ARTIFICIAL NEURAL NETWORKS IN INDUCTION MOTOR SPEED ESTIMATION AND CONTROL

PRASHANT MEHROTRA


CENTRE FOR NFLD. STUDIES
MEMORIAL UNIVERSITY
OF NEWFOUNDLAND

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

ARTIFICIAL NEURAL NETWORKS IN INDUCTION MOTOR SPEED ESTIMATION AND CONTROL

by

©Prashant Mehrotra, M.Tech., I.I.T. Mumbai, India

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Faculty of Engineering & Applied Science
Memorial University of Newfoundland
St. John's, Newfoundland, Canada

January, 1999

Abstract

The squirrel-cage induction motor has various inherent advantages not present in other types of ac motors, and is widely used in the industry. Its usage is expected to go up because of possible applications like electric vehicles, which require a light and efficient motor drive. However, the induction motor has a complex and non-linear structure which makes precise control a complicated and expensive process. Added to this complexity is the fact that the motor parameters undergo a variation during regular operation, chiefly due to a change in temperature and nonlinear magnetic characteristics. This variation reduces the efficacy of the control technique, though its effect can be mitigated with the help of robust control techniques. Also, most control techniques require speed feedback from a shaft encoder and these devices have various disadvantages and are considered undesirable for a number of applications. Thus, present day research in this area is mostly focussed on obtaining speed sensorless and robust induction motor drives.

Artificial neural networks (ANNs) have shown great promise in image processing and control applications where robustness is desirable. However, these are at the stage of infancy in the area of induction motor control. The ability of ANNs to map arbitrary nonlinear functions has been used to advantage by many researchers. The motivation behind this work was to investigate the possibility of using ANNs to eventually come up with an ANN based sensorless induction motor drive. This central idea was broken down into two major components — speed estimation of induction motors using ANNs, and control of induction motors using ANNs. Both these areas have attracted attention in recent years, though very little work has been done so far. Because of the complexity of the problem, researchers have been unable to come up with a satisfactory solution.

This work makes an important contribution to the area of induction motor drives,

by presenting for the first time, off-line trained ANN speed estimators. Using the d-q axis dynamic equations of the squirrel-cage induction motor, four methods are proposed whereby an ANN is trained off-line to estimate the speed of the motor. The results presented in the thesis indicate that the proposed schemes are able to track the speed under load variations. The effectiveness and superiority of the fourth method is further demonstrated under vector control conditions in the presence of an inverter. This method has also been experimentally verified.

A novel strategy for control of induction motors using just one off-line trained ANN is also presented. The control strategy employs the magnitude and frequency of the d-q axis quantities to simplify the off-line training of the ANN and allow the ANN to mimic a vector controller. This scheme has the added benefit that subsequent to off-line training, the ANN can be on-line trained for improved performance and robustness, though it can function well without any on-line training also. Simulation results show that after off-line training the ANN is able to run the induction motor for various changes in speed reference and load torque, and the network is able to generalize effectively. Further simulation results are presented to show the robustness of the control strategy under induction motor parameter variation when the ANN controller is functioning under on-line control. An off-line trained ANN is particularly useful for real-time implementation, because of the reduced computational burden.

Though the problem of obtaining a robust and sensorless induction motor drive is by no means completely solved, the results obtained as part of this work point in a promising direction.

Acknowledgements

I express my sincerest gratitude and appreciation to my supervisors Dr. J.E. Quaicoe and Dr. R. Venkatesan, for their guidance, advice, encouragement and thoughtfulness throughout this program. I also acknowledge the assistance from Dr. M. Hinchey in my supervisory committee, for his useful comments and suggestions. I extend my appreciation to the School of Graduate Studies for their support in this program. I am also grateful to the Dean and associate Dean, Faculty of Engineering and Applied Science, for their help in my program.

I sincerely acknowledge the assistance received from Mr. David Press, Mr. Tony Galway, Mr. Scott Squires, Mr. Philip van Ulden, Ms. Valerie Fortier and Mr. Tom Pike of the C-CAE. I also wish to thank Mr. Richard Newman for his continued assistance with my experimental work and Mr. Don Guy, Mr. Dennis Johnson and Ms. Moya Crocker for their help and support. I express my gratitude to Dr. B. Jeyasurya, Dr. C. Moloney, Dr. P. Artiss and Dr. M. Collins for their help. Thanks are also due to Mrs. Lilian Beresford, international student advisor at MUN for her assistance.

I thank the following friends, who helped me out at some point or the other, and made my stay here a lot pleasanter than it would have otherwise been: V. Adluri, B. Balasubramanian, S. Bellini, Y. El-Sayed, H. Khandwala, M. Krész, A. Petersons, Y.V.S.N. Prasad, I. Rada, R. Ramjuttan, R. Roy, S. Sharan, H. Sivakumar, K. Subramaniam, P. Timko, F. Zahra and A. M. Zeiner. Finally, I thank my parents and my sister for their continued encouragement and support, and I would like to dedicate this thesis to them.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

B	Damping coefficient (All electrical machines)
e_{ids}	error between estimated value and measured value of the direct axis stator current
e_{iqs}	error between estimated value and measured value of the quadrature axis stator current
$e_{\omega r}$	induction motor speed error (reference speed minus actual speed)
i_a	Armature current (DC machine)
i_{ds}	direct-axis stator current in the stationary reference frame (AC machines)
i_{dr}	direct-axis rotor current in the stationary reference frame (AC machines)
i^e_{dr}	direct-axis rotor current in the rotor flux oriented reference frame
i^e_{ds}	direct-axis stator current in the rotor flux oriented reference frame
i_f	field current (DC machine)
i_{mr}	magnitude of the space phasor of the rotor magnetizing currents expressed in the rotor flux oriented reference frame
i_{qr}	quadrature-axis rotor current in the stationary reference frame (AC machines)
i^e_{qr}	quadrature-axis rotor current in the rotor flux oriented reference frame
i_{qs}	quadrature-axis stator current in the stationary reference frame (AC machines)
i^e_{qs}	quadrature-axis stator current in the rotor flux oriented reference frame

$\bar{i}_r$	space phasor of the rotor current expressed in the stationary reference frame
$\bar{i}_r^e$	space phasor of the rotor current expressed in the rotor flux oriented reference frame
$\bar{i}_s$	space phasor of the stator current expressed in the stationary reference frame
$\bar{i}_s^e$	space phasor of the stator current expressed in the rotor flux oriented reference frame
J	Moment of Inertia (All electrical machines)
$\mathcal{J}$	cost function for ANN training
k_e	back EMF constant (DC machine)
k_f	flux constant (DC machine)
k_t	torque constant (DC machine)
L_{lr}	Rotor leakage inductance (AC machines)
L_{ls}	Stator leakage inductance (AC machines)
L_m	Magnetizing inductance (AC machines)
L_r	Rotor self inductance (AC machines)
L_s	Stator self inductance (AC machines)
p	differentiation operator
P	Number of pole pairs (All electrical machines)
R_r	Rotor phase winding resistance (AC machines)
R_s	Stator phase winding resistance (AC machines)
T	Time step for simulation
T_{em}	Electromagnetic torque (All electrical machines)
T_l	Load torque (All electrical machines)
v_a	Armature voltage (DC machine)
v_{dr}^e	direct-axis rotor voltage in the rotor flux oriented reference frame

v_{ds}	direct-axis stator voltage in the stationary reference frame (AC machines)
v_{ds}^{e}	direct-axis stator voltage in the rotor flux oriented reference frame
v_{dx}	$v_{ds} - R_s i_{ds}$
v_{qr}^{e}	quadrature-axis rotor voltage in the rotor flux oriented reference frame
v_{qs}	quadrature-axis stator voltage in the stationary reference frame (AC machines)
v_{qs}^{e}	quadrature-axis stator voltage in the rotor flux oriented reference frame
v_{qx}	$v_{qs} - R_s i_{qs}$
$\overline{\mathbf{v}}_s$	space phasor of the stator voltage expressed in the stationary reference frame
$\overline{\mathbf{v}}_s^{e}$	space phasor of the stator voltage expressed in the rotor flux oriented reference frame
$w_{ji}(n)$	synaptic weight for neuron j in one layer to neuron i in the previous layer at the nt
$y_i(n)$	Output of neuron i at the nth iteration
z^{-1}	Delay operator
α	ANN momentum parameter for training the nth iteration
β	slope of the neuron activation function
$\delta_j(n)$	local gradient of neuron j at the nth iteration
$\Delta w_{ji}(n)$	increment for synaptic weight from neuron j in one layer to neuron i in the previous layer at the nth iteration
η	ANN learning rate
θ_e	Angle of the rotor flux vector referred to the stationary frame of reference
σ_ω	uncertainty in the rotor speed measurement
σ_i	uncertainty in the stator current measurement
$\Phi()$	neuron activation function
ψ_{dr}	direct-axis rotor flux linkage in the stationary reference frame (AC machines)
ψ_{dr}^{e}	direct-axis rotor flux linkage in the rotor flux oriented reference frame

ψ_{ds}	direct-axis stator flux linkage in the stationary reference frame (AC machines)
ψ^e_{ds}	direct-axis stator flux linkage in the rotor flux oriented reference frame
ψ_f	field flux (DC machine)
ψ_{qr}	quadrature-axis rotor flux linkage in the stationary reference frame (AC machines)
ψ^e_{qr}	quadrature-axis rotor flux linkage in the rotor flux oriented reference frame
ψ_{qs}	quadrature-axis stator flux linkage in the stationary reference frame (AC machines)
ψ^e_{qs}	quadrature-axis stator flux linkage in the rotor flux oriented reference frame
$\overline{\psi}_r$	space phasor of the rotor flux linkage expressed in the stationary reference frame
$\overline{\psi}^e_r$	space phasor of the rotor flux linkage expressed in the rotor flux oriented reference frame
$\overline{\psi}^e_s$	space phasor of the stator flux linkage expressed in the rotor flux oriented reference frame
$\overline{\psi}_s$	space phasor of the stator flux linkage expressed in the stationary reference frame
ω_c	Filter cut-off frequency in radians/sec
ω_s	Synchronous speed in electrical radians/sec (AC machines)
ω_r	Motor speed in electrical radians/sec (All electrical machines)
2DOF	Two Degree of Freedom
AC	Alternating Current
ADC	Analog to Digital Converter
AGP	Accelerated Graphics Port
ANN	Artificial Neural Networks
CNN	Cellular Neural Network
CPS	Connections Per Second
CUPS	Connection Updates Per Second
DAC	Data Acquisition System

DC	Direct Current
DSP	Digital Signal Processor
DTC	Direct Torque Control
DVC	Direct Vector Control
EKF	Extended Kalman Filter
EMF	Electromagnetic Force
EMTP	ElectroMagnetic Transients Program
ETANN	Electrically Trainable Analog Neural Network
EV	Electric Vehicle
FF	Feed Forward network
FFT	Fast Fourier Transform
GCPS	Giga Connections Per Second
GPIB	General Purpose Interface Bus
HP	Horse Power
IGBT	Insulated Gate Bipolar Transistor
IM	Induction Motor
IVC	Indirect Vector Control
LCD	Liquid Crystal Display
LMS	Least Mean Square
LPF	Low Pass Filter
LR	Learning Rate
MCPS	Mega Connections Per Second
MCUPS	Mega Connection Updates Per Second
MIMO	Multiple Input Multiple Output
ML	Multi-Layer
MMF	Magneto-Motive Force
MRAC	Model Reference Adaptive Control

MRAS	Model Reference Adaptive System
MS-DOS	MicroSoft Disk Operating System
NARMAX	Non-linear AutoRegressive Moving Average with eXogenous inputs
PC	Personal Computer
PI	Proportional Integral
PLC	Programmable Logic Controller
PMSM	Permanent Magnet Synchronous Machine
PNN	Probabilistic Neural Network
PWM	Pulse Width Modulated
RCE	Restricted Coulomb Energy
RISC	Reduced Instruction Set Computing
RWC	Random Weight Change
SIMD	Single Instruction-stream Multiple Data-stream
SISO	Single Input Single Output
SSE	Sum Squared Error
TDL	Tapped Delay Line
THD	Total Harmonic Distortion
VSI	Voltage Source Inverter

Chapter 1

Introduction

The history of adjustable speed motor drives is quite old and goes back to the 19th century. Earlier, this area was dominated by DC machines, and AC machines were relegated to constant speed operations. This was primarily due to the fact that with a *separately excited* DC machine, independent control of the flux and torque can be achieved, and all quantities are DC, resulting in a simpler control strategy. However, the slip-ring and brushes arrangement results in a lot of wear and tear. Also, due to sparking at the brushes, these machines cannot be used in mines and other potentially hazardous areas because of the risk of explosion.

