural network flux estimator is split up basically into two parts, the field orientation part and the artificial neural network part.

Field oriented control requires complex matrix transformations to perform coordinate transformations from ac quantities of the induction motor to dc quantities of the d - q model used by the controller, and vice versa. These transformation are laid out in Chapter 2. The proposed FOC drive system is discussed in Chapter 4. This drive system is intended for implementation with an existing pulse width modulator (PWM) voltage source inverter. Its purpose in this thesis is simply the generation of training and test data for the artificial neural network flux estimator.

Artificial neural network theory is discussed in Chapter 3. The proposed artificial neural network flux estimator, studied in Chapters 5 and 6, contains four layers : one input layer (24 neurons), two hidden layers (25 neurons for the first hidden layer and 15 neurons for the second hidden layer) and one output layer containing 3 output neurons (namely $\sin(\phi)$, $\cos(\phi)$ and flux magnitude Ψ_r). The inputs to the flux estimator are the d - q stator currents. The neural network's ability to accurately estimate the flux magnitude and the field angle under a variety of load and parameter variation conditions, has been verified by the test cases presented in Chapter 6.

The advantages that the neural network has over conventional methods of estimating flux are :

- It has fast processing speed
- It is robust (i.e. fault tolerant)
- It is adaptive (i.e. extrapolates well)
- It is an alternative technique to mathematical programming
- It requires less memory

The primary disadvantage of the neural network flux estimator is the (potentially) long training process involving a large degree of trial and error investigation.

8.2 Future Work

Some suggestions for future work are :

- 1) the design of an adaptive controller to compensate for rotor resistance variations
- 2) the incorporation of motor non-linearities in the induction motor model
- 3) the physical implementation of the ANN flux estimator
- 4) the study of the ANN flux estimator with other inputs (e.g. stator voltages and rotor speed)

Reference

- [1] B. K. Bose, "Power electronics and AC drives," *Prentice Hall*, 1986.
- [2] J. Zhang, "Field oriented control of induction motor speed," *Msc. Thesis*, Dept. of Electrical Engg., The University of Calgary, Calgary, AB., Canada, August 1985.
- [3] F. Blaschke, "A new method for the structure decoupling of AC induction machines," *Second IFAC Symposium on Multi-Variable Technical Control Systems*, Pt 3, pp.11-13, Oct. 1971.
- [4] K. Hasse, "Zum Dynamischen Verhalten der Asynchronmaschine bei Betrieb mit variabler Staenderfrequenz und Staender-spannung" ("On the dynamic behavior of induction machines driven by variable frequency and voltage sources"), *ETZ-A Bd 89 H.4*, pp.77-81, 1968.
- [5] R. Gabriel, W. Leonhard and C. Nordby, "Field oriented control of a standard AC-motor using microprocessors," *IEEE Trans. Ind. Appl.*, Vol. IA-16, No. 2, pp. 186-192, Mar./Apr., 1980.
- [6] H. Nakano, H. Akagi, I. Takahash and A. Nabae, "A new equivalent circuit of induction motor based on the total linkage flux of the secondary windings," *Electrical Engg. in Japan*, Vol. 103, No. 2, pp. 68-73, 1983.
- [7] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, Apr. 1987.
- [8] R. Rosenblatt, "Principles of neurodynamics," *New York, Spartan Books*, 1959.
- [9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in D.E. Rumelhart & J.L. McClelland (Eds.), "*Parallel*

- distributed processing: explorations in the microstructure of cognition*," Vol. 1: Foundations. MIT Press, 1986.
- [10] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, Vol. 79, 2554-2558, Apr. 1982.
- [11] T. Kohonen, "Self organization and associative memory," *Springer-Verlag*, Berlin, 1984.
- [12] A. Maren, C. Harston, R. Pap, "Handbook of neural computing applications," *Academic Press Inc.*, San Diego, CA, 1990.
- [13] T. H. Barton, T. Grant, V. Thiagarajan and J. Zhang, "The field oriented control of induction motor drives," *Research Work Report*, Dept. of Electrical Engg., The University of Calgary, Calgary, Alberta, Aug. 1984.
- [14] R. Gabriel and W. Leonhard, "Microprocessor control of induction motor," *Int. Semiconductor Power Converter Conf.*, pp. 385, 1982
- [15] P. L. Jansen, R. D. Lorenz, "A physically insightful approach to the design and accuracy assessment of flux observers for field oriented induction machine drives," *IEEE Trans. Ind. Appl.*, Vol. 30, No. 1, pp. 101-110, Jan/Feb. 1994
- [16] Neural Inc, "Neural computing," *Neural Ware*, 1993.
- [17] P. Werbos, "Beyond regression: new tools for prediction and analysis in the behavioral sciences," *PhD thesis*, Harvard, Cambridge, MA, Aug. 1974.
- [18] D. B. Parker, "Learning logic," *Technical Report TR-47*, Centre for Computational Research in Economics and Management Science, MIT, Cambridge, MA, Apr. 1985

- [19] J. A. Freeman, D. M. Skapura, "Neural Network : algorithms, applications and programming techniques," Reading, Mass., *Addison-Wesley*, 1991.
- [20] S. Becker, Y. le Cun, "Improving the convergence of back-propagation learning with second-order methods," *Technical Report CRG-TR-88-5*, U. of Toronto, Toronto, Canada, 1988.
- [21] E. D. Dahl, "Accelerated learning using the generalized delta rule," *In Proceedings of the IEEE 1st International Conference on Neural Networks*, Vol. 2, pp.523-530, San Diego. CA, 1987.
- [22] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, 1(4):295-308, 1988.
- [23] C. Cybenko, "Approximations by superposition of a sigmoidal function," *Math. Contr., Signal, Syst.*, Vol. 2, 1989, pp. 303-314.
- [24] E. B. Baum, D. Haussler, "What size net gives valid generalization?" *Neural Computation*, 1:151-160, 1989.
- [25] Y. le Cun, J. S. Denker, S. A. Solla, "Optimal brain damage," In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pp.168-177. Morgan Kaufmann, 1990.
- [26] S. C. Huang, Y. F. Huang, "Bounds on the number of neurons in multilayer perceptrons," *IEEE Transactions on Neural Networks*, 2(1):47-55, 1991.
- [27] D. Hush, J. M. Salas, B. Horne, "Error surface for multi-layer perceptrons," *IEEE Transaction on Systems, Man and Cybernetics*, 22(5), 1992.

- [28] J. R. Smith, A. J. Tait, "Electrical drive simulator for teaching purposes," *IEE Proceedings*, Vol. 135, Pt. A, No. 1, Jan. 1988.
- [29] A. K. P. Toh, E. P. Nowicki, F. Ashrafzadeh, "A flux estimator for field oriented control of an induction motor using artificial neural network," *Conf. Record of IEEE, IAS Annual Meeting*, 1994.

Appendix A

A.1 Induction Motor Model

The implementation of field oriented control is based on the induction motor model discussed in this appendix. The dynamic behavior of the induction machine has an important effect upon overall performance of the system of which it is a part of. In order to study the dynamic behavior of the induction machine, a computer simulation of the induction motor was developed. The computer simulation is based on [28]. The program is written in C. The induction motor simulator is capable of performing detailed analysis of different loading conditions and/or voltage disturbances. It is able to do this in the rotor reference frame, stator reference frame or the synchronously rotating reference frame.

The machine model is based on the two-axis electrical equations discussed in Chapter 2. In practical simulations, the variations of parameters due to saturation and eddy current effects will have to be taken into account. In the present case, no attempts have been made to include these parameter variations because the principle objective is to illustrate the basic operation of the machine in different loading conditions and when a fault occurs at the terminals of the machine. The numerical integration employed here is based on the Runge-Kutta algorithm. The integration step length is chosen on the basis of achieving an accurate solution during the run-up period while maintaining an economical simulation. Values of 0.001 s to 0.0025 s can be used without incurring instability, in the present case a value of 0.001 s is chosen.

A.2 Results of Load and Voltage Disturbances

The results are based on the induction motor model in the stator reference frame. The following plots are for a free acceleration of a 10 hp induction motor for line start conditions. As observed from Fig. A.1, it takes the induction motor approximately 1.0 s to reach a speed of 1.0 p.u. without field oriented control. See section A.3 for the motor parameters.

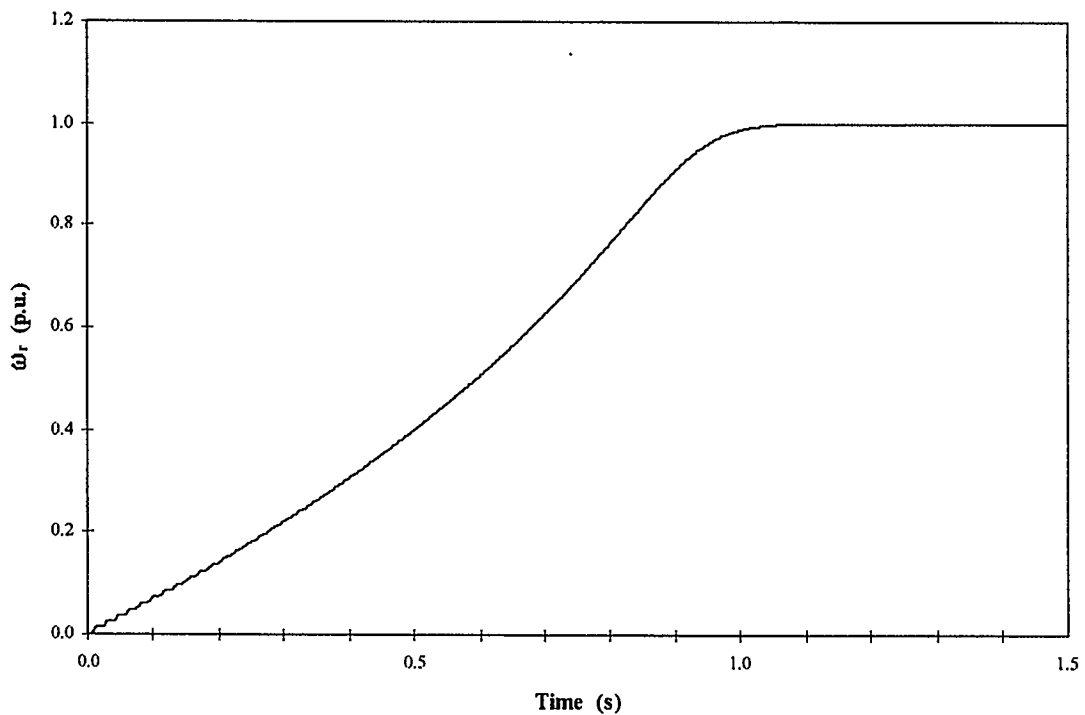


Figure A.1 Rotor speed for line start operation.

Also, it takes about 1.0 s for the induction motor to reach steady state for the case of electromagnetic torque and flux magnitude as shown in Figs. A.2 and A.3.

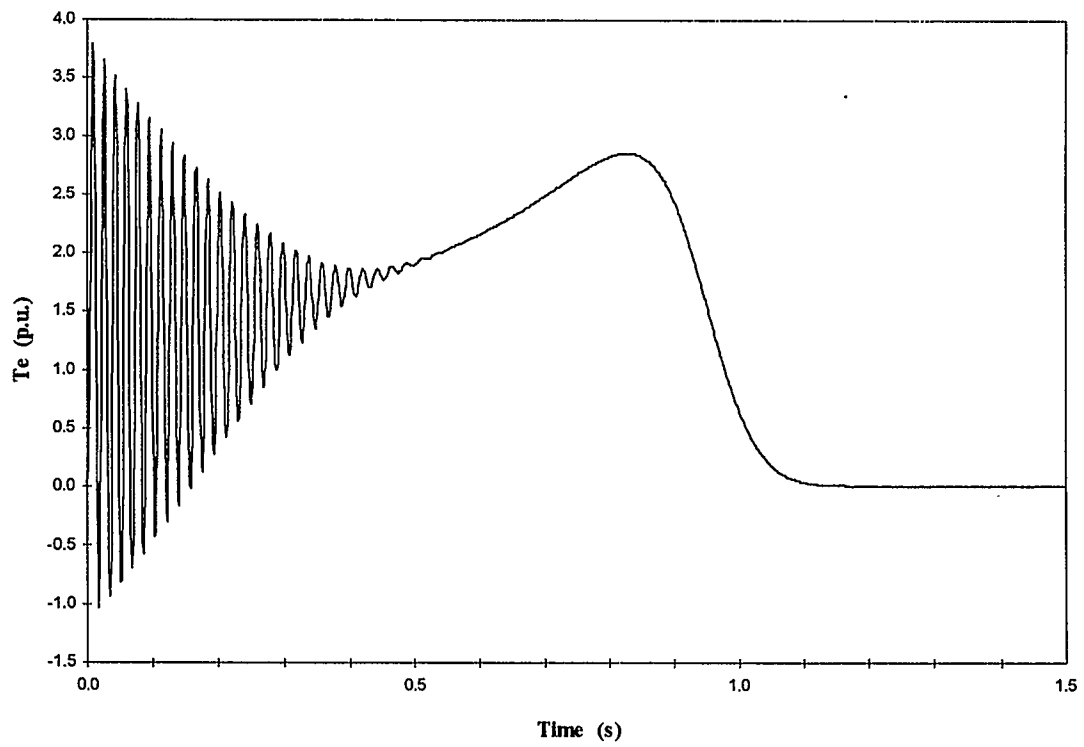


Figure A.2 Electromagnetic Torque for line start operation.

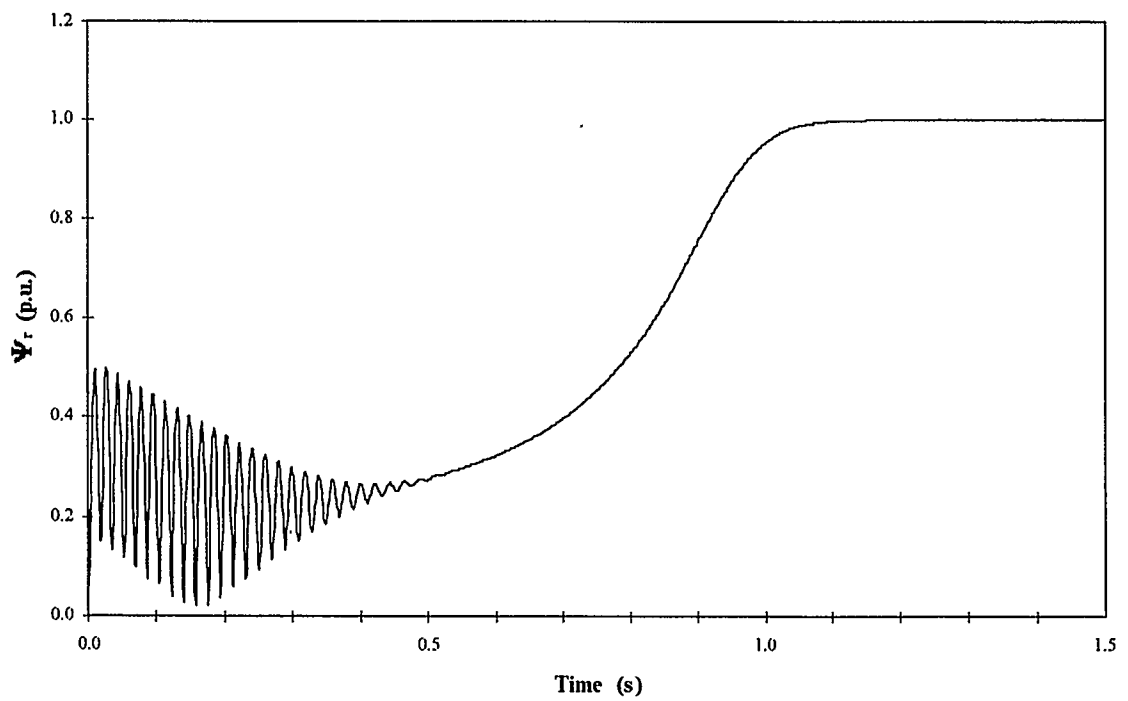


Figure A.3 Flux magnitude for line start operation.

The d - q currents in the stationary reference frame are shown as follows.

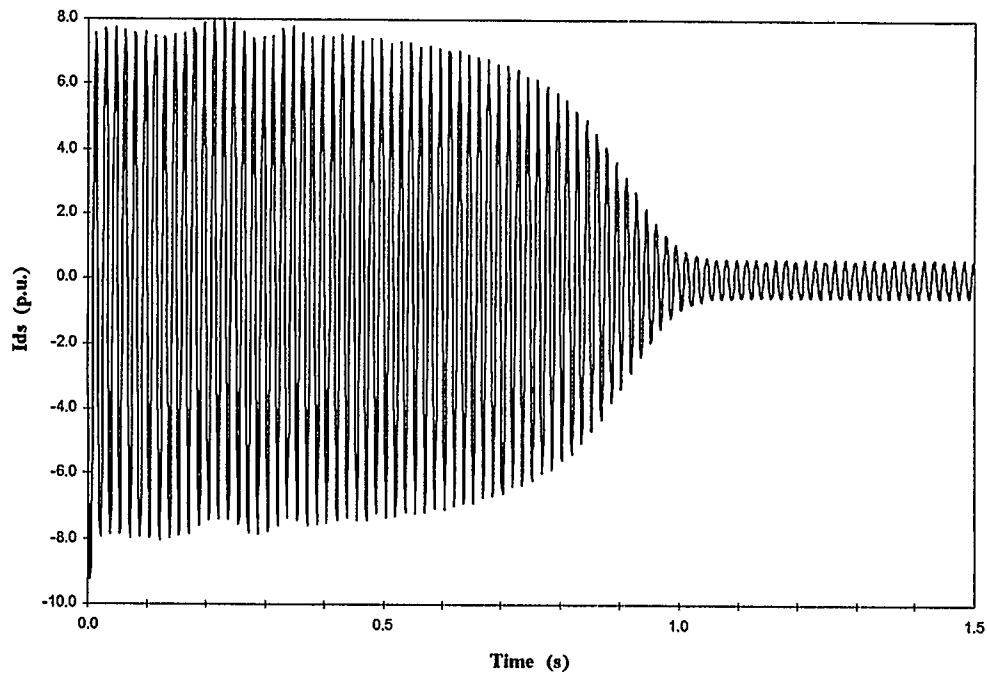


Figure A.4 I_{ds} vs time in the stationary reference frame.

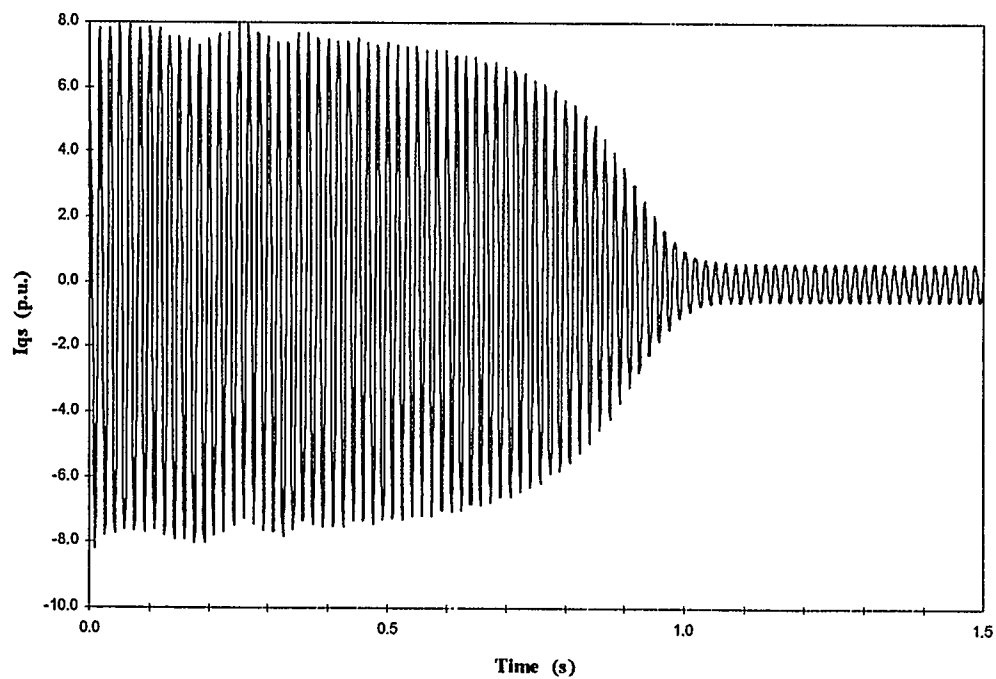


Figure A.5 I_{qs} vs time in stationary reference frame.