After 1970, adjustable speed AC drive technology gained a lot of momentum, and it was found that AC motor drives accounted for more than 50% of all the energy consumed in developed countries. AC machines are now replacing DC machines even in high performance applications. It is predicted that in the future, the omnipresent internal combustion engine will be replaced by AC machines, leading to maintenance and pollution free automobiles.

AC machines can be further classified as two main types — synchronous machines and induction machines. Synchronous machines are a very important class of electric machines. Their forte has been the area of power generation, because of which they are known as synchronous generators or alternators. Synchronous machines run only at synchronous speed, i.e. the speed of rotation of the air gap flux vector. The field winding of synchronous machines is on the rotor and carries DC current, which is supplied through an arrangement of commutators and brushes not unlike DC machines. These machines, thus, have the same drawbacks as DC machines. The electrically excited rotor can also be replaced by a permanent magnet. This type of machine is called the permanent magnet synchronous machine (PMSM). This offers many advantages like elimination of rotor copper losses and brushes, leading to increased efficiency. However, because a permanent magnet is used, the airgap cannot be considered uniform [1]. Thus, it is difficult to obtain smooth torque and a servo like performance from these machines. Also, the use of a permanent magnet rules out flux control, making it difficult to operate the drive in the constant power region. The PMSM is usually expensive because of the expensive permanent magnet material and has saturation problems at the teeth because rotor flux is non-uniform.

Induction machines do not have many of the problems associated with synchronous machines, and are widely used in the industry. The machine of choice for this thesis work was the squirrel-cage induction machine and it will be discussed in more detail in section 1.4.

1.1 Motion control — issues involved and the future

The field of motion control stemmed from the area of power electronics initially, but as Harashima [2] points out, it has acquired a separate status of its own. The early stages of research in this area, in the 1970s, focussed on the application of basic control theory to drive a motor, using a power electronic converter. As the field evolved, it drew in researchers who originally trained in various other fields, including control engineering, computer science, mechanical engineering, artificial intelligence (including fuzzy logic and artificial neural networks) and electrical machines and power systems. According to Harashima, as the situation stands today, a traditionally trained Power Electronics engineer would no longer be suitable for the field of motion control, since the field is so interdisciplinary and requires experts from various fields pooling their efforts.

1.1.1 Applications of motion control

Motion control encompasses every technology related to the movement of objects. It covers every motion system from micro-sized systems such as silicon-type micro induction actuators to macro-sized systems such as a space platform. In consumer and commercial applications, motion controllers are found in vacuum cleaners, washers and dryers, music systems, VCRs, computer peripherals like printers, plotters and disk drives, ceiling and portable fans, mixers and blenders, drills, elevators and escalators, and various other products. In the industrial setting, motion controllers are ubiquitous, and can be found in pumps, compressors, machine tools, rolling mills, PLCs and other applications. Control of robotic manipulators is also included in the field of motion control because most of the robotic manipulators are driven by electrical servo

motors and the key objective is the control of motion. Transportation offers another area rife with motion controllers. They are found in electric vehicles including electric trains, street cars, trolleys, ships and steamers, airplanes and space shuttles. Motion controllers are also widespread in military applications like missile launcher guidance and radar control.

As can be seen from the above, motion controller applications can be found in virtually every aspect of modern life. Thus, a large number of researchers are involved with the research and development of motion controllers.

1.1.2 Future trends

There are two central issues and problems in motion control. One is to make the resulting system of controller and plant robust against parameter variations and disturbances. The other is to make the system intelligent, i.e., to make the system self-adjusting to changes in environment and system parameters. Various methods used in control system theory have been applied to improve the robustness of the system. One famous method is the H-∞ method based on frequency-domain optimization [3]. Sliding-mode control is also effective for robust control [4]. It has switching inputs and is thus attractive for power electronics engineers.

To make a motion control system intelligent requires more time and research efforts because our knowledge and computational resources are not yet sufficient for realization. Soft computational methods such as artificial neural networks (ANNs) and fuzzy logic are familiar intelligent control methods, and they have been applied by many researchers working in the drives area. However, a lot of work remains to be done, not only in the drives area but also in maturing the base technologies themselves.

For induction motor drives in particular, another issue has gained importance in recent years and that stems from problems associated with the speed sensor. Most closed loop control schemes require a speed sensor, and this is usually expensive and cannot be used in some situations. Thus, many researchers have focussed on ways to avoid the use of a speed sensor, but many of these schemes have other problems not associated with speed sensor-based control.

In the future, it is expected that computational power and memory will get still cheaper leading to realization of more powerful control techniques. It will be possible to make a totally automatic control system which will derive a mathematical model of the plant by providing some test signals at the input, or obviate the need for a plant model by using an intelligent control technique like ANNs. It will be possible to identify the required parameters, decide on the control strategy and self-commission the drive. For induction motor drives, it will be possible to have an accurate estimate of the speed without using a speed sensor. Motion control will also have to deal with nonlinear systems, and this would require further maturity of base technologies like nonlinear adaptive control theory, fuzzy logic and ANNs.

1.2 Motivation for this work

Newer motion control applications like electric vehicles demand a high torque-to-size ratio machine, and a powerful but inexpensive controller. The squirrel-cage induction motor satisfies the first requirement admirably, and has been the motor of choice for a number of electric vehicle applications. For example, the EV1 electric vehicle, which is made by General Motors and is commercially available, uses an induction motor drive [5]. Awareness about the harmful effects of automobile pollution on the environment is on the rise, and it is inevitable that there will be more research on efficient and

compact induction motor controllers for electric vehicles and other environmentally-friendly applications like harnessing alternate-energy sources.

It was seen in subsection 1.1.2 that intelligent control techniques will become more commonplace in the future. ANN technology spawned off from research on the functioning of the brain and has grown by leaps and bounds in recent years. It has some important features like large scale parallelism for both computation and information storage. With the advent of ANN integrated circuits, ANN applications should run in a fraction of the time needed by conventional sequential processors. Also, partial malfunctioning of an ANN controller would not cause the controller to stop functioning, because of the inherent robustness of the ANN architecture. At present, dedicated ANN hardware is a very expensive proposition, but that is chiefly because of lack of demand. With higher demand, and production in larger volumes, the price of ANN hardware is bound to come down. Thus, it would be worthwhile to investigate ANN techniques for induction motor drives to establish a theoretical foundation in this area. This might lead to more effective ANN based induction motor drives, thereby furthering a demand in ANN hardware.

1.3 Objectives of the proposed research

The first objective of this research is to estimate induction motor speed using ANNs. The next objective is to come up with a strategy for controlling an induction motor using only an ANN controller. Both of these are unreported in the literature, and even though this itself is quite ambitious, it was decided that experimental verification of the ANN based schemes should be a third objective of this work. Achieving these objectives should pave the way for the development of an ANN based sensorless induction motor drive, which is the ultimate goal of many researchers working in this field.

With these objectives in mind, the next two sections provide a brief discussion on induction machines and ANNs respectively, and the rest of the thesis outlines the literature review undertaken, and the work done, in an attempt to achieve these objectives.

1.4 Induction machines

Induction machines have been the workhorse of the industry. These machines enjoy various advantages like simplicity, ruggedness, low cost, reliability and compactness. The squirrel-cage induction machine, in particular, has been used in constant speed drives for more than a century now. The principle of operation of the induction machine is similar to that of the transformer. The induction motor always runs at a speed slightly less than the synchronous speed, because there is no torque production in an induction machine at synchronous speed. The slip ring induction machine has access to the rotor terminals, but does away with some of the benefits of the squirrel-cage induction machine. The basic problem with the induction machine is the difficulty of control. The induction machine is a highly coupled, non-linear dynamic plant and its applications in the area of speed control have traditionally been limited. The complex and expensive techniques such as *pole changing*, *Scherbius* and *Kramer* schemes have been known for a long time but are not generally favoured for high performance applications.

The d-q axis induction machine dynamic equations in the stationary reference frame

are given below [1]

$$\begin{bmatrix} v_{ds} \\ v_{qs} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} R_s + pL_s & 0 & pL_m & 0 \\ 0 & R_s + pL_s & 0 & pL_m \\ pL_m & \omega_r L_m & R_r + pL_r & \omega_r L_r \\ -\omega_r L_m & pL_m & -\omega_r L_r & R_r + pL_r \end{bmatrix} \begin{bmatrix} i_{ds} \\ i_{qs} \\ i_{dr} \\ i_{qr} \end{bmatrix} \tag{1.1}$$

where v_{ds}, v_{qs} are the direct and quadrature (d-q) axis components respectively of the applied stator voltage, i_{ds}, i_{qs} refer to the d-q axis stator currents, i_{dr}, i_{qr} are the d-q axis rotor currents, ω_r is the motor speed in electrical radians per second, R_s, R_r refer to the stator and rotor resistance, L_s, L_r refer to the stator and rotor inductance, L_m is the magnetizing inductance and P is the number of pole pairs.

$$T_{em} = -\frac{3}{2} P L_m (i_{ds} i_{qr} - i_{qs} i_{dr}) \tag{1.2}$$

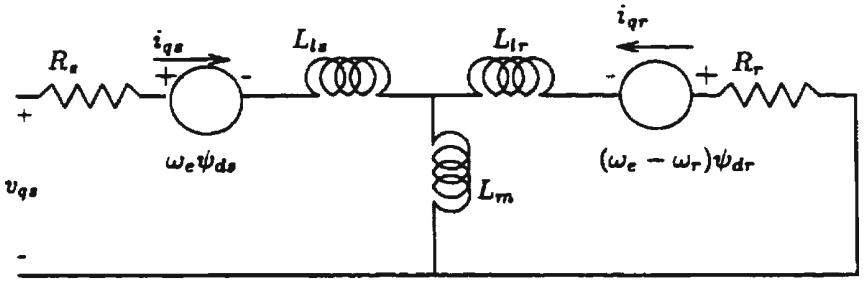
$$T_{em} = T_l + J\frac{d\omega_r}{dt} + B\omega_r \tag{1.3}$$

where T_{em} is the electromagnetic torque, T_l is the load torque, J is the moment of inertia and B is the damping coefficient.

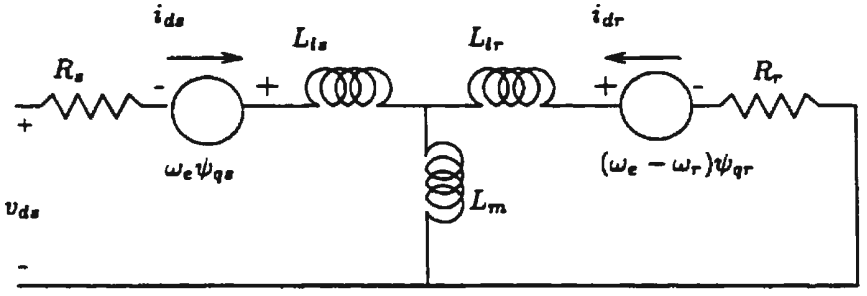
A $d-q$ axis equivalent circuit model of the induction machine in the synchronous reference frame is shown in Figure 1.1. In this figure, L_{ls} and L_{lr} refer to the stator and rotor leakage inductance, ψ_{ds} and ψ_{qs} are the d-q axis stator and rotor flux components and ω_e refers to the synchronous speed.