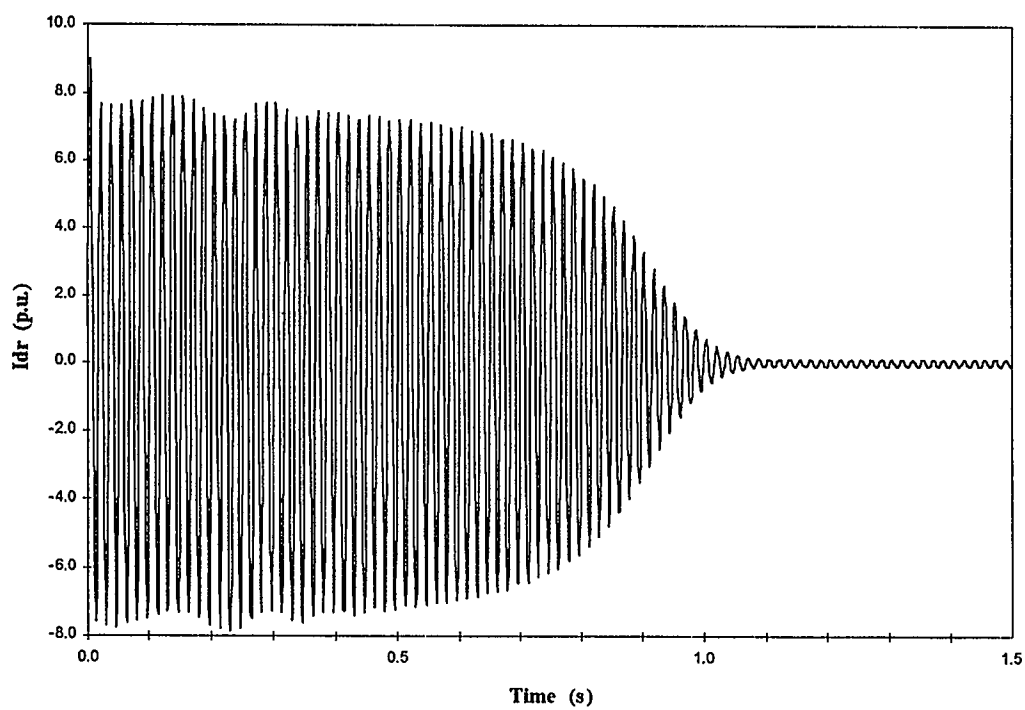


Figure A.6 I_{dr} vs time in stationary reference frame.

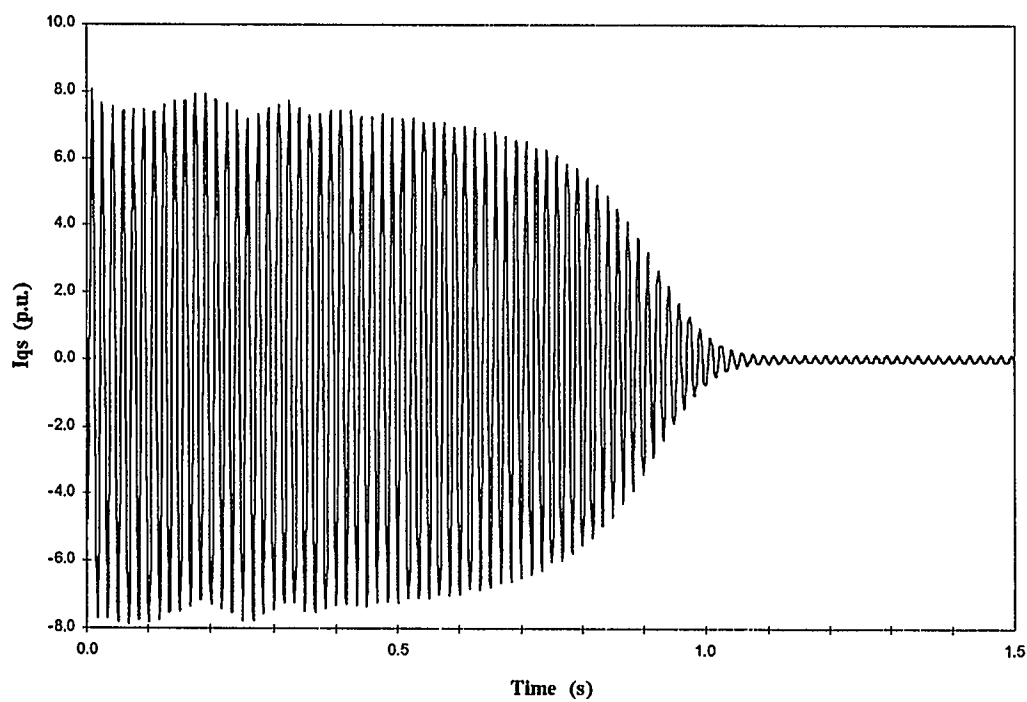


Figure A.7 I_{qr} vs time in stationary reference frame.

The following simulated sequence is performed on a 30 hp induction motor (see section A.3 for the motor parameters). The simulated sequence of normal and abnormal operation is given firstly by the free acceleration of the motor from a direct on-line start, followed by a load application of 1.0 p.u. for 0.5 s. At $t = 2.5$ s, then a terminal short-circuit is applied to the motor followed by its removal at $t = 3.0$ s. The response curves of the electromagnetic torque, stator current, rotor speed and rotor flux magnitude are shown in Figs. A.8-A.11, respectively.

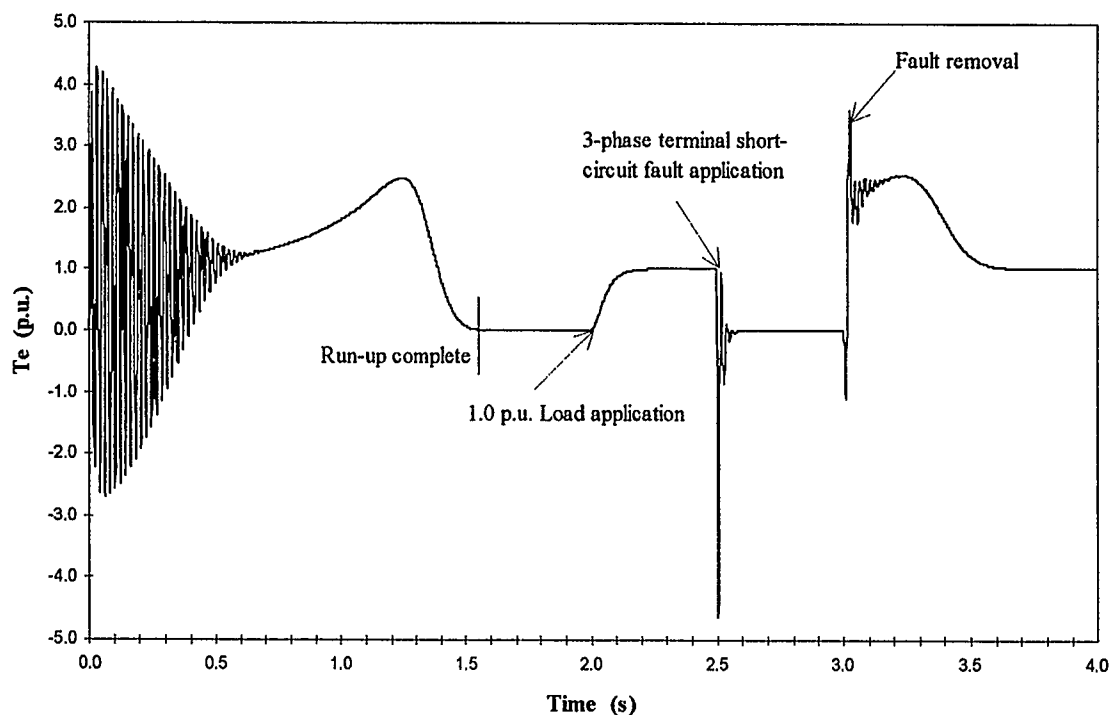


Figure A.8 Electromagnetic torque for various load and fault conditions (line start).

The above simulation was performed on the Sun-sparc station, it took approximately 30 s to complete.

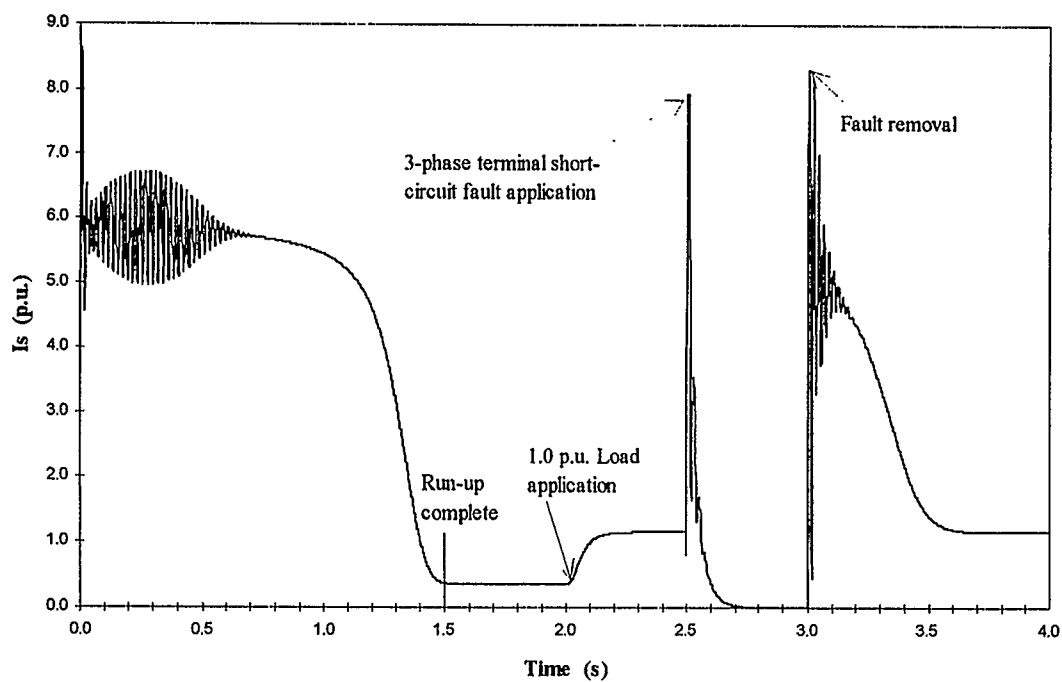


Figure A.9 Stator current for various load and fault conditions (line start).

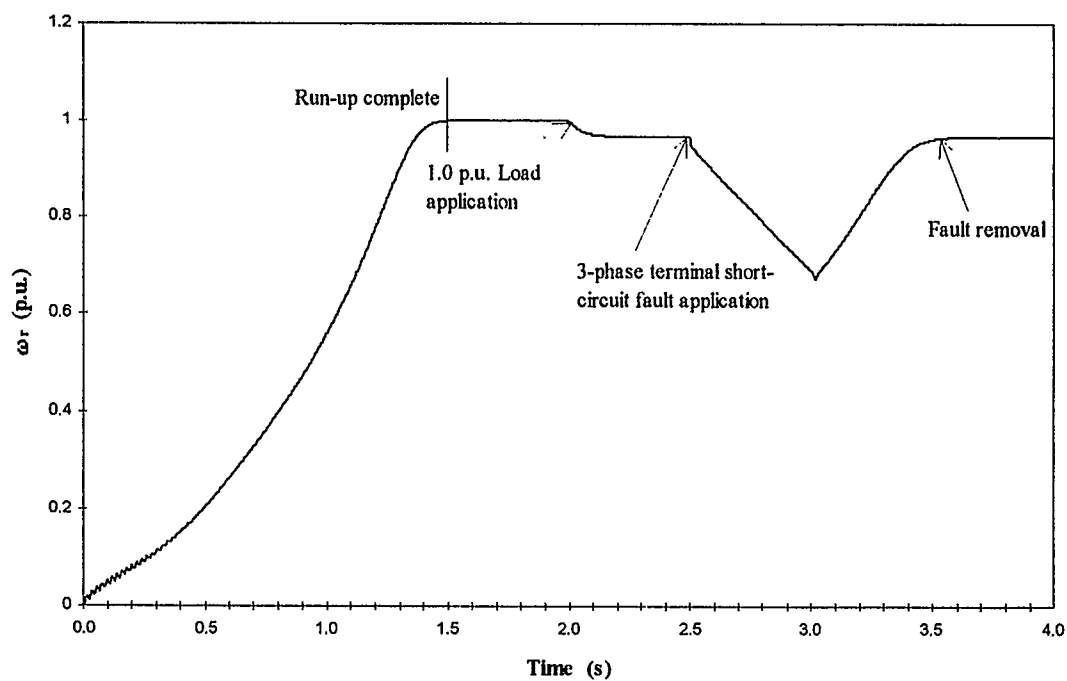


Figure A.10 Rotor speed for various load and fault conditions (line start).

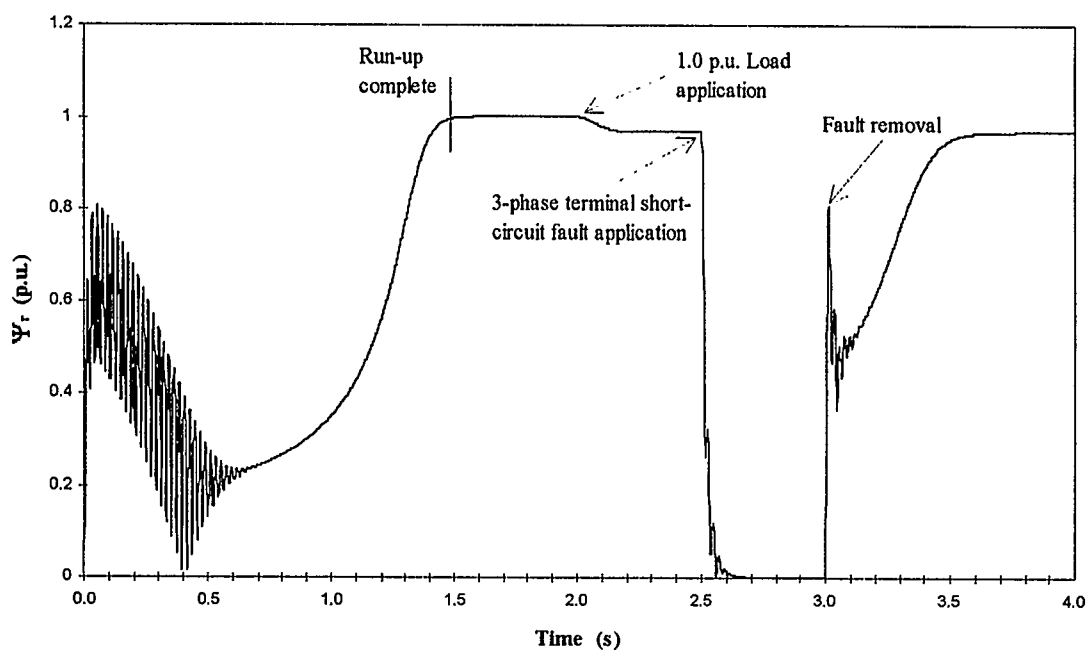


Figure A.11 Rotor flux magnitude for various load and fault conditions (line start).

A.3 Motor Parameters

The following motor parameters are for 2-pole machines.

Parameters	10 hp	30 hp
Line frequency	60 Hz	50 Hz
Stator resistance, R_s	0.0453 p.u.	0.0147 p.u.
Rotor resistance, R_r	0.0222 p.u.	0.0287 p.u.
Stator reactance, X_s	0.0775 p.u.	3.2340 p.u.
Rotor reactance, X_r	0.0322 p.u.	3.2484 p.u.
Leakage reactance, X_m	2.0420 p.u.	3.1568 p.u.
Inertia constant, H	1.0000 p.u.	1.0167 p.u.

A.4 Induction Motor Model Program Listing

The following is a program listing of the induction motor in the stationary reference frame (stator frame).

The Following program is to give an Estimate of the Rotor Flux in an Induction Machine with D-Q axis on Stationary ref. frame. The method used to accomplish this is Runge-Kutta Numerical method (modified).

Author: Toh, K.P. Allan
Date : April 28th 1994
Updated : May 24th 1994

*****/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define pi 3.14159265359
```

```
void ME(double ALI_me[5][5], double R_me[5][5], double G_me[5][5],
double H_me[5][5], double WE_me, double XS_me, double XM_me,
double XR_me, double RS_me, double RR_me, double XRR_me,
double RRR_me, int N_me);
```

```
void RK(int N_rk, double dx_rk, int nes_rk, double Y_rk[8], double E_rk[8],
double v_rk[8], double ALI_rk[5][5], double G_rk[5][5],
double H_rk[5][5], double R_rk[5][5], double am_rk, double *TM_rk,
double VT_rk, double *TE_rk, double WE_rk, double WF_rk, int tq_rk,
int inrun_rk, int k_rk, double XS_rk, double XM_rk, double XR_rk,
double VDSS_rk, double VQSS_rk, double t_rk, double *WRR_rk,
double *CHI_R_rk);
```

```
void AUX(double c_aux[8], double PC_aux[8], double v_aux[8],
double ALI_aux[5][5], double G_aux[5][5], double H_aux[5][5],
double R_aux[5][5], double am_aux, double *TM_aux, double VT_aux,
double *TE_aux, double WE_aux, double WF_aux, int tq_aux,
int inrun_aux, int k_aux, double XS_aux, double XM_aux, double XR_aux,
double VDSS_aux, double VQSS_aux, double t_aux, double *WRR_aux,
double *CHI_R_aux);
```

```
void MM(double A_mm[5][5], double B_mm[5][5], double C_mm[5][5],
```

```

int N_mm);

void main()
{

double c[8], PC[8], v[8], R[5][5], G[5][5], H[5][5], ALI[5][5];
double XS, XM, XR, RS, RR, XRR, RRR, ah, freq, dx, slip, WRR;
double am, TM, VT, TE, WE, WR, WF, VT1, VT2, TM1, TM2, VTT, AIM;
double Ls, Lm, Lr, t, CHI_R, VA_REF_MAX, WR_REF, WE_REF,
KP, KPP, KI;
double WR_ERR, WR_ERR_INTEG, TE_REF, IQS_REF,
IDS_REF, WSL_REF;
double THETAE_REF, IQSS_REF, IDSS_REF, IA_REF, IB_REF, IC_REF;
double IA, IB, IC, IA_ERR, IB_ERR, IC_ERR, VA_REF, VB_REF, VC_REF;
double VAO, VBO, VCO, VDSS, VQSS, IDSS, IQSS, KVSI, W_R, WR_rad_s;
int inrun, ndis, itype, iter, ir, tq, k, N, i, j, nes;

char infile[10];
FILE *outptr1, *outptr2, *inptr;

printf("Enter DATA FILENAME please :\n");
scanf("%s", infile);
printf("Reading & Processing, Please Wait...\n");
inptr = fopen(infile, "r");

outptr1 = fopen("Results.m", "w");
outptr2 = fopen("VI.m", "w");

```

/******

Input Data Stream Sequence

INRUN : No. of iterations before application of disturbance

NDIS : The ending iteration number of the disturbance being applied

ITYPE : Type of Disturbance,

1= Voltage Disturbance

2= Load Disturbance

3= No Disturbance

ITER : Total No. of iterations

IR : Type of starting/initial conditions,

0= Motor Start

1= Read Initial States

TQ : Load type,

0= Free Acceleration

1= Constant Load Torque

2= Load Proportional to square of speed

XS : Total stator reactance
 XM : Mutual reactance
 XR : Total rotor reactance
 RS : Stator resistance
 RR : Rotor resistance
 AH : Inertia constant
 XRR : Rotor reactance (run mode)
 RRR : Rotor resistance (run mode)
 DX : Integration step interval
 FREQ : Supply frequency
 VT : Inverter DC bus Voltage
 TM : Load torque
 VT1 : Voltage disturbance value
 TM1 : Torque disturbance value
 VT2 : Removal of disturbance voltage value (i.e resume value)
 TM2 : Removal of torque disturbance value (i.e resume value)

Variable Dictionary

c[1] = ids, c[2] = iqs, c[3] = idr, c[4] = iqr
 c[5] = theta slip, c[6] = w_slip
 v[1] = vds, v[2] = vqs, v[6] = load torque(TM)
 CHI_r = rotor flux magnitude

*****/