1.5 Artificial neural networks

Artificial neural networks have emerged as powerful problem solving tools in the areas of pattern recognition and function emulation. There are various types of ANNs,



q-Axis Circuit



d-Axis Circuit

Figure 1.1: d-q axis equivalent circuits for induction machine in the synchronous reference frame

though they can be broadly classified into three main categories — *Feedforward, Feedback and Self-organizing.* The most widely used architectures can be further classified within these three categories, as can be seen in figure 1.2 [6]. The most common type

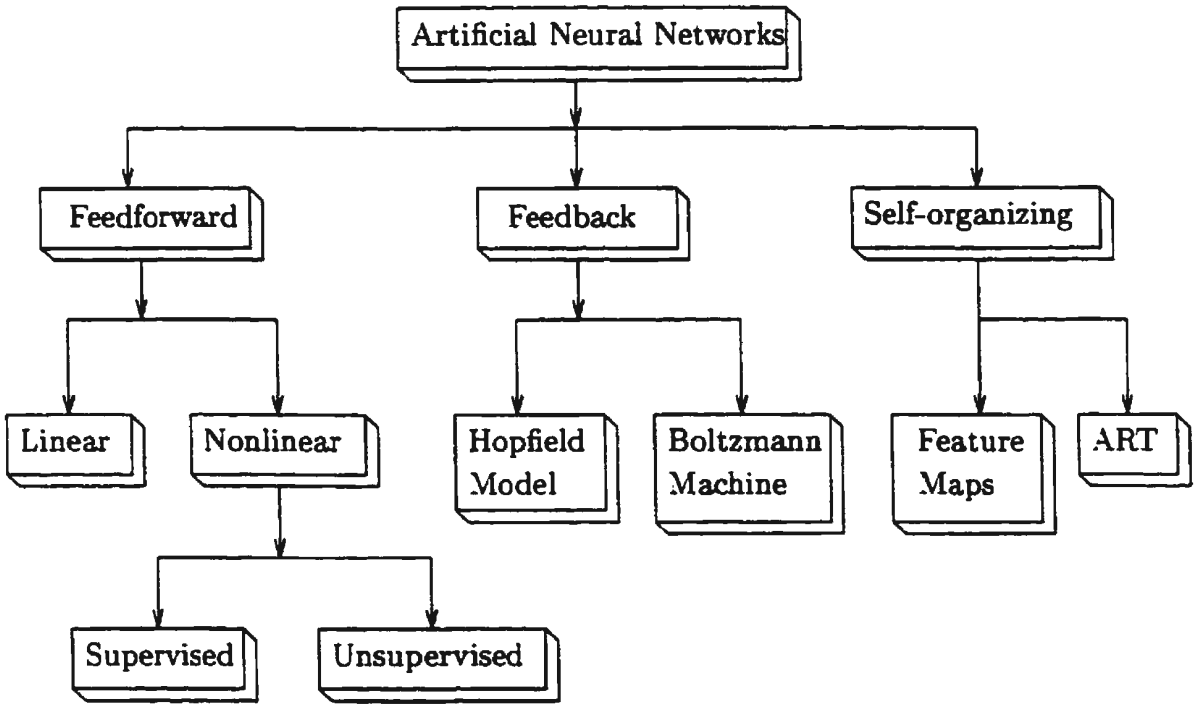


Figure 1.2: Artificial neural network architectures

is known as the multilayered feedforward network. The basic structure of a multilayered feedforward network is shown in Figure 1.3. Feedforward networks are so named because the output of each layer feeds the next layer of units. The *Perceptron*, proposed by Rosenblatt in 1962 and *Adaline* proposed by Widrow in the same year are the earliest feedforward ANN architectures [7]. These ANNs consist of two basic parts — a typically non-linear processing element called the *neuron*, and a connection element called the *synapse* which connects various neurons. Each synapse has a number associated with it, called the *synaptic weight.* All the knowledge in the ANN is stored

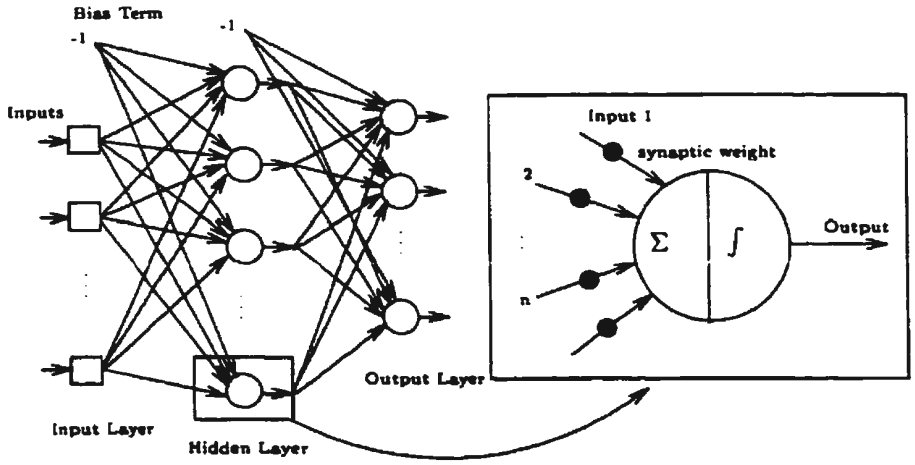


Figure 1.3: Schematic of an artificial neural network

in these weights, also known as the *free parameters* of the network.

The neurons are laid out in layers. The first layer is called the *input layer* and acts as the sensory organ for the ANN. The various inputs of the ANN are received through this layer. The last layer of the ANN is called the *output layer* and supplies the ANN output. All the intermediate layers are called *hidden layers*, because their inputs and outputs are not readily accessible to the external world. The inputs to the hidden layer or output layer neurons are the outputs of the previous layer neurons multiplied by the synaptic weights. The structure of a feedforward network is generally denoted by a set of numbers representing the number of inputs, neurons or outputs in each layer. For example, the structure of a four layered network with 10 inputs, 20 neurons in the first hidden layer, 15 neurons in the second hidden layer, and 2 outputs would be denoted by 10-20-15-2.

The process of training an ANN is defined as *learning* and several algorithms for

ANN learning are available in the literature [8]. However, one of the most popular algorithms is known as the *error backpropagation algorithm*. As the name implies, this algorithm modifies the weights of the networks by propagating the errors at the output backwards through the network. The training data is presented to the ANN one data-vector at a time, and this is referred to as an *iteration*. The presentation of the whole data set to the ANN is referred to as as *epoch*. The weights are modified at each iteration as [8]

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \tag{1.4}$$

where $w_{ji}(n)$ represents the synaptic weight from neuron j in one layer to neuron i in the previous layer at the nth iteration, $\Delta w_{ji}(n)$ stands for the weight increment, $\delta_j(n)$ is the *local gradient* of the neuron j at the nth iteration, $y_i(n)$ is the output of neuron i at the nth iteration, α is called the *momentum* parameter and η is called the *learning rate*. This equation is known as the *generalized delta rule*. There are no fixed rules for choosing the training parameters (α and η). A high value of η makes the ANN converge faster but might lead to instabilities, or might miss the optimum set of weights required for effective training. A low value of η makes learning slower, but is more stable. However, the training algorithm might get trapped in a *local minima* of the multidimensional weight surface and still not reach an optimum set of weights. It is good to try out different learning rates for training. The momentum parameter, α, is less than unity and is usually not too small. It has a smaller effect on network training.

The neuron in any layer computes the weighted sum of the inputs and passes this sum through a non-linear function called the *activation function*. Usually the *sigmoid function* is used as the activation function, because it has many desirable properties — it is non-linear and differentiable and its output is limited in an asymptotic fashion.

This function is given by

$$f(x) = \frac{[1 - e^{-\beta x}]}{[1 + e^{-\beta x}]} \tag{1.5}$$

where β represents the slope of the activation function. There is another term associated with this sum, and this is called the *bias term*. This term can be represented as a constant input of -1 multiplied by the synaptic weight, for each of the layers other than the input layer, as shown in Figure 1.3.

In summary, an ANN is a parallel distributed information processing structure with the following characteristics [9]:

- It is a neurally inspired mathematical model.
- It consists of a large number of highly interconnected processing elements.
- Its connections (weights) hold the knowledge.
- A processing element can dynamically respond to its input stimulus, and the response completely depends on its local information; that is, the input signals arrive at the processing elements via impinging connections and connection weights.
- It has the ability to learn, recall, and generalize from training data by assigning or adjusting the connection weights.
- Its collective behaviour demonstrates the computational power, and no single neuron carries specific information (*distributed representation* property).

ANNs have been used successfully in various areas including image processing and recognition, control systems, speech processing, optimization, communication, signal classification, robotics, power systems and many others [9]. ANNs have the ability to

approximate almost any continuous nonlinear function, and this feature is extremely useful in applications where the functional relationship between the inputs and outputs is very complex or unknown. Image and speech processing with ANNs has received widespread attention and commercial products are now available which use this technology. More recently, ANNs have been applied to problems in control systems, though the field has yet to reach maturity.

1.6 Outline of the thesis

A review of present day research in the drives area and a critique of some work which has been done is presented in chapter 2. Chapter 3 discusses some existing commercial simulators and their main features and limitations. It also describes the design of an object-oriented simulator that was built for the purpose of doing this research. Chapter 4 deals with speed estimation of induction motors using ANNs, and four schemes for the same have been proposed in this chapter. Chapter 5 discusses some issues involved with induction motor control using ANNs and proposes a scheme for controlling an induction motor using an off-line trained ANN. A scheme is also proposed for improving the steady state performance and robustness of this ANN by on-line training. Chapter 6 introduces some commercially available ANN hardware, and then describes an experimental setup which was built for the purpose of verifying the ANN speed estimator developed in chapter 4. Experimental results are then presented and compared with simulation results. Chapter 7 concludes the thesis by highlighting the contributions made by this work and also outlines some avenues for further research in this area.

Chapter 2

Review of Literature

The previous chapter introduced the area of motor drives and outlined the motivation and objectives of doing the present research. It also identified the squirrel-cage induction motor as the motor of choice for this work. The motor drives incorporating this motor have undergone a major change over the last twenty years or so. This chapter traces the evolution of the squirrel-cage induction motor drive and, also highlights the directions taken by recent researchers in this area.

2.1 Scalar control techniques

A simple, economical, but low performance control method of the induction motor that is extremely popular in industry is the open-loop V/f control [10]. A small drift in speed and airgap flux due to fluctuations in load torque and supply voltage, respectively, as well as sluggish transient response, are some of the problems associated with this scheme. However, they are of no consequence in a majority of industrial applications.

An improvement over the basic *V/f* scheme is presented by Koga et al [11]. This method extends the applicability of a *V/f* controlled induction motor drive system by reducing the steady state speed error, caused by load changes from no-load, to zero without using a rotor speed sensor. In another technique called the *loss minimization technique* [12], an effort is made to operate an inverter fed induction motor drive at the point of maximum efficiency at any torque-speed operating point. This method is an improvement over the *V/f* method and falls under the closed-loop category of control techniques. In a few other techniques, advanced concepts from control systems engineering have been applied to induction motor control. A robust speed control method using a load torque observer and feedforward control is presented by Iwasaki et al [13]. Model Reference Adaptive Control (MRAC) and Sliding Mode control methods are used by Alonge [14]. This scheme uses two controllers — one for the machine and one for the inverter. The machine controller is further split into three subcontrollers namely the speed, the rotor flux and the stator current vector subcontrollers. Such control techniques overcome some of the problems of conventional scalar techniques like lack of robustness and high susceptibility to load torque disturbance. However, they are more complex than the simpler techniques mentioned earlier, and usually require a Digital Signal Processor (DSP) for their implementation. Krein et al [15] have done a comparative analysis of induction motor control methods and come to the conclusion that, on an axis representing control complexity, there is a galaxy of possible control methods.

2.2 Field-oriented control

The first major breakthrough in the area of induction motor drives came with the discovery of the concept of *field orientation* by Blaschke [16] in 1972. Blaschke examined

how *field orientation* occurs naturally in a separately excited DC motor. The armature flux and the field flux are always perpendicular to each other due to the effect of the compensating winding. In an induction machine, a similar condition can be created in the rotating frame of reference, by controlling the stator currents in a particular fashion.