```
fscanf(inptr,"%d", &inrun);
fscanf(inptr,"%d", &ndis);
fscanf(inptr,"%d", &itype);
fscanf(inptr,"%d", &iter);
fscanf(inptr,"%d", &ir);
fscanf(inptr,"%d", &tq);
fscanf(inptr,"%lf", &XS);
fscanf(inptr,"%lf", &XM);
fscanf(inptr,"%lf", &XR);
fscanf(inptr,"%lf", &RS);
fscanf(inptr,"%lf", &RR);
fscanf(inptr,"%lf", &ah);
fscanf(inptr,"%lf", &XRR);
fscanf(inptr,"%lf", &RRR);
fscanf(inptr,"%lf", &dx);
fscanf(inptr,"%lf", &freq);
```



```

fscanf(inptr,"%lf", &VT);
fscanf(inptr,"%lf", &TM);

am = ah/(pi*freq);
WE = 2.0*pi*freq;
WF = 2.0*pi*freq;
  VA_REF_MAX = 1.0;
  WE_REF = 314.0;
  WRR = 0.0;
  VDSS = 0.0;
  VQSS = 1.0;

/*****
Initial States Read in
Order in which the states are read in:
  d-axis stator current
  q-axis stator current
  d-axis rotor current
  q-axis rotor current
*****/
if (ir == 1) {
  for(i=1; i<=7; i++)
    fscanf(inptr,"%lf", &c[i]);
  for(j=1; j<=7; j++)
    fscanf(inptr,"%lf", &v[j]);
}

/***** INITIAL CONDITIONS *****/

if (ir == 0) {
  for(i=1; i<=7; i++){
    c[i] = 0.0;
    v[i] = 0.0;
  }
}
v[1] = VDSS;
v[2] = VQSS;
c[6] = WE;

/***** Dynamic Cycle *****/

fscanf(inptr,"%lf", &VT1);
fscanf(inptr,"%lf", &TM1);

```

```

fscanf(inptr, "%lf", &VT2);
fscanf(inptr, "%lf", &TM2);

for(k = 1; k <= iter; k++) {

    /***** Perform Matrix inversion *****/

    ME(ALI, R, G, H, WE, XS, XM, XR, RS, RR, XRR, RRR, 1);

    /**** Disturbance Application *****/

    if (k == inrun)
        if (itype == 1) VT = VT1;
        else if (itype == 2) TM = TM1;

    /**** Disturbance Removal *****/

    if (k == ndis)
        if (itype == 1) VT = VT2;
        else if (itype == 2) TM = TM2;

    /**** Perform (modified) Runge-Kutta *****/

    t = dx*k;

    RK(7, dx, 1, c, PC, v, ALI, G, H, R, am, &TM, VT, &TE, WE, WF,
    tq, inrun, k, XS, XM, XR, VDSS, VQSS, t, &WRR, &CHI_R);
    Ls = XS/WE;
    Lm = XM/WE;
    Lr = XR/WE;

    /***** Calc. Rotor speed, Terminal Voltage and Current *****/

    W_R = WRR*pi/15.0;          /* Change from RPM to rad/sec */
    slip = (WE - W_R)*100.0/WE;
    VTT = sqrt(v[2]*v[2] + v[1]*v[1]);
    AIM = sqrt(c[2]*c[2] + c[1]*c[1]);

    /***** Print out Time, Terminal Voltage, Slip, Torque, Current,
    Rotor speed and Flux Mag. *****/

    fprintf(outptr1, "%lf\t %lf\t %lf\t %lf\t %lf\t %lf\t %lf\n",

```

```

        t, VTT, slip, TE, AIM, W_R/WE, CHI_R);

    fprintf(outptr2, "%lf\t %lf\t %lf\t %lf\t %lf\t %lf\t %lf\t %lf\n",
        t, IA, IB, IC, c[1], c[2], c[3], c[4]);

    } /* k-loop */

} /*End of main() */

/*****
Subroutine : ME
This subroutine performs matrix inversion
*****/
void ME(double ALI_me[5][5], double R_me[5][5], double G_me[5][5],
double H_me[5][5], double WE_me, double XS_me, double XM_me,
double XR_me, double RS_me, double RR_me, double XRR_me,
double RRR_me, int N_me)

{
    double ALS, ALM, ALR, u;
    int i, j;

    /***** Convert Reactances to Inductances *****/

    ALS = XS_me/WE_me;
    ALM = XM_me/WE_me;
    ALR = XR_me/WE_me;
    for(i = 1; i <= 4; i++){
        for(j = 1; j <= 4; j++){
            ALI_me[i][j] = 0.0;
            R_me[i][j] = 0.0;
            G_me[i][j] = 0.0;
            H_me[i][j] = 0.0;
        }
    }

    /***** Input [R] matrix *****/

    R_me[1][1] = RS_me;
    R_me[2][2] = RS_me;
    R_me[3][3] = RR_me;
    R_me[4][4] = RR_me;

```

```

        /***** Input [G] matrix *****/

G_me[3][2] = XM_me;
G_me[3][4] = XR_me;
G_me[4][1] = -XM_me;
G_me[4][3] = -XR_me;

        /***** Input [H] matrix *****/

H_me[1][2] = XS_me;
H_me[1][4] = XM_me;
H_me[2][1] = -XS_me;
H_me[2][3] = -XM_me;
H_me[3][2] = XM_me;
H_me[3][4] = XR_me;
H_me[4][1] = -XM_me;
H_me[4][3] = -XR_me;

        /***** Running Parameters *****/

if (N_me == 2) {
    ALR = XRR_me/WE_me;
    R_me[3][3] = RRR_me;
    R_me[4][4] = RRR_me;
}

        /***** Input [L] matrice *****/

ALI_me[1][1] = ALR;
ALI_me[1][3] = -ALM;
ALI_me[2][2] = ALR;
ALI_me[2][4] = -ALM;
ALI_me[3][1] = -ALM;
ALI_me[3][3] = ALS;
ALI_me[4][2] = -ALM;
ALI_me[4][4] = ALS;

        /***** CALCULATE [L]^-1 matrice *****/

u = ALS*ALR - ALM*ALM;
for(i = 1; i <= 4; i++) {
    for(j = 1; j <= 4; j++) {
        ALI_me[i][j] = ALI_me[i][j]/u;
    }
}

```

```

}

return;
} /* end of subroutine ME */

/*****
Subroutine : RK
This subroutine RK uses the Runge-Kutta algorithm to calculate
[x]t vector (Merson Modified)
*****/
void RK(int N_rk, double dx_rk, int nes_rk, double Y_rk[8], double E_rk[8],
double v_rk[8], double ALI_rk[5][5], double G_rk[5][5],
double H_rk[5][5], double R_rk[5][5], double am_rk, double *TM_rk,
double VT_rk, double *TE_rk, double WE_rk, double WF_rk, int tq_rk,
int inrun_rk, int k_rk, double XS_rk, double XM_rk, double XR_rk,
double VDSS_rk, double VQSS_rk, double t_rk, double *WRR_rk,
double *CHI_R_rk)

{

double A[8], B[8], C[8], D[8], h, Z;
int i;
h = dx_rk/3.0;
for(i = nes_rk; i <= N_rk; i++) D[i] = Y_rk[i];

AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
TE_rk, WE_rk, WF_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
VDSS_rk, VQSS_rk, t_rk, WRR_rk, CHI_R_rk);

for(i = nes_rk; i <= N_rk; i++) {
A[i] = h*E_rk[i];
Y_rk[i] = D[i] + A[i];
}

AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
TE_rk, WE_rk, WF_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
VDSS_rk, VQSS_rk, t_rk, WRR_rk, CHI_R_rk);

for(i = nes_rk; i <= N_rk; i++) {
B[i] = h*E_rk[i];
Y_rk[i] = D[i] + (A[i] + B[i])*0.5;
}

```

```
AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
    TE_rk, WE_rk, WF_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
    VDSS_rk, VQSS_rk, t_rk, WRR_rk, CHI_R_rk);
```

```
for(i = nes_rk; i <= N_rk; i++) {
    B[i] = h*E_rk[i];
    Y_rk[i] = D[i] + (A[i] + B[i]*3.0)*0.375;
}
```

```
AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
    TE_rk, WE_rk, WF_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
    VDSS_rk, VQSS_rk, t_rk, WRR_rk, CHI_R_rk);
```

```
for(i = nes_rk; i <= N_rk; i++) {
    C[i] = h*E_rk[i];
    Y_rk[i] = D[i] + (A[i] - B[i]*3.0 + C[i]*4.0)*1.5;
}
```

```
AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
    TE_rk, WE_rk, WF_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
    VDSS_rk, VQSS_rk, t_rk, WRR_rk, CHI_R_rk);
```

```
for(i = nes_rk; i <= N_rk; i++) {
    Z = D[i];
    D[i] = h*E_rk[i];
    Y_rk[i] = Z + (A[i] + C[i]*4.0 + D[i])*0.5;
}
```

```
return;
```

```
} /* end of subroutine RK */
```

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

This subroutine AUX is used by the Runge-Kutta algorithm
to calc. [x]t vector, i.e. iqs, ids, iqr, idr

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

```
void AUX(double c_aux[8], double PC_aux[8], double v_aux[8],
double ALI_aux[5][5], double G_aux[5][5], double H_aux[5][5],
double R_aux[5][5], double am_aux, double *TM_aux, double VT_aux,
double *TE_aux, double WE_aux, double WF_aux, int tq_aux,
int inrun_aux, int k_aux, double XS_aux, double XM_aux, double XR_aux,
double VDSS_aux, double VQSS_aux, double t_aux, double *WRR_aux,
double *CHI_R_aux)
```

```

{

double A[8][8], B[8][8], F[5][5], Z[5][5], it[2][5], it_G[2][5];
double ALS, ALM, ALR, W, WR, WD, it_G_I, T_accel, I[5][2];
int i, j, l;
double temp, CHI_dr, CHI_qr ;

    /**** Convert Reactances to Inductances *****/

    ALS = XS_aux/WE_aux;
    ALM = XM_aux/WE_aux;
    ALR = XR_aux/WE_aux;
    W = c_aux[6];
    WR = WE_aux - W;

    v_aux[1] = -VT_aux*(double)sin((double)WE_aux*t_aux);
    v_aux[2] = VT_aux*(double)cos((double)WE_aux*t_aux);

    v_aux[6] = *TM_aux;
    for(i = 1; i <= 4; i++) {
        for(j = 1; j <= 4; j++) {
            F[i][j] = *WRR_aux*pi/15.0*G_aux[i][j]/WE_aux + R_aux[i][j];
        }
    }

    MM(ALI_aux, F, Z, 4);
    for(i = 1; i <= 7; i++) {
        for(j = 1; j <= 7; j++) {
            if ( (i<5) && (j<5) ) {
                A[i][j] = -Z[i][j];
                B[i][j] = ALI_aux[i][j];
            }
            else A[i][j] = B[i][j] = 0.0 ;
        }
    }

    A[5][6] = 1.0;
    A[6][1] = -v_aux[1]/am_aux + c_aux[1]*R_aux[1][1]/am_aux;
    A[6][2] = -v_aux[2]/am_aux + c_aux[2]*R_aux[2][2]/am_aux;
    A[7][1] = ALM*R_aux[3][3]/ALR;
    A[7][7] = -R_aux[3][3]/ALR;
    B[6][6] = 1.0/am_aux;

```

```

for(i = 1; i <= 7; i++) {
    PC_aux[i] = 0.0;
    for(j = 1; j <= 7; j++)
        PC_aux[i] = PC_aux[i] + A[i][j]*c_aux[j] + B[i][j]*v_aux[j];
    }

    /***** Calc. TE = it*[G]*I, it = I-transpose *****/

for(l = 1; l <= 4; l++) {
    it_G[l][l] = 0.0;
    for(j = 1; j <= 4; j++) {
        it[l][j] = c_aux[j];
        it_G[l][l] = it_G[l][l] + it[l][j]*G_aux[j][l];
    }
}

it_G_I = 0.0;
for(j = 1; j <= 4; j++) {
    I[j][1] = c_aux[j];
    it_G_I = it_G_I + it_G[l][j] * I[j][1];
}

*TE_aux = it_G_I;
if (tq_aux == 0) T_accel = *TE_aux - *TM_aux;
if (tq_aux == 2) {
    WD = (*WRR_aux*pi/15.0)/(WE_aux);
    *TM_aux = 1.1*WD*WD;
}

if (tq_aux == 1) {
    *TM_aux = 1.0;
}

T_accel = *TE_aux - *TM_aux;

/***** Calc. of WR using Euler's approx. (step size = 0.001) *****/

*WRR_aux = *WRR_aux + ((T_accel)/am_aux)*0.001; /* in RPM */

/***** Calc. Flux Magnitude *****/

CHI_dr = ALR*(c_aux[3]) + ALM*(c_aux[1]);
CHI_qr = ALR*(c_aux[4]) + ALM*(c_aux[2]);

```



```

*CHI_R_aux = sqrt((CHI_dr*CHI_dr) + (CHI_qr*CHI_qr));

return;

} /* end of subroutine AUX */

/*****
This subroutine MM is used by AUX subroutine to multiply matrices to obtain
[L]^-1(Wr[G] + [R])
*****/

void MM(double A_mm[5][5], double B_mm[5][5], double C_mm[5][5], int N_mm)

{

int i, j, l;
for(i = 1; i <= 4; i++) {
    for(l = 1; l <= 4; l++) {
        C_mm[i][l] = 0.0;
        for(j = 1; j <= 4; j++) {
            C_mm[i][l] = C_mm[i][l] + A_mm[i][j]*B_mm[j][l];
        }
    }
}

return;

} /* end of subroutine MM */

```

Appendix B

B.1 Field Oriented Control of Induction Motor Program Listing

The following is a program listing in C of the FOC drive system with PI controllers incorporated (uses machine model from appendix A.4).

/******

The Following program is to give an Estimate of the Rotor Flux Magnitude and Angle in an Induction Machine with the D-Q axis on Synchronous Rotating Ref. frame. The induction motor model is based on the stationary (stator) ref. frame, the numerical method used to accomplish this is Runge-Kutta method (modified).

Using motor model fixed on STATOR (stationary reference frame). The sine and cosine of angle ϕ (i.e the field angle) will be calculated. This program incorporates Field Oriented Control for the induction motor.

Author: Toh, K.P. Allan
Date : April 28th 1994
Updated : June 16th 1994

*****/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define pi 3.14159265359
```

```
void ME(double ALI_me[5][5], double R_me[5][5], double G_me[5][5],
double H_me[5][5], double WE_me, double XS_me, double XM_me,
double XR_me, double RS_me, double RR_me, double XRR_me,
double RRR_me, int N_me);
```

```
void RK(int N_rk, double dx_rk, int nes_rk, double Y_rk[5], double E_rk[5],
double v_rk[5], double ALI_rk[5][5], double G_rk[5][5],
double H_rk[5][5], double R_rk[5][5], double am_rk, double *TM_rk,
double VT_rk, double *TE_rk, double WE_rk, int tq_rk,
int inrun_rk, int k_rk, double XS_rk, double XM_rk, double XR_rk,
double *Vds_sta_rk, double *Vqs_sta_rk, double t_rk, double *WRR_rk,
double *Ids_syn_rk, double *Iqs_syn_rk, double *Idr_syn_rk,
double *Iqr_syn_rk, double *CHI_r_rk, double *CHI_dr_sta_rk,
double *CHI_qr_sta_rk, double *Sin_Phi_rk, double *Cos_Phi_rk);
```

```

void AUX(double c_aux[5], double PC_aux[5], double v_aux[5],
double ALI_aux[5][5], double G_aux[5][5], double H_aux[5][5],
double R_aux[5][5], double am_aux, double *TM_aux, double VT_aux,
double *TE_aux, double WE_aux, int tq_aux, int inrun_aux, int k_aux,
double XS_aux, double XM_aux, double XR_aux, double *Vds_sta_aux,
double *Vqs_sta_aux, double t_aux, double *WRR_aux,
double *Ids_syn_aux, double *Iqs_syn_aux, double *Idr_syn_aux,
double *Iqr_syn_aux, double *CHI_r_aux, double *CHI_dr_sta_aux,
double *CHI_qr_sta_aux, double *Sin_Phi_aux, double *Cos_Phi_aux);

```

```

void MM(double A_mm[5][5], double B_mm[5][5], double C_mm[5][5],
int N_mm);

```

```

void main()

```

```

{

```

```

double c[5], PC[5], v[5], R[5][5], G[5][5], H[5][5], ALI[5][5];
double XS, XM, XR, RS, RR, XRR, RRR, ah, freq, dx, WRR;
double am, TM, VT, TE, WE, WR, VT1, VT2, TM1, TM2, VTT, AIM;
double Ls, Lm, Lr, t, CHI_r, KP, KPP, KI, WE_base;
double Iqs_syn_REF, Ids_syn_REF, WSL_REF;
double Iqs_sta_REF, Ids_sta_REF;
double VAO, VBO, VCO, Vds_sta, Vqs_sta, Ids_sta, Iqs_sta, W_r;
double Ids_syn, Iqs_syn, Idr_syn, Iqr_syn, Vds_foc, Vqs_foc, PHI;
double Sin_Phi, Cos_Phi, Ids_foc, Iqs_foc, CHI_dr_sta, CHI_qr_sta;
double Iqr_sta, sin_err, cos_err, Sin_Phi_AUX, Cos_Phi_AUX, Ids_ERR;
double Ids_ERR_INTEG, Iqs_ERR_INTEG, Ids_foc_REF, Iqs_foc_REF;
double Vqs_foc_lim, vds_foc, vqs_foc, CHI_r_ERR_INTEG, CHI_REF;
double KP_CHI, KI_CHI, CHI_base, Idr_sta, Iqs_ERR;
double Ids_foc_lim, Vds_foc_lim, CHI_r_ERR, WR_ERR_INTEG;
double WR_ERR, WR_REF, KP_WR, KI_WR, Iqs_foc_lim;

```

```

int inrun, ndis, itype, iter, ir, tq, k, N, i, j, nes;

```

```

char infile[10];

```

```

FILE *inptr, *outptr1, *outptr2;

```

```

printf("Enter DATA FILENAME please :\n");
scanf("%s", infile);

```

```
printf("Reading & Processing, Please Wait...\n");
inptr = fopen(infile, "r");
outptr1 = fopen("Results.m", "w");
outptr2 = fopen("VI.m", "w");
```

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

Input Data Stream Sequence

```
INRUN : No. of iterations before application of disturbance
NDIS  : The ending iteration number of the disturbance being applied
ITYPE : Type of Disturbance,
        1= Voltage Disturbance
        2= Load Disturbance
        3= No Disturbance
ITER  : Total No. of iterations
IR    : Type of starting/initial conditions,
        0= Motor Start
        1= Read Initial States
TQ    : Load type,
        0 = Free Acceleration
        1 = Constant Load Torque
        2 = Load Proportional to square of speed
XS    : Total stator reactance
XM    : Mutual reactance
XR    : Total rotor reactance
RS    : Stator resistance
RR    : Rotor resistance
AH    : Inertia constant
XRR   : Rotor reactance (run mode)
RRR   : Rotor resistance (run mode)
DX    : Integration step interval
FREQ  : Supply frequency
VT    : Inverter DC bus Voltage
TM    : Load torque
VT1   : Voltage disturbance value
TM1   : Torque disturbance value
VT2   : Removal of disturbance voltage value (i.e resume value)
IM2   : Removal of torque disturbance value (i.e resume value)
```