The basic idea of field orientation is given mathematical rigor by the use of space phasors. Field orientation is performed by decoupling the stator current space phasor along one of rotor flux space phasor, stator flux space phasor and air-gap flux space phasor. Figure 2.1 shows how the stator current space phasor can be decoupled along the rotor flux space phasor. The $d-q$ axis model of the induction motor with the

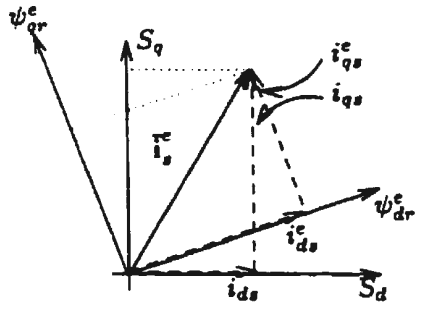


Figure 2.1: Transformation of stator current space phasor

reference axes rotating at synchronous speed ω_e is given by [17]

$$\bar{v}^e_s = R_s\bar{i}^e_s + p\bar{\psi}^e_s + j\omega_e\bar{\psi}^e_s \tag{2.1}$$

$$0 = R_r\bar{i}^e_r + p\bar{\psi}^e_r + j(\omega_e - \omega_r)\bar{\psi}^e_r \tag{2.2}$$

$$T_{em} = \frac{3P}{2}\frac{L_m}{L_r}(\psi^e_{dr}i^e_{qs} - \psi^e_{qr}i^e_{ds}) \tag{2.3}$$

where

$$\bar{v}^e_s = v^e_{ds} + jv^e_{qs} \tag{2.4}$$

$$\overline{i}_s^e = i_{ds}^e + j i_{qs}^e \tag{2.5}$$

$$\overline{i}_r^e = i_{dr}^e + j i_{qr}^e \tag{2.6}$$

$$\overline{\psi}_s^e = \psi_{ds}^e + j \psi_{qs}^e \tag{2.7}$$

$$\overline{\psi}_r^e = \psi_{dr}^e + j \psi_{qr}^e \tag{2.8}$$

In the above, ψ refers to the flux and the superscript e implies that the quantity is expressed in the rotor flux-oriented reference frame, the q and d in the subscript stand for the quandrature and direct axes respectively and the s and r in the subscript stand for stator and rotor quantities respectively.

The field orientation concept implies that the current components supplied to the machine should be oriented in phase (flux component) and in quadrature (torque component) to the rotor flux vector $\overline{\psi}_r^e$. This can be accomplished by choosing ω_e to be the instantaneous speed of $\overline{\psi}_r^e$ and locking the phase of the reference system such that the rotor flux is entirely in the d-axis (flux axis), resulting in the mathematical constraint

$$\psi_{qr}^e = 0 \tag{2.9}$$

It should be noted that control is performed on the DC quantities obtained in the synchronous frame and depending on which flux phasor is chosen for decoupling, we get *rotor field-oriented control, stator field-oriented control* and *magnetizing field-oriented control* respectively. Field-oriented control, also called *vector control*, can be classified in a different way as *indirect field-oriented control* or *direct field-oriented control*. In the first method, the flux is indirectly estimated while in the latter scheme the flux is obtained by direct measurement or explicit computation using stator quantities. With the evolution of fast microprocessors and Digital Signal Processors (DSPs), implementation of vector control in real-time has been achieved and both direct and indirect vector controlled induction motor drives have been developed [18]. A block diagram of a rotor field-oriented induction motor drive is shown in Fig. 2.2. All the quantities

with the superscript $*$ refer to the reference (or command) values.

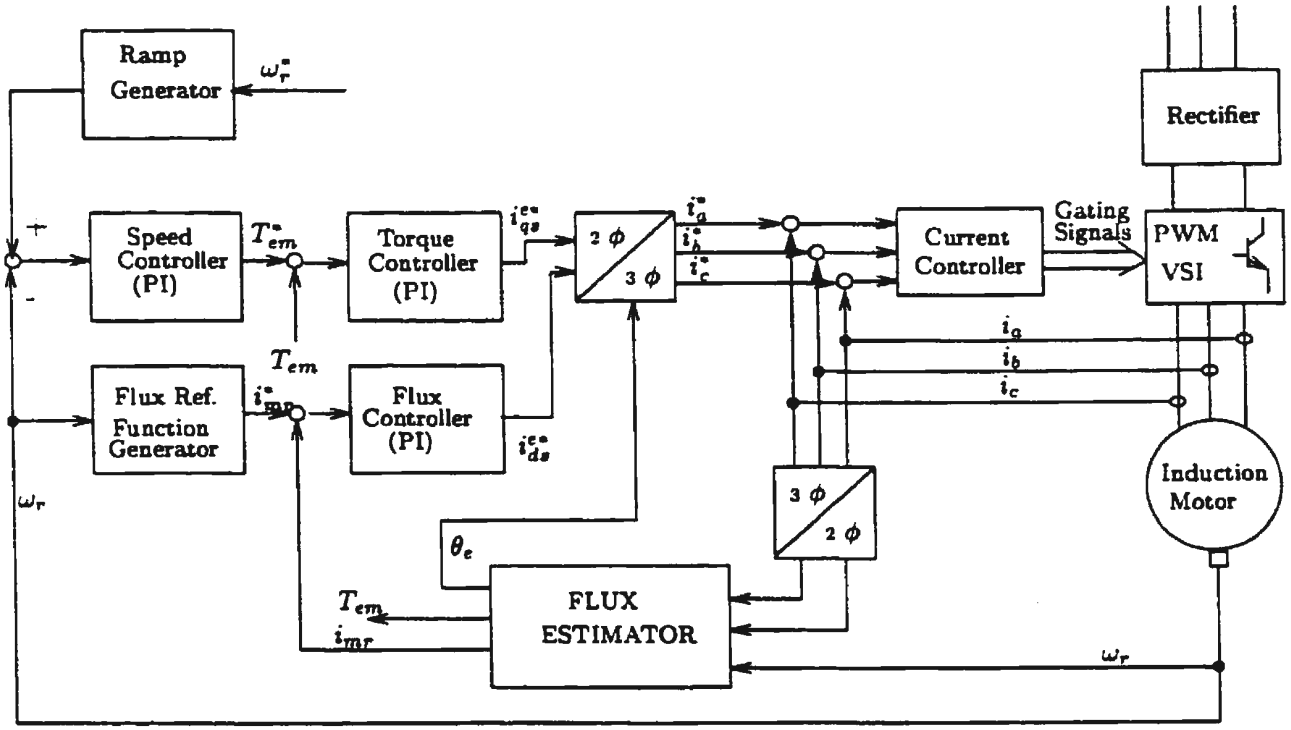


Figure 2.2: Rotor field-oriented induction motor drive

Indirect Vector Control (IVC) suffers from parameter variation problems, but can function quite accurately in the zero speed region. Direct Vector Control (DVC) does not suffer so much from parameter variation, but computation of flux around zero speed is error prone. An attempt to combine the benefits of both strategies is made in [19]. In DVC, flux is explicitly computed, but direct integration is avoided by use of cascaded Low Pass Filters (LPFs) with programmable time constants. Integration is avoided because the gain becomes too high for low frequency signals. Here, the speed is computed using stator quantities under stator flux-oriented control. Stator resistance is the main parameter which undergoes variation. It is compensated by sensing the

temperature and using the stator winding's temperature coefficient to compute the new value of the resistance. The drive is operated in two modes. At startup it operates in IVC. When the torque current exceeds a threshold, the drive is transitioned to DVC mode. It is brought back to IVC mode when the synchronous speed approaches zero.

Field-oriented control has gained widespread acceptance as a high-performance control strategy for induction motor drives. However, as mentioned earlier, it has problems associated with parameter variation of the motor. This leads to lack of robustness and poorer performance, and has prompted researchers to investigate intelligent control techniques like *fuzzy logic* and ANNs.

2.3 Speed sensorless drives

The field-oriented schemes provided very good dynamic response compared to the V/f method and other control schemes. However, most such schemes used speed sensors for closed loop speed control and also for the estimation of the rotor flux vector in the indirect field-oriented control scheme. Unfortunately, speed sensors cannot be mounted in some cases, such as motor drives in hostile environments and high speed motor drives. Also, speed sensors are expensive and reduce the advantage of an induction motor drive system. They lower the reliability of the system, especially in defective environments and require special attention to noise. Because of these problems, researchers put in a lot of effort towards developing induction motor vector controlled drives which required no speed sensor, popularly known as *sensorless drives* [20].

Pioneering work in the area of sensorless vector control of induction motor was done by Ohtani, Takada and Tanaka [21]. They consider the induction motor equations in a frame of reference which rotates at synchronous speed and derive the conditions for field

orientation by setting the quadrature-axis rotor flux to zero. The slip speed is computed by assuming conditions for field orientation, and the motor speed is computed by subtracting slip speed from the synchronous speed. In such a system, the flux has to be obtained from the stator quantities because indirect estimation is not possible without speed feedback. The d-q axis fluxes in the stationary frame can be obtained by direct integration, but this method leads to instability near zero speed. Also, the computed flux depends on motor parameters and a variation in these causes incorrect estimation of the flux. The standard method of using LPFs is also undesirable because it produces phase shifts. The authors propose a method in which an extra LPF is used with the flux command as the input. This approximately compensates the phase shift problem introduced by the first LPF.

As mentioned in the literature, the main types of speed sensorless control schemes can be classified as [22], [23]

- Direct computation of speed
- Slip frequency-based approach
- Observer based methods
- MRAS/MRAC based methods
- Kalman filter/Extended Kalman filter based methods
- Rotor harmonic detection based methods
- Superimposition of signals on the current command

2.3.1 Direct computation of speed

The direct calculation of speed method derives expression for the rotor speed and rotor resistance using stator voltage feedback. These expressions have a numerator over denominator form where both the numerator and denominator are zero for sinusoidal waveforms. But, actually, the waveforms are non-sinusoidal, and hence speed and resistance can be obtained. The technique loses reliability as load increases. Kanmachi et al [24] have obtained expressions for the induction motor speed and the rotor resistance as shown in equations (2.10), (2.11).

$$\omega_r = \frac{(\psi_{ds} - L_s i_{ds})p\psi_{qr} - (\psi_{qs} - L_s i_{qs})p\psi_{dr}}{(\psi_{ds} - L_s i_{ds})p\psi_{dr} - (\psi_{qs} - L_s i_{qs})p\psi_{qr}} \tag{2.10}$$

$$R_r = \frac{-L_m\psi_{dr}p\psi_{dr} - L_m\psi_{qr}p\psi_{qr}}{(\psi_{ds} - L_s i_{ds})p\psi_{qr} - (\psi_{qs} - L_s i_{qs})p\psi_{dr}} \tag{2.11}$$

In the above equations, ψ and L_m represent flux and mutual inductance respectively, and p stands for the differentiation operator. In both these expressions, the numerator and denominator are identically zero for normal sinusoidal operation. However, the authors claim that, for inverter-fed induction motor drive, the expressions are not identically zero, and thus the speed and rotor resistance can be obtained. The authors have also studied the influence of parameter variations on this scheme. By varying the stator resistance in the simulations, they found that the output of the speed and rotor resistance estimators is erroneous. Furthermore, the speed calculation error increases with an increase in the load torque, whereas the rotor resistance calculation error changes very slightly with a change in load torque.

2.3.2 Slip frequency-based approach

In this approach, the inverter frequency is controlled such that vector control conditions are satisfied [25]. Under these conditions, the slip frequency is given by

$$\hat{\omega}_{\text{slip}} = \frac{R_s}{L_s}\frac{i_{qs}}{i_{ds}} \tag{2.12}$$

The motor speed is then obtained as

$$\hat{\omega}_r = \omega_e - \hat{\omega}_{\text{slip}} \tag{2.13}$$

where the $\hat{}$ indicates estimated quantities.

It should be noted that in this scheme the vector control algorithm and the slip frequency computation are interlinked, and failure of one would result in failure of the other. A block diagram of this scheme is shown in Figure 2.3.