Variable Dictionary

```
c[1] = ids, c[2] = iqs, c[3] = idr, c[4] = iqr
```

```

c[5] = theta_slip, c[6] = w_slip
v[1] = vds, v[2] = vqs, v[6] = load torque(TM)
CHI_r = rotor flux magnitude

```

```

*****/

```

```

fscanf(inptr,"%d", &inrun);
fscanf(inptr,"%d", &ndis);
fscanf(inptr,"%d", &itype);
fscanf(inptr,"%d", &iter);
fscanf(inptr,"%d", &ir);
fscanf(inptr,"%d", &tq);
fscanf(inptr,"%lf", &XS);
fscanf(inptr,"%lf", &XM);
fscanf(inptr,"%lf", &XR);
fscanf(inptr,"%lf", &RS);
fscanf(inptr,"%lf", &RR);
fscanf(inptr,"%lf", &ah);
fscanf(inptr,"%lf", &XRR);
fscanf(inptr,"%lf", &RRR);
fscanf(inptr,"%lf", &dx);
fscanf(inptr,"%lf", &freq);
fscanf(inptr,"%lf", &VT);
fscanf(inptr,"%lf", &TM);

```

```

/**** CHI_base is different for differnt m/c *****/

```

```

/*CHI_base = 393.7007874; */ /* 10 hp */
CHI_base = 323.2062055; /* 30 hp */

```

```

/**** Initial States *****/

```

```

am = ah/(pi*freq);
WE = WE_base = 2.0*pi*freq;
WRR = 0.0;
CHI_r = 0.0;
CHI_REF = 1.0;
WR_REF = 310.0;
Vds_foc_lim = 1.0;
Vqs_foc_lim = 1.0;
Ids_foc_lim = 6.0;
Iqs_foc_REF = 6.0;
Vds_sta = 0.0;

```

```

Vqs_sta = 0.0;

/**** P.I. controllers coefficients ****/

KP = 0.8;
KI = 400.0;
KP_WR = 0.1;
KI_WR = 0.05;
KP_CHI = 100.0;
KI_CHI = 100.0;

/*****
Initial States Read in
Order in which the states are read in:
d-axis stator current
q-axis stator current
d-axis rotor current
q-axis rotor current
*****/

if (ir == 1) {
    for(i=1; i<=4; i++)
        fscanff(inptr, "%lf", &c[i]);
    for(j=1; j<=4; j++)
        fscanff(inptr, "%lf", &v[j]);
}

/***** INITIAL CONDITIONS *****/

if (ir == 0) {
    for(i=1; i<=4; i++){
        c[i] = 0.001;
        v[i] = 0.001;
    }
}

/***** Dynamic Cycle *****/

fscanff(inptr, "%lf", &VT1);
fscanff(inptr, "%lf", &TM1);
fscanff(inptr, "%lf", &VT2);
fscanff(inptr, "%lf", &TM2);

```

```

for(k = 1; k <= iter; k++) {

    /***** Perform Matrix inversion *****/

    ME(ALI, R, G, H, WE, XS, XM, XR, RS, RR, XRR, RRR, 1);

    /**** Disturbance Application *****/

    if (k == inrun)
        if (itype == 1) VT = VT1;
    else if (itype == 2) TM = TM1;

    /**** Disturbance Removal *****/

    if (k == ndis)
        if (itype == 1) VT = VT2;
    else if (itype == 2) TM = TM2;

    /**** Perform (modified) Runge-Kutta *****/

    t = dx*k;

    RK(4, dx, 1, c, PC, v, ALI, G, H, R, am, &TM, VT, &TE,
    WE, tq, inrun, k, XS, XM, XR, &Vds_sta, &Vqs_sta, t, &WRR,
    &Ids_syn, &Iqs_syn, &Idr_syn, &Iqr_syn, &CHI_r, &CHI_dr_sta,
    &CHI_qr_sta, &Sin_Phi_AUX, &Cos_Phi_AUX);

    Ls = XS/WE;
    Lm = XM/WE;
    Lr = XR/WE;

    /**** Change variable name *****/

    Ids_sta = c[1];
    Iqs_sta = c[2];
    Idr_sta = c[3];
    Iqr_sta = c[4];

    /***** Calc. Terminal Voltage, Current, sine & cos of field angle (phi) *****/

    W_r = WRR*pi/15.0;
    VTT = sqrt(v[2]*v[2] + v[1]*v[1]);
    /* Change from RPM to rad/sec */

```

```

AIM = sqrt((c[2]*c[2] + c[1]*c[1]));

WR = W_r;
Sin_Phi = CHI_qr_sta/(CHI_r);
Cos_Phi = CHI_dr_sta/(CHI_r);

/***** Perform Stationary to FOC Co-ordinate change *****/

Ids_foc = Iqs_sta*Sin_Phi + Ids_sta*Cos_Phi;
Iqs_foc = Iqs_sta*Cos_Phi - Ids_sta*Sin_Phi;

/***** Calc. of P.I. Controller Command Signals for CHI_r *****/

CHI_r_ERR = (CHI_REF - CHI_r*CHI_base);
CHI_r_ERR_INTEG = CHI_r_ERR_INTEG + CHI_r_ERR*dx;
Ids_foc_REF = KP_CHI*CHI_r_ERR + KI_CHI*CHI_r_ERR_INTEG;

/***** Adding Current limiter for Ids foc ref. *****/

if (Ids_foc_REF > Ids_foc_lim) Ids_foc_REF = Ids_foc_lim;

/***** Calc. of P.I. Controller Command Signals for Wr *****/

WR_ERR = (WR_REF - WR);
WR_ERR_INTEG = WR_ERR_INTEG + WR_ERR*dx;
Iqs_foc_REF = KP_WR*WR_ERR + KI_WR*WR_ERR_INTEG;

/***** Adding Current limiter for Iqs foc ref. *****/

if (Iqs_foc_REF > Iqs_foc_lim) Iqs_foc_REF = Ids_foc_lim;

/***** Calc. of P.I. Controller Cmd Sig. for Vds & Vqs FOC *****/

Ids_ERR = (Ids_foc_REF - Ids_foc);
Iqs_ERR = (Iqs_foc_REF - Iqs_foc);
Ids_ERR_INTEG = Ids_ERR_INTEG + Ids_ERR*dx;
Iqs_ERR_INTEG = Iqs_ERR_INTEG + Iqs_ERR*dx;
Vds_foc = KP*Ids_ERR + KI*Ids_ERR_INTEG;
Vqs_foc = KP*Iqs_ERR + KI*Iqs_ERR_INTEG;

```



```

/***** Adding Voltage limiter *****/

if (Vds_foc > Vds_foc_lim) Vds_foc = Vds_foc_lim;
if (Vqs_foc > Vqs_foc_lim) Vqs_foc = Vqs_foc_lim;

/***** Perform FOC. Ref frame to Stat. Ref frame transfm *****/

Vds_sta = Vds_foc*cos_Phi - Vqs_foc*sin_Phi;
Vqs_sta = Vqs_foc*cos_Phi + Vds_foc*sin_Phi;

/***** Print out Time, Terminal Voltage, sin (phi), Torque, Current,
Rotor speed and Flux Mag. *****/

fprintf(outptr1, "%lf\t %lf\t %lf\t %lf\t %lf\t %lf\t %lf\n",
t, VTT, Sin_Phi, TE, AIM, W_r/WE_base, CHI_r*CHI_base);

fprintf(outptr2, "%lf\t %lf\t %lf\t %lf\t %lf\t %lf\t %lf\t %lf\n",
t, Vds_sta, Vqs_sta, Ids_foc_REF, Ids_foc, Iqs_foc, Sin_Phi, CHI_r_ERR);

} /* k-loop */

} /*End of main() */

/*****
Subroutine : ME
This subroutine performs matrix inversion
*****/

void ME(double ALI_me[5][5], double R_me[5][5], double G_me[5][5],
double H_me[5][5], double WE_me, double XS_me, double XM_me,
double XR_me, double RS_me, double RR_me, double XRR_me,
double RRR_me, int N_me)

{

double ALS, ALM, ALR, u;
int i, j;

/***** Convert Reactances to Inductances *****/

ALS = XS_me/WE_me;
ALM = XM_me/WE_me;

```

```

ALR = XR_me/WE_me;
for(i = 1; i <= 4; i++){
    for(j = 1; j <= 4; j++){
        ALI_me[i][j] = 0.0;
        R_me[i][j] = 0.0;
        G_me[i][j] = 0.0;
        H_me[i][j] = 0.0;
    }
}

/***** Input [R] matrix *****/

R_me[1][1] = RS_me;
R_me[2][2] = RS_me;
R_me[3][3] = RR_me;
R_me[4][4] = RR_me;

/***** Input [G] matrix in reactance *****/

G_me[3][2] = XM_me;
G_me[3][4] = XR_me;
G_me[4][1] = -XM_me;
G_me[4][3] = -XR_me;

/***** Input [H] matrix in inductance *****/

H_me[3][2] = XM_me/WE_me;
H_me[3][4] = XR_me/WE_me;
H_me[4][1] = -XM_me/WE_me;
H_me[4][3] = -XR_me/WE_me;

/***** Running Parameters *****/

if (N_me == 2) {
    ALR = XRR_me/WE_me;
    R_me[3][3] = RRR_me;
    R_me[4][4] = RRR_me;
}

/***** Input [L] matrice *****/

ALI_me[1][1] = ALR;

```

```

ALI_me[1][3] = -ALM;
ALI_me[2][2] = ALR;
ALI_me[2][4] = -ALM;
ALI_me[3][1] = -ALM;
ALI_me[3][3] = ALS;
ALI_me[4][2] = -ALM;
ALI_me[4][4] = ALS;

```

```

/***** CALCULATE [L]^-1 matrice *****/

```

```

u = ALS*ALR - ALM*ALM;
for(i = 1; i <= 4; i++) {
    for(j = 1; j <= 4; j++) {
        ALI_me[i][j] = ALI_me[i][j]/u;
    }
}

```

```

return;
} /* end of subroutine ME */

```

```

/*****

```

Subroutine : RK

This subroutine RK uses the Runge-Kutta algorithm to calculate [x]t vector
(Merson Modified)

```

*****/