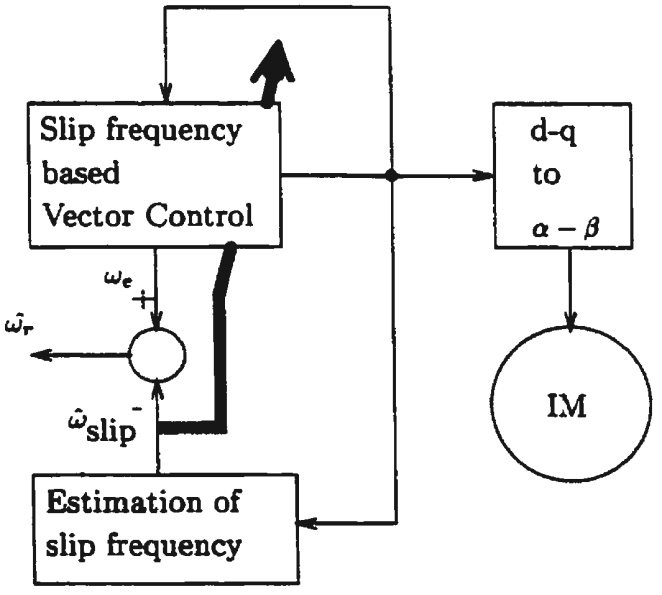


Figure 2.3: Slip frequency-based sensorless control of induction motor

2.3.3 Observer based methods

Observer theory is aimed at providing a real-time estimate of the state of a system, using only the input and output signals, both of which are assumed to be known [26]. The estimate provided by the observer contains a *prediction error* term. The induction machine dynamic equations can be written in state variable form as

$$\dot{x} = \mathbf{A}x + \mathbf{B}\bar{\mathbf{v}}_s \tag{2.14}$$

$$\bar{\mathbf{i}}_s = \mathbf{C}x \tag{2.15}$$

where $\bar{\mathbf{v}}_s$ is the stator voltage, $\bar{\mathbf{i}}_s$ is the stator current phasor, and x is the induction motor state vector given by

$$x = \begin{bmatrix} i_{ds} & i_{qs} & \psi_{dr} & \psi_{qr} \end{bmatrix}^T \tag{2.16}$$

where ψ stands for the flux linkage and the subscripts follow the standard notation. The stator voltage phasor $\bar{\mathbf{v}}_s$ and stator current phasor $\bar{\mathbf{i}}_s$ are given by

$$\bar{\mathbf{i}}_s = i_{ds} + j i_{qs} \tag{2.17}$$

$$\bar{\mathbf{v}}_s = v_{ds} + j v_{qs} \tag{2.18}$$

The state observer, which estimates the stator current and the rotor flux, can be written as [27]

$$\dot{\hat{x}} = \hat{\mathbf{A}}\hat{x} + \mathbf{B}\bar{\mathbf{v}}_s + \mathbf{G}(\hat{\bar{\mathbf{i}}}_s - \bar{\mathbf{i}}_s) \tag{2.19}$$

where $\hat{}$ means estimated values and $\mathbf{G}$ is the observer gain matrix, which is chosen such that equation (2.19) is stable. By utilizing Lyapunov's theorem, induction motor speed can be estimated as

$$\hat{\omega}_r = K_P(e_{ids}\hat{\psi}_{qr} - e_{iqs}\hat{\psi}_{dr}) + K_I \int (e_{ids}\hat{\psi}_{qr} - e_{iqs}\hat{\psi}_{dr})dt \tag{2.20}$$

where e_{ids}, e_{iqs} stand for the error between estimated value and measured value of the direct and quadrature axis stator current respectively, and K_P, K_I are proportional and integral gains. Figure 2.4 shows a block diagram of this scheme.

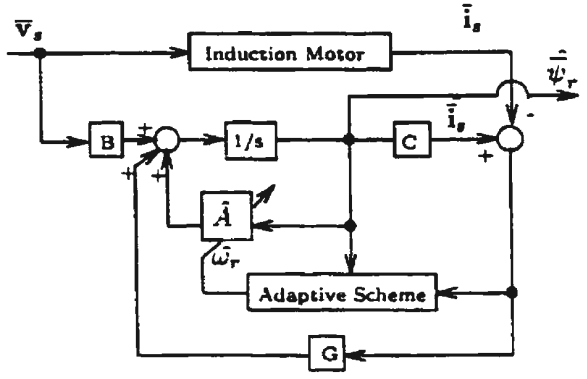


Figure 2.4: Block diagram of observer based schematic

The observer based methods yield a reasonably accurate value for the speed. However, most of these methods use integration techniques which have problems with the initial values and drift. To avoid these problems, the pure integrator is replaced by a low pass filter (LPF) with a high gain. However, this replacement causes instability of identification at low speeds.

2.3.4 Model reference adaptive control based methods

The Model Reference Adaptive System (MRAS) or Model Reference Adaptive Control (MRAC) is a common technique to obtain speed sensorless control and parameter independence. In this technique, there is a *reference model*, an *adjustable model*, and an *adaptation mechanism* [28]. A block diagram of the MRAC scheme is shown in Fig. 2.5. The reference model is a certain quantity which is estimated without using the rotor speed. In the adjustable model, the same quantity is estimated using the rotor speed and other parameters. It is assumed that the difference in the two quantities is chiefly due to the rotor speed variation. The error between the two is used as a

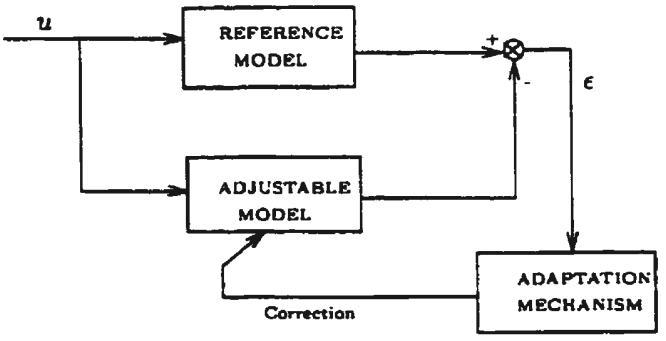


Figure 2.5: Block schematic of model reference adaptive control

correction for the speed. These techniques usually use integrators for computing the flux from the stator quantities. Use of integrators causes various problems like drift in the value and very high gain at a low frequency. Thus, they can only be used in a limited range of the drive operation. To overcome the integration problem, some researchers have used LPFs instead of integrators. These cause a phase shift in the output, as has been mentioned before. One technique has been suggested, which uses back emf and does not require integrators [29]. An extension of the MRAC technique is presented by Zhen and Xu [19]. Here, the reference model and the adjustable models have interchangeable roles. Initially, the models are used to estimate the speed. After the speed becomes constant, the models interchange their roles, and an estimation of the rotor resistance and rotor time constant is effected. It is assumed that the speed does not change during this time.

2.3.5 Extended Kalman filter based techniques

The Kalman filter is an optimal observer for the state space solution of problems where random noise is assumed to be present both in the inputs and the outputs. The

formulation of the state space problem is as follows [30].

$$\dot{x}(t) = Ax(t) + Bu(t) + \omega(t) \tag{2.21}$$

$$y(t) = Cx(t) + \nu(t) \tag{2.22}$$

where $x(t)$ is the state-variable vector, $u(t)$ is the input vector, $y(t)$ is the output vector, $\omega(t)$ is the noise matrix of the state model and $\nu(t)$ is the noise matrix of the output model. It should be noted that the above state space representation is linear. In the dynamic model of the induction motor, however, the dimension of the state vector is increased by adding the angular speed of the rotor. This causes the state model to become non-linear. In such a case, the *extended Kalman filter* has to be used to estimate the desired parameter. The discrete-state model and the output model for the extended Kalman filter case are given by [31]

$$\dot{x}(t) = f[x(t), u(t), t] + G(t)\omega(t) \tag{2.23}$$

$$y(t) = h[x(t), t] + \nu(t) \tag{2.24}$$

where $f[x(t), u(t), t]$ and $h[x(t), t]$ represent the non-linear part of the state model. The extended Kalman filter relinearizes the non-linear state model for each new estimate as it becomes available. The rotor speed can be estimated from the dynamic model of the induction motor by using the extended Kalman filter algorithm in the following steps. i) estimation of error covariance matrix ii) computation of Kalman filter gain iii) update of error covariance matrix and iv) state estimation.

The extended Kalman filter algorithm has a very good performance as regards to noise sensitivity. Also, it is able to function at low speeds. However, the noise covariance matrices and initial values for the algorithm must be chosen very carefully. Otherwise, instability may result with this algorithm. Also, the steady-state error for this algorithm is somewhat high especially at low speeds [23]. Another disadvantage of this scheme is that it is very computation intensive and requires a very fast processor for real-time implementation.

2.3.6 Rotor harmonic detection based methods

Rotor harmonic detection based methods use the fact that speed related harmonics arise from rotor slots and rotor eccentricity. They are independent of time-varying machine parameters and exist at any non-zero speed. But these techniques fail under light load conditions and at very low speeds. They also require some form of user initialization, since they depend on typically unknown parameters such as the number of rotor slots, and they are susceptible to noise. If analog filters are used, the bandwidth gets reduced. FFT based technique is also used, but it relies on a particular slot harmonic, which limits application to different machines. In one technique proposed by Hurst et al [32], the speed related harmonics arising from rotor mechanical and magnetic saliencies, such as rotor slotting and rotor eccentricity, are used to detect the speed. The harmonics arising from the rotor speed depend on the number of rotor slots, the order of rotor eccentricity and the order of the airgap MMF harmonics. However, if only the eccentricity harmonics are considered, an expression for the slip can be obtained with poorer resolution. The authors use the latter fact to run an initialization routine and to determine the other parameters from a limited domain of their typical values. The speed detection technique uses both analog and digital filtering and the application of a Hamming window to determine the rotor speed. This method is, however, computationally very intensive, and so a simpler method is also suggested which uses the mechanical model of the machine and tries to estimate the moment of inertia, the viscous damping and the load torque. The speed can be estimated with a knowledge of these parameters and the electromagnetic torque.

2.3.7 Superimposition of signals on the current command

In most sensorless schemes, the speed estimate obtained depends on the motor parameters, which are subject to variation. Thus, it would be desirable to have a sensorless scheme in which the motor parameters, especially the rotor time constant, can be obtained simultaneously. Ohnishi et al [25] have shown that simultaneous identification of the rotor resistance and motor speed is possible only when the rotor flux is persistently time-variant. Under vector control, the rotor flux is maintained constant so that the orthogonality of rotor flux and rotor current is achieved. Thus, it would be impossible to obtain the speed and rotor time constant simultaneously. Kubota and Matsuse [33] propose a scheme in which the equations of the induction motor are considered in the synchronous reference frame. As shown in the equation below, only the ratio between the slip speed and rotor resistance can be obtained.

$$i^e_{qr} = \frac{-\omega_{slip}}{R_r}\psi^e_{dr} \tag{2.25}$$

If, however, AC components are superimposed on the field current command, the motor speed and rotor resistance can be estimated simultaneously. The frequency of these AC components have to be different from the fundamental frequency of the inverter output. This is not a very popular technique and suffers from the drawback that the flux command, and hence the controlled flux, will have ripples, leading to poorer performance.

Apart from the techniques mentioned above, there are some other techniques, which are not as well known as the above. Some of them are used under special conditions. One such scheme obtains the motor speed from split phase stator windings [34]. This requires a modification in the stator windings of standard induction motor. Another scheme for obtaining the speed using *Direct Self Control* is presented by Baader et al [35]. This is used in medium to low performance drives.

2.4 Disturbance torque and robust motion control

High performance motion control often presents conflicting control requirements. To attain high performance, the control system should be robust against load and parameter changes. In mathematical terms, this implies that if a force F is applied to a system, and x is the change in position in a position control system, or change in speed in a speed control system, then the *control stiffness* is defined as [25]

$$\text{control stiffness} = \left.\frac{\partial F}{\partial x}\right|_{t\to\infty} \tag{2.26}$$

The ideal control system should not allow any stationary and transient deviation for any load. This implies that the control stiffness should be infinite. On the other hand, an ideal control system should be able to adjust instantaneously to the smallest change in the reference, thereby implying that the smallest error signal should be able to drive the system instantaneously towards its new steady state. This, in turn, implies that the control stiffness should be zero. A regular system, of course, has a control stiffness somewhere between the two extremes.

A robust controller, typically used for high performance drives, has to be

- insensitive to the external disturbance
- insensitive to parameter variation

Though these two are different requirements, they can be combined in a a single quantity called the disturbance torque. Using a disturbance torque observer is one way to counter the effects of the disturbance torque. To illustrate the procedure, consider the dynamical equation of motion of the DC machine given by

$$T_{em} = k_t \psi_f i_a = T_l + J\frac{d\omega_r}{dt} + B\omega_r \tag{2.27}$$

where ψ_f is the field flux, i_a is the armature current and k_t is the torque constant. Assume that the inertia J and the torque constant k_t undergo variations. The parameter variation of each from the nominal value gives

$$J = J_n + \Delta J \tag{2.28}$$

$$k_t = k_{tn} + \Delta k_t \tag{2.29}$$

Incorporating these variations, the equation of motion can be written as

$$k_{tn}\psi_f i_a = T_{\text{dis}} + J_n \frac{d\omega_r}{dt} + B\omega_r \tag{2.30}$$

where

$$T_{\text{dis}} = T_l + \Delta J \frac{d\omega_r}{dt} - \Delta k_t \psi_f i_a \tag{2.31}$$

Using equations (2.27) and (2.31) we get

$$T_{\text{dis}} = k_{tn}\psi_f i_a - J_n \frac{d\omega_r}{dt} \tag{2.32}$$

and this equation can be used to estimate the disturbance torque as shown in Figure 2.6. The estimated disturbance torque can be fed back in the control loop, making the control strategy more robust.

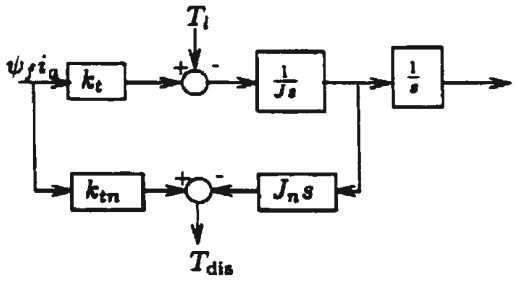


Figure 2.6: Estimation of disturbance torque

2.5 Use of artificial neural networks in induction motor drives

Artificial neural networks have found widespread use in function approximation. It has been shown that, theoretically, a three layer ANN can approximate arbitrarily closely, any nonlinear function, provided it is non-singular [36]. This property has been exploited by a few researchers working in the induction motor drives area.

2.5.1 Estimation of flux, torque and speed

Toh et al [37] developed a flux estimator for use in vector controlled induction motor drives using ANNs. Two ANNs are used, one for the magnitude of the rotor flux and the other for the sine of its space phase angle. The inputs to the ANNs are the d-q axis stator currents in the synchronously rotating reference frame and five previous values of the same. Standard multilayer ANNs with backpropagation are used. The ANNs have been simulated on a PC and training has been performed assuming the availability of the rotor flux magnitude and space phase angle.

Mohamadian et al [38] have implemented an ANN which essentially computes the rotor flux angle and performs the transformation from the synchronous frame to the stationary frame. The rotor flux angle is shown to depend on the synchronous frame d-q axis current, the rotor speed and the previous values of these quantities. The stationary frame d-q axis voltages and their previous values are also given as inputs to the ANN to improve its accuracy. The ANN transforms the synchronous frame d-q axis current commands (which are also given as inputs to the ANN) to the stationary frame d-q axis current commands. A 20-15-2 network is trained by the backpropagation algorithm to achieve the transformation. Though the authors call it an *ANN controller*,

control action is not performed by the ANN.

In another study by Simões and Bose [39], four feedback signals for a direct vector controlled induction motor drive have been estimated using ANNs. A 4-20-4 multilayer backpropagation network has been used for the estimation of the rotor flux magnitude, the electromagnetic torque and the sine and cosine of the rotor flux angle. The ANN is simulated using a commercially available neural network software. The output is not very good, though the authors claim that it can run an induction motor drive in closed loop. The simulation is run on a PC.

In an interesting application of neural networks, Marino et al [40] develop a robust neural network observer for estimating rotor flux and electromagnetic torque of an induction motor. They argue that existing observers suffer from the drawback that, in the presence of uncertainties, their performance deteriorates. To overcome this problem, some researchers have resorted to using adaptive techniques like on-line training, while others use the observer for rotor time constant estimation only.

The authors of [40] have come up with a different way of attacking this problem. They propose an off-line trained observer in which the training set is generated by taking the parameter variations into account, using a stochastic model of the induction motor. A parameter vector Z is defined as

$$Z = \begin{bmatrix} R_s & R_r & L_r & L_s & L_m \end{bmatrix} \tag{2.33}$$

and the identification problem can be defined as the search for the parameter vector which minimizes the following function (also known as the reduced χ^2 function)

$$G(Z) = \frac{1}{3K-5}\sum_{k=1}^{K}\left\{\frac{[\hat{\omega}_r(k)-\omega_r(k,Z)]^2}{{\sigma_\omega}^2}+\frac{[\hat{i}_{ds}(k)-i_{ds}(k,Z)]^2}{{\sigma_i}^2}+\frac{[\hat{i}_{qs}(k)-i_{qs}(k,Z)]^2}{{\sigma_i}^2}\right\} \tag{2.34}$$

where the quantities with ˆ indicate estimated quantities, K is the number of measurements and σ_ω and σ_i are the uncertainties in the rotor speed and the stator current measurements, respectively. Once a suitable minimization procedure is complete, then the parameters lying within a resulting *confidence ellipsoid* will yield a good fit for the measured data. To estimate this confidence ellipsoid, the authors use a direct, global optimization algorithm called the *Price algorithm*. The algorithm is outlined in detail in this paper. After implementing the algorithm, a confidence ellipsoid is obtained, and, if the variations of the plant parameters are within this confidence ellipsoid, then the reduced $\mathcal{X}^2$ function has a small value (< 1.5).

For ANN training, the training set used includes a number of input/output pairs generated via simulations in which parameter variations are introduced. These variations are within the confidence ellipsoid previously determined by the Price algorithm. To ensure a further richness of training set, random signals are also added to the stator voltages. Thus, the training set contains not only parameter variations, as mentioned earlier, but also noise in the stator voltages. The ANN used in this study is of structure 4-20-3 with a hyperbolic tangent activation function. Performance of this ANN, called *stochastic neural network* by the authors, has been shown to be significantly better in the case of varying plant parameters, than using an ANN with deterministic training data, or an extended Kalman filter algorithm.

In one application, Ba-razzouk et al [41] have trained a 5-8-8-2 ANN to estimate the induction motor stator flux using measured stator quantities. After training, the ANN is used in a direct field-oriented controlled drive, which has been simulated using the MATLAB-SIMULINK environment. The rotor flux is computed from the stator flux estimate provided by the ANN and the stator current. The same paper also presents an ANN based decoupler which is used for indirect field-orientation. A 2-8-8-1 ANN is used for implementing the mapping between the flux and torque reference and the

stator current references. A current-fed induction motor model is used for testing the ANN decoupler in simulation.

Even though a lot of research has been carried out into developing ANN techniques to estimate some of the motor parameters like flux and torque, not much work has gone into speed estimation of induction motors using ANNs. Ben-Brahim [42] has used *linear* ANN technique to estimate induction motor speed. Though the technique gives a fairly good estimate of the speed, it lies more in the realm of adaptive control than neural networks. The speed value is not obtained at the output. Instead, the magnitude of one of the weights corresponds to the speed magnitude. Mehrotra et al [43] outline a couple of techniques for estimating the motor speed using ANNs, and these will be discussed in detail in chapter 4, since they form part of the work done for this thesis.

2.5.2 Current control

In one study, Burton et al [44] have used a current control strategy outlined by Wishart and Harley [45] to train an ANN to control induction motor stator currents, but with a different training algorithm. The training algorithm, called Random Weight Change (RWC) algorithm gives almost the same performance as the popular backpropagation algorithm, but is supposed to be slightly faster than the latter. In the RWC algorithm, the weights are perturbed by a fixed step-size and a random sign. This is done for a fixed number of trials and after each trial the error with the desired output is computed. Finally, the set of weight changes which result in the least error are chosen and the whole process is repeated till convergence is reached. Though this scheme has only been tried out in simulation, the authors have proposed a hardware for implementing the RWC algorithm.

Cabrera et al [46] have used ANNs to function as switching state selector for running the inverter in inverter-fed induction motor drives. A 3-5-3 ANN is trained to emulate an existing switching state selector, and various training strategies like backpropagation, adaptive neuron model, extended Kalman filter (EKF) and the parallel recursive prediction error have been tried out and compared. From this work, the authors have found the EKF and parallel recursive prediction error method to be the most effective. However, the authors point out that the use of neural networks in such an application does not demonstrate any tangible benefits over the conventional direct torque control (DTC) method. One reason for this could be that the problem of switching state selection is a very deterministic one and neural networks may not be the appropriate tool to solve such a problem, since it can be handled by a simple algorithm.

2.5.3 Performance enhancement of existing controllers

Neural Networks have been used frequently to improve the performance of existing controllers. Cabrera et al [47] have used a neural network to tune the stator resistance of direct torque controlled (DTC) induction motors. DTC is a commonly used control strategy for tracking a reference electromagnetic torque and stator flux. It uses only stator measurements and avoids the complications which arise in control methods which make use of the rotor time constant. A block diagram of the method is shown in Figure 2.7.

The neural network used in this strategy produces the change in stator resistance, which is added to the previous stator resistance estimate to produce the present estimate for the stator resistance. This value is handed over to the plant which is composed of the induction motor and the DTC strategy. The error between the measured stator current and the reference stator current is used to train the ANN. The ANN is

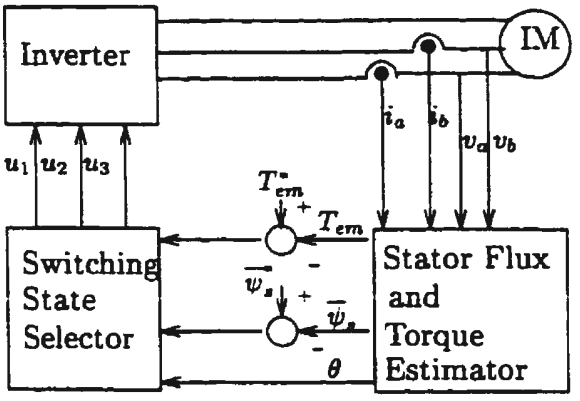


Figure 2.7: Block diagram of DTC

trained using the *Parallel Recursive Prediction Error* algorithm. This method is computationally more intensive than the gradient based methods like backpropagation, but it is supposed to train the network faster for certain applications. The authors have trained various networks ranging in size from 2-2-1 to 2-5-1, and they seem to operate satisfactorily. The networks have also been tried out in an experimental setup using TMS320C30 DSP.

Kung et al [48] use a neural network for improving the performance of a *two-degree-of-freedom* (2DOF) controller. The 2DOF controller comprises of a feedback controller and a feedforward compensator. It is a scalar control technique and requires retuning of the controller parameters under different operating conditions. The model of the plant is given by

$$G_P(z^{-1}) = \frac{\Theta z^{-1}}{1 - \Phi z^{-1}} \tag{2.35}$$

where Θ and Φ are two plant parameters which are derived from the inertia of the drive (J) and its damping coefficient (B). The structures of the feedforward compensator

and the feedback controller are given below

$$G_f(z^{-1}) = \frac{c_0 - c_1 z^{-1}}{d_0 - d_1 z^{-1}} \tag{2.36}$$

$$G_c(z^{-1}) = \frac{R_0 + R_1 z^{-1}}{(1 - z^{-1})(1 + S_1 z^{-1} + S_2 z^{-2} + \cdots + S_d z^{-d})} \tag{2.37}$$

where c_0, c_1, d_0, d_1, R_0, R_1 and $S_1 \ldots S_d$ are controller parameters. The order of the denominator of G_c is $d + 1$, where d is the time delay of the drive model.