```

```

void RK(int N_rk, double dx_rk, int nes_rk, double Y_rk[5], double E_rk[5],
double v_rk[5], double ALI_rk[5][5], double G_rk[5][5],
double H_rk[5][5], double R_rk[5][5], double am_rk, double *TM_rk,
double VT_rk, double *TE_rk, double WE_rk, int tq_rk,
int inrun_rk, int k_rk, double XS_rk, double XM_rk, double XR_rk,
double *Vds_sta_rk, double *Vqs_sta_rk, double t_rk, double *WRR_rk,
double *Ids_syn_rk, double *Iqs_syn_rk, double *Idr_syn_rk,
double *Iqr_syn_rk, double *CHI_r_rk, double *CHI_dr_sta_rk,
double *CHI_qr_sta_rk, double *Sin_Phi_rk, double *Cos_Phi_rk)

```

```

{

```

```

double A[5], B[5], C[5], D[5], h, Z;
int i;
h = dx_rk/3.0;
for(i = nes_rk; i <= N_rk; i++) D[i] = Y_rk[i];

```

```
AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
TE_rk, WE_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
Vds_sta_rk, Vqs_sta_rk, t_rk, WRR_rk, Ids_syn_rk, Iqs_syn_rk,
Idr_syn_rk, Iqr_syn_rk, CHI_r_rk, CHI_dr_sta_rk, CHI_qr_sta_rk,
Sin_Phi_rk, Cos_Phi_rk);
```

```
for(i = nes_rk; i <= N_rk; i++) {
  A[i] = h*E_rk[i];
  Y_rk[i] = D[i] + A[i];
}
```

```
AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
TE_rk, WE_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
Vds_sta_rk, Vqs_sta_rk, t_rk, WRR_rk, Ids_syn_rk, Iqs_syn_rk,
Idr_syn_rk, Iqr_syn_rk, CHI_r_rk, CHI_dr_sta_rk, CHI_qr_sta_rk,
Sin_Phi_rk, Cos_Phi_rk);
```

```
for(i = nes_rk; i <= N_rk; i++) {
  B[i] = h*E_rk[i];
  Y_rk[i] = D[i] + (A[i] + B[i])*0.5;
}
```

```
AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
TE_rk, WE_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
Vds_sta_rk, Vqs_sta_rk, t_rk, WRR_rk, Ids_syn_rk, Iqs_syn_rk,
Idr_syn_rk, Iqr_syn_rk, CHI_r_rk, CHI_dr_sta_rk, CHI_qr_sta_rk,
Sin_Phi_rk, Cos_Phi_rk);
```

```
for(i = nes_rk; i <= N_rk; i++) {
  B[i] = h*E_rk[i];
  Y_rk[i] = D[i] + (A[i] + B[i]*3.0)*0.375;
}
```

```
AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
TE_rk, WE_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
Vds_sta_rk, Vqs_sta_rk, t_rk, WRR_rk, Ids_syn_rk, Iqs_syn_rk,
Idr_syn_rk, Iqr_syn_rk, CHI_r_rk, CHI_dr_sta_rk, CHI_qr_sta_rk,
Sin_Phi_rk, Cos_Phi_rk);
```

```
for(i = nes_rk; i <= N_rk; i++) {
  C[i] = h*E_rk[i];
  Y_rk[i] = D[i] + (A[i] - B[i]*3.0 + C[i]*4.0)*1.5;
```

```

    }

    AUX(Y_rk, E_rk, v_rk, ALI_rk, G_rk, H_rk, R_rk, am_rk, TM_rk, VT_rk,
    TE_rk, WE_rk, tq_rk, inrun_rk, k_rk, XS_rk, XM_rk, XR_rk,
    Vds_sta_rk, Vqs_sta_rk, t_rk, WRR_rk, Ids_syn_rk, Iqs_syn_rk,
    Idr_syn_rk, Iqr_syn_rk, CHI_r_rk, CHI_dr_sta_rk, CHI_qr_sta_rk,
    Sin_Phi_rk, Cos_Phi_rk);

    for(i = nes_rk; i <= N_rk; i++) {
        Z = D[i];
        D[i] = h*E_rk[i];
        Y_rk[i] = Z + (A[i] + C[i]*4.0 + D[i])*0.5;
    }

    return;
} /* end of subroutine RK */

/*****
This subroutine AUX is used by the Runge-Kutta algorithm to calc. [x]t vector,
i.e. iqs, ids, iqr, idr
*****/
void AUX(double c_aux[5], double PC_aux[5], double v_aux[5],
double ALI_aux[5][5], double G_aux[5][5], double H_aux[5][5],
double R_aux[5][5], double am_aux, double *TM_aux, double VT_aux,
double *TE_aux, double WE_aux, int tq_aux,
int inrun_aux, int k_aux, double XS_aux, double XM_aux,
double XR_aux, double *Vds_sta_aux, double *Vqs_sta_aux, double t_aux,
double *WRR_aux, double *Ids_syn_aux, double *Iqs_syn_aux,
double *Idr_syn_aux, double *Iqr_syn_aux, double *CHI_r_aux,
double *CHI_dr_sta_aux, double *CHI_qr_sta_aux, double *Sin_Phi_aux,
double *Cos_Phi_aux)

{

double A[5][5], B[5][5], F[5][5], Z[5][5], it[2][5], it_G[2][5];
double ALS, ALM, ALR, W, WR, WD, it_G_I, T_accel;
int i, j, l;
double CHI_dr_syn, CHI_qr_syn, CHI_dr_sta, CHI_qr_sta, I[5][2];

/***** Convert Reactances to Inductances *****/
ALS = XS_aux/WE_aux;
ALM = XM_aux/WE_aux;

```

```
ALR = XR_aux/WE_aux;
```

```
/****** Voltages in STATOR Ref. frame******/
```

```
v_aux[1] = *Vds_sta_aux;
v_aux[2] = *Vqs_sta_aux;
for(i = 1; i <= 4; i++) {
    for(j = 1; j <= 4; j++) {
        F[i][j] = *WRR_aux*pi/15.0*H_aux[i][j] + R_aux[i][j];
    }
}
```

```
MM(ALI_aux, F, Z, 4);
for(i = 1; i <= 4; i++) {
    for(j = 1; j <= 4; j++) {
        A[i][j] = -Z[i][j];
        B[i][j] = ALI_aux[i][j];
    }
}
```

```
for(i = 1; i <= 4; i++) {
    PC_aux[i] = 0.0;
    for(j = 1; j <= 4; j++)
        PC_aux[i] = PC_aux[i] + A[i][j]*c_aux[j] + B[i][j]*v_aux[j];
}
```

```
/****** Converting frame fixed on STATIONARY (Stator) frame to
           frame fixed on SYNCHRONOUS ROTATING
           frame valid for fix WE_aux only. *****/
```

```
*Ids_syn_aux = c_aux[1]*(double)cos((double)WE_aux*t_aux) +
c_aux[2]*(double)sin((double)WE_aux*t_aux);
*Iqs_syn_aux = -c_aux[1]*(double)sin((double)WE_aux*t_aux)+
c_aux[2]*(double)cos((double)WE_aux*t_aux);
*Idr_syn_aux = c_aux[3]*(double)cos((double)WE_aux*t_aux) +
c_aux[4]*(double)sin((double)WE_aux*t_aux);
*Iqr_syn_aux = -c_aux[3]*(double)sin((double)WE_aux*t_aux)+
c_aux[4]*(double)cos((double)WE_aux*t_aux);
```

```

/***** Calc. TE = it*[G]*I, it = I-transpose *****/
for(l = 1; l <= 4; l++) {
    it_G[l][l] = 0.0;
    for(j = 1; j <= 4; j++) {
        it[l][j] = c_aux[j];
        it_G[l][l] = it_G[l][l] + it[l][j]*G_aux[j][l];
    }
}

it_G_I = 0.0;
for(j = 1; j <= 4; j++) {
    I[j][1] = c_aux[j];
    it_G_I = it_G_I + it_G[1][j] * I[j][1];
}

*TE_aux = it_G_I;

if (tq_aux == 0) T_accel = *TE_aux - *TM_aux;

if (tq_aux == 2) {
    WD = (*WRR_aux*pi/15.0)/(WE_aux);
    *TM_aux = 1.1*WD*WD;
}

if (tq_aux == 1) {
    *TM_aux = 1.0;
}
T_accel = *TE_aux - *TM_aux;

/***** Calc. of WR using Euler's approx. (step size = 0.001) *****/

*WRR_aux = *WRR_aux + ((T_accel)/am_aux)*0.001; /* in RPM */

/***** Calc. d-q Flux Magnitude in Syn. frame *****/

CHI_dr_syn = ALR*(*Idr_syn_aux) + ALM*(*Ids_syn_aux);
CHI_qr_syn = ALR*(*Iqr_syn_aux) + ALM*(*Iqs_syn_aux);

/***** Calc. d-q Flux Magnitude in Stationary frame *****/

*CHI_dr_sta_aux = ALR*(c_aux[3]) + ALM*(c_aux[1]);
*CHI_qr_sta_aux = ALR*(c_aux[4]) + ALM*(c_aux[2]);

```

```

/**** Rotor Flux Magnitude *****/

*CHI_r_aux = sqrt((*CHI_dr_sta_aux*(*CHI_dr_sta_aux)) +
                  (*CHI_qr_sta_aux*(*CHI_qr_sta_aux)));

/**** Rotor Flux Angle (Field Angle, Phi) *****/

*Sin_Phi_aux = *CHI_qr_sta_aux/*CHI_r_aux);
*Cos_Phi_aux = *CHI_dr_sta_aux/*CHI_r_aux);

return;
} /* end of subroutine AUX */

/*****
This subroutine MM is used by AUX subroutine to multiply matrices to
obtain  $[L]^{-1}(Wr[G] + [R])$ 
*****/

void MM(double A_mm[5][5], double B_mm[5][5], double C_mm[5][5],
int N_mm)

{

int i, j, l;
for(i = 1; i <= 4; i++) {
    for(l = 1; l <= 4; l++) {
        C_mm[i][l] = 0.0;
        for(j = 1; j <= 4; j++) {
            C_mm[i][l] = C_mm[i][l] + A_mm[i][j]*B_mm[j][l];
        }
    }
}

return;
} /* end of subroutine MM */

```