It is assumed that the plant parameters Θ and Φ undergo variation during operation of the plant, though the variations are restricted to the ranges

$$\begin{aligned} \Phi_{\min} \leq \Phi \leq \Phi_{\max} \\ \Theta_{\min} \leq \Theta \leq \Theta_{\max} \end{aligned} \tag{2.38}$$

Within the ranges defined in equation (2.38), the parameters are divided into N sets, and for each set of plant parameters, the controller parameters are determined by a rigorous method outlined in the paper. Finally, the ANN is trained using backpropagation method, to produce the desired set of controller parameters with the plant parameters as inputs. Since the mapping between the inputs and outputs is quite complex and unknown, the ANN's generalization property is relied upon to produce a set of controller parameters for any combination of plant parameters not covered in the training set. For real-time operation, a plant parameter identifier is used for providing the inputs to the ANN. The ANN outputs are used to adjust the controller parameters on-line. With this scheme, the 2DOF controller can function satisfactorily under wide operating ranges.

Tadakuma et al [49] have used a 3-2 linear ANN for improving the robustness of a vector-controlled induction motor drive. The basic control scheme uses both feedforward and feedback controllers. The ANN uses on-line training to approximate the induction motor model and one of the ANN weights is used for the computation

of the synchronous speed. The ANN is able to accurately identify the motor in about 8 seconds.

2.5.4 Induction motor control

Narendra and Parthasarthy [50] proposed methods for identification and control of dynamical systems using ANNs. Wishart and Harley [45] use the basic principles outlined in [50] to identify and control induction machines. A block diagram of the control scheme is shown in Fig. 2.8 For the induction motor, the NARMAX (Non-

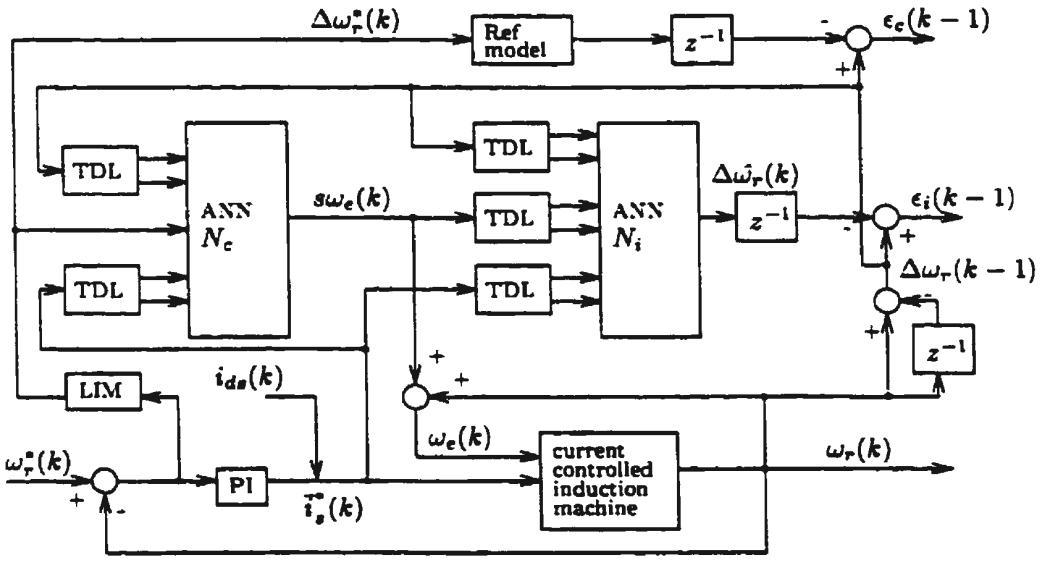


Figure 2.8: ANN control of induction motor [50]

linear AutoRegressive Moving Average with eXogenous inputs) model for the stationary frame stator current is derived and used for identification of the electromagnetic model. In its general form, the NARMAX model represents a system in terms of its delayed inputs and outputs. Random steps in the stator voltage are given for the purpose of

identification. The neural network used is of the multi-layer backpropagation type, and a quantity based on the rotor time constant is also computed as an extra weight. As opposed to the regular ANN architechture, this ANN has no non-linearity in the output layer and the weighted sum of the inputs is used as the output. This gives an estimate of the rotor time constant and makes the system robust against variation of this parameter. Once the identification is over, the ANN is used for current control. The stator currents predicted by the ANN are used to compute the input voltage for the induction motor, and the ANN output is made to track the reference currents by backpropagating the error.

The rotor speed is also controlled in this system by identifying a NARMAX model for the speed increment rather than the absolute value of speed. To simplify the NARMAX model, the load torque is assumed to be a function of the motor speed, as is the case in a fan or a pump type of load. For the current control case, the relationship between the control variable (voltage) and the controlled quantity (current) was linear. In the speed control case, this relationship is non-linear, thus necessitating two ANNs, one for identification of speed and the other for control. The identification ANN (N_i) predicts the value for the speed increment, which is compared with the actual speed increment, and the error (ϵ_i) is backpropagated through the ANN. A PI controller is used for basic speed control, and the control ANN (N_c) produces the slip frequency, and the difference between the desired speed increment and the actual speed increment (ϵ_c) is backpropagated through this ANN. The induction motor drive therefore employs three ANNs. The drive is simulated on a digital computer, and its performance is quite good.

In another recent application [51], an ANN is used to replace the PI speed controller in a vector controlled induction motor drive. The authors develop a model for a robust observer, the output of which is used in the computation of the quadrature axis stator

current reference. The neural network, which is first trained off-line to emulate a PI controller, produces an estimate of the quadrature axis stator current reference, and the difference between the neural network output and the output of the previous block is used for on-line training of the network. This process is demonstrated in Figure 2.9. As is clear from this figure, the neural controller block is redundant, because the desired

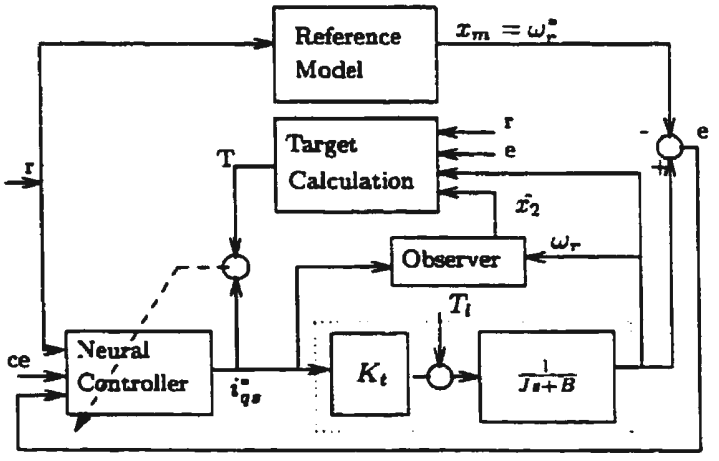


Figure 2.9: Artificial neural network controller

output, which is computed in the "Target Calculation" block, can be directly used as a reference input to the induction motor.

2.6 Summary

From the literature review presented above, it can be seen that induction motor drives have undergone various stages in their development. With developments in the area of power electronics, induction motor drives evolved from essentially constant speed drives to adjustable speed drives. The simple *V/f* control scheme and other scalar techniques

were successfully used in low to medium performance drives. With the evolution in the field-oriented control strategy, it was possible to decouple flux and torque control in induction motor drives. With the developments in the area of microprocessors and DSPs, high performance vector controlled drives could be fabricated. However, use of vector control in both the direct and indirect mode posed some problems. Also, the use of expensive shaft encoders for these drives was considered an undesirable feature. A lot of work was put into developing speed sensorless drives, and most of them used field-oriented control. Various types of schemes were developed to perform sensorless control, but these schemes also had some problems, mainly that of sensitivity to parameter variation. Artificial neural networks have emerged as a powerful tool to identify and control non-linear system. Researchers in the drives area have put in some effort to develop ANN based drives, and they have come up with various applications ranging from estimation of control quantities to assisting conventional controllers in induction motor control. However, complete control of induction motor using one or more ANNs has not been reported in literature so far. Also, most of these methods have been tried out in simulation only, barring a couple of schemes where the ANNs involved are very small (less than 25 weights). In spite of this progress, the potentials of this new technique have yet to be fully exploited.

In the next few chapters, novel ANN based speed estimation and control schemes are proposed with a view to developing an intelligent and robust controller for induction motor drives. As a first step, a versatile object-oriented simulator is developed in the next chapter.

Chapter 3

Development of the Object-Oriented Software

A first step towards studying ANNs and their applications for research purposes is to implement and train them using a software simulator. One of the main benefits of doing simulation based research is that a good deal of flexibility can be obtained as opposed to building a real system. For example, it is much easier to modify a few parameters in simulation and study the effects. To harness this feature more effectively, the simulation software must be well designed. Modularity and reuseability are important features in a good design.

This chapter introduces some of the existing commercial simulators and their features, and it points out why none of them was found suitable for this work. It then discusses the design and development of the object-oriented simulator which was used for this research. The simulator is composed of many different modules, each of which represents a physical system. The user can plug these modules together as required. Since object-oriented languages have been designed to provide these features, it was

decided to use C++ for building the simulator. The simulator was implemented on a UNIX platform, running on a DEC-ALPHA workstation. Section 3.3 discusses the building blocks of this simulator. Since the design of the ANN simulator was more involved, it will be discussed in a separate section.

3.1 An overview of some existing simulators

This section takes a look at some of the popular commercially available software packages, which are widely used by researchers in the motor drives area. This list is by no means exhaustive, but gives a general idea of what products are commonly used and what are the main features and limitations of using these packages.

3.1.1 MATLAB neural network toolbox

MATLAB Neural Network Toolbox [52] comes with a large suite of ANN simulators. It can simulate the perceptron which uses a hard limit activation function and its learning rule. Linear networks can be simulated along with the least-mean-square or LMS (also called Widrow-Hoff) learning rule. These have a single minimum on the error surface, which is a multi-dimensional parabola, and this can be located by the training method. For updating weights after each input, as opposed to updating after each epoch, the adaptive Widrow-Hoff algorithm can be used. Feedforward networks are implemented with the backpropagation training scheme, and these can use the tansig, logsig or linear activation function. Apart from the simple backpropagation algorithm, one can also use the *trainbpx* function which uses momentum and an adaptive learning rate and *trainlm* which implements Levenberg-Marquardt optimization.

Radial basis function networks are implemented with the *simurb* function. Several associative learning rules are implemented — *learnh* (Hebb learning rule), *learnhd* (Hebb learning rule with weight decay), *learnis* (instar learning rule), *learnk* (Kohenen learning rule) and *learnos* (outstar learning rule). Associative learning rules are for producing associations between pairs of vectors (associative memory), i.e. a_1 produces p_1, a_2 produces p_2 etc. These provide the basis for unsupervised networks like the competitive layers and self-organizing map networks. Many Self-organizing networks have been implemented — *trainc* (competitive layers), *trainsm* (Self-organizing Maps) *trainlvq* (Learning Vector Quantization). Finally, recurrent networks like Elman and Hopfield networks are implemented with the *trainelm* and *simuhop* functions respectively.

As can be seen from the above, the MATLAB Neural Network toolbox offers a wide variety of network types and learning algorithms, and it is a very useful tool for researchers working in this area. However, it has a few limitations which reduce its applicability for more general problems. For example, it is unable to simulate more than three layers in a feedforward network, and the user does not have the option of using arbitrary connections and neurons, because of the Matrix approach which has been used for developing this toolbox. Thus, the user would be unable to experiment with non standard architectures.

3.1.2 SIMULINK

SIMULINK [53] is a software package for modeling, simulating, and analyzing dynamical systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can be also multirate, i.e. have different parts that are sampled or updated at different rates. For modeling, SIMULINK

provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations. With this interface, one can draw the models just as you would with a pencil and paper (or as most textbooks depict them). SIMULINK includes a comprehensive block library of sinks, sources, linear and nonlinear components and connectors. Users can customize and create their own blocks. Models are hierarchical, so one can build models using both top-down and bottom-up approaches. One can view the system at a high-level, then double-click on blocks to go down through the levels to see increasing levels of model detail. Model analysis tools include linearization and trimming tools, which can be accessed from the MATLAB command line, plus the many tools in MATLAB and its application toolboxes. Also, because MATLAB and SIMULINK are integrated, users can simulate, analyze and revise their models in either environment at any point.

SIMULINK provides a very flexible and easy to use environment for development of powerful GUI simulators, with the added benefit that the user can access all of MATLAB's powerful routines for solving equations, handling matrices and so on. It is widely used in the industry, and there are hardware accelerator cards available these days which can directly implement systems designed with SIMULINK in real-time [54].

3.1.3 EMTP

The *Electromagnetic Transients Program*, or EMTP [55] for short, is a computer program for simulating electromagnetic, electromechanical and control system transients on multiphase electric power systems. It was first developed as a digital computer counterpart to the analog Transient Network Analyzer. Many other capabilities have been added to the EMTP over a fifteen-year period, and the program is widely used in the utility industry.

Studies involving the use of EMTP can be put into two general categories. One is design, which includes insulation coordination, equipment ratings, protective device specification and control system design. The other is solving operating problems such as unexplained outages or equipment failures. The EMTP is used to solve the ordinary differential and/or algebraic equations associated with an "arbitrary" interconnection of different electrical (power system) and control system compenents. The implicit trapezoidal-rule (second-order) integration is used on the describing equations of most elements which are modeled by ordinary differential equations. The result is a set of real, simultaneous, algebraic equations which is solved at each time-step. These equations are written in nodal-admittance form, and they are solved by triangular factorization. Initial conditions for differential equations of the various components can be determined automatically by the program for most cases of practical interest. The calculation of initial conditions is normally limited to linear elements. Nonlinear resistances are always ignored during the steady state solution. Nonlinear reactances can either be represented by their linear part or fully modeled to include the harmonic distortion effects. Injections of the electric network may also be specified in terms of power and voltage magnitude, thereby providing multi-phase load flow capability. Control system modeling allows for the superposition of an arbitrary number of linear phasor solutions of different frequencies. Program output consists of component variables (e.g., branch currents or voltages, machine torques or speeds, etc.) as functions of time, for those variables requested by the user. Both printed and plotted output are possible, with plotting possible in either character or vector-graphic modes.

EMTP also provides a very powerful environment for simulating electromagnetic systems and power systems. However, it cannot be easily integrated with an ANN simulator, thereby limiting its application in motor drives research.

3.2 Motivation for building another simulator

With so many available simulators, the question naturally arises whether to choose one of them or to develop a new one tailored to one's specific needs. For this work, it was felt that experimenting with different and non-standard neural network architectures would be required, and this implies that the MATLAB neural network toolbox, which is one of the best ones available, would be inadequate. Also, if a new neural network simulator has to be developed in a general purpose programming language, then the other building blocks of the simulator should also be developed in the same language to ensure compatibility. Further, it was felt that having one's own software would ensure complete control over, and transparency of, the source code. It was also felt that porting this source code over to a PC based hardware for experimental verification would be much easier than if a commercially available simulator was used. With all these things in mind, a simulator was built for the purpose of this research, and the rest of this chapter takes a look at its design.

3.3 Building blocks

A block diagram of the simulator as a whole is shown in Figure 3.1. The simulator is composed of many entities, most of them representing a physical object. Each of these entities has been written as a separate class, which makes it a distinct unit and also makes it possible to use multiple objects belonging to each class. This section describes the breakup of the software at the header file level, with details of different classes within a header file. Since the software is built in a hierarchical fashion, with the higher layers using the services provided by the lower layers, the description will follow a bottom-up approach.

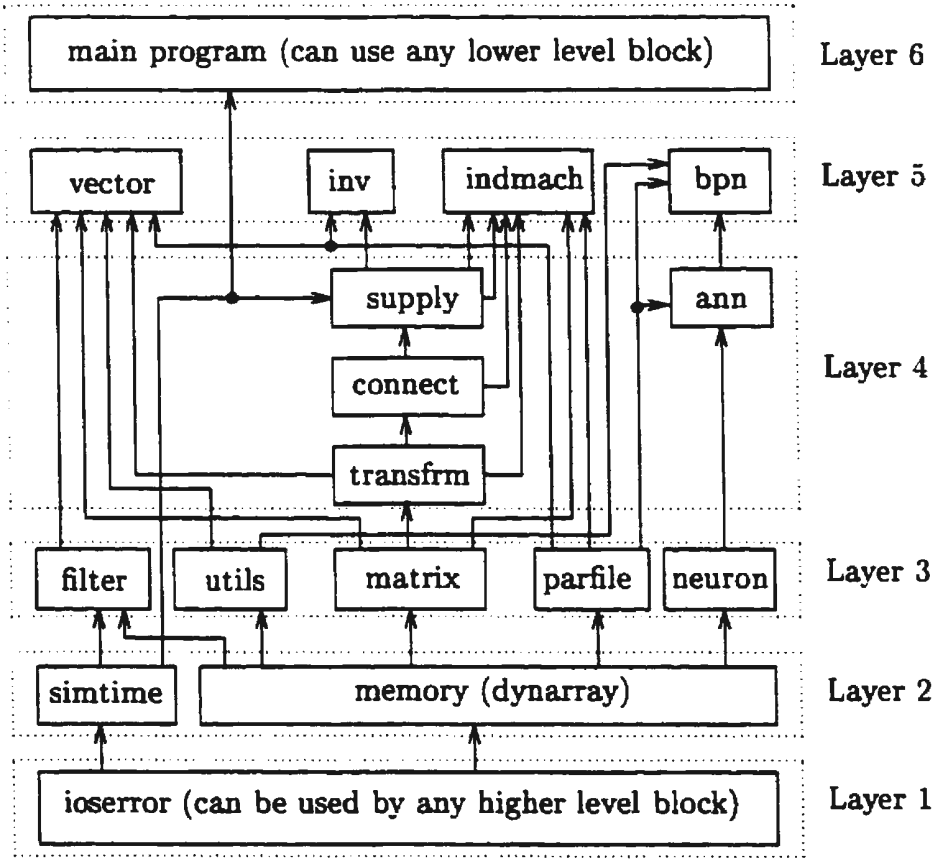


Figure 3.1: Block level description of the simulator

- *ioserror*: This header file contains a class with the same name which is used by most blocks at the higher level for error management. This class constitutes the lowest layer in the hierarchy. It provides a standard interface for error handling and each class at the higher level can include a *static* object of the ioserror class. Thus, multiple objects of the higher level class can share the same error handler. Each class can generate a warning or a critical error. A critical error results in immediate shutdown of the program, while a warning just generates a message upon regular program shutdown. The error messages are generated by the respective class and the ioserror class reports which class generated the error message. Having a standard error interface greatly simplifies program development and debugging.

- *memory*: This header file is placed in the second layer of the hierarchy and contains a class called *dynarray* which creates a dynamically sized array. This can be used in applications where the size of the array is not known beforehand. The user can keep writing to an object of the dynarray class, and it will automatically adjust the array size to accommodate the data.

  The memory header file also contains overloaded functions for allocating memory and generating a fatal error message, in case the operating system is unable to allocate the requested memory.

- *simtime*: This header file is also placed in the second layer of the hierarchy and contains a class with the same name, which controls the timing operation for the complete simulation. It must be used by the main program, if any timing operations are being used. The entire simulation uses just one clock for any function or class which requires timing. The main program increments the clock, and every other function or class receives the correct time from this clock. Furthermore, every class or function computes the increment interval on its own, and thus each can be run at different intervals, without risking timing accuracy.

- *filter*: This header file is placed in the third layer of the hierarchy and contains the following class descriptions:

  1. *Digital Filters*: Two kinds of filters have been simulated — *average filter* and *first order filter*. The average filter provides a weighted average of a set number of previous values, as specified in equation (3.1).

$$w(i) = \frac{e^{-[\frac{(i-n/2)}{\sigma}]^2}}{\sum_{k=0}^{n-1} w(k)} \tag{3.1}$$

  A bell shaped weighting curve is used, and the filter length n, and the variance σ can be set for each individual filter. Since this is only an averaging filter, it doesn't use any timing operations.

  The first order filter uses the *bilinear transformation*

$$s = \frac{2}{T}\frac{(1 - z^{-1})}{(1 + z^{-1})} \tag{3.2}$$

  to convert a first order analog filter of the form

$$H(s) = \frac{A}{A + s} \tag{3.3}$$

  to a digital filter. It needs a knowledge of the time step T for computing the filter output.

  2. *PI_controller*: This class simulates a PI controller which is given by the transfer function

$$H(s) = K_p + \frac{K_i}{s} \tag{3.4}$$

  and makes use of the integrator class for performing the integration. The PI constants, K_p and K_i, can be set independently for each controller.

  3. *Integrator and Differentiator*: These two classes perform numerical integration and differentiation. The integrator class uses the *trapezoidal rule* for

performing the integration. The differentiator uses the *backward difference* method for computing the derivative. Both classes need a knowledge of the time step T.

- *utils*: This header file is placed in the third layer of the hierarchy and contains a class for some useful tasks which many classes might need to perform.

    1. *datafile*: This class enables easier handling of datafiles which are used for outputting data onto the disk. The maximum file size and the sampling interval can be set externally.
    2. *tdltype*: This class is used to simulate a tapped delay line which is frequently used in ANN applications to generate previous values of certain inputs. The class does not use any timing operations, and the nth previous value is produced as the output, n being externally selectable.
    3. *counter*: This class is used for producing clocking pulses for certain applications. It outputs a pulse after n iterations, where n is externally selectable.

- *matrix*: This header file is placed in the third layer of the hierarchy and contains two classes — one called *matrix* and the other called *complex*. The matrix class simulates an $m \times n$ matrix, where m and n are externally selectable. Many matrix operations are provided, like addition, subtration and multiplication. The complex class simulates complex numbers, and various operations for complex numbers are also defined, e.g. addition, subtraction, multiplication and division.

- *parfile*: The header file is placed in the third layer of the hierarchy and contains the definition of a class by the same name, which is used to facilitate reading parameter files used by various other classes. For example, the induction machine has many parameters like the stator and rotor resistances and inductances, number of poles, moment of inertia and so on, which are set externally. All these are read from a parameter file, and an object of the parfile class, provides a standard

and convenient interface for doing the same. The parameter files can contain comment statements, and the parameters can be arranged in any fashion, but they are read correctly by an object of the parfile class.

- *neuron*: This header file is placed in the third layer of the hierarchy, and contains classes for simulating different kinds of neurons. It is discussed in greater detail in the next section.

- *transfrm*: This header file is placed in the fourth layer of the hierarchy and this layer is further subdivided into three sublayers. *transfrm* occupies the lowest of these sublayers. It contains two classes for performing data transformation operations. These two classes are discussed below:

  1. *three_phase*: This class simulates a three-phase quantity and automatic conversion is performed between three-phase and two-phase direct-axis and quadrature-axis quantities and vice-versa as given in equations (3.5) to (3.9).

$$i_d = \frac{2i_a - i_b - i_c}{3} \tag{3.5}$$

$$i_q = \frac{i_b - i_c}{\sqrt{3}} \tag{3.6}$$

$$i_a = i_d \tag{3.7}$$

$$i_b = 0.866 i_q - 0.5 i_d \tag{3.8}$$

$$i_c = -(i_a + i_b) \tag{3.9}$$

  2. *mag_freq*: This class performs transformation between direct, quadrature-axis quantities and magnitude, frequency and vice-versa as given in equations (3.10) to (3.14).

$$i_{\text{mag}} = \sqrt{i_d^2 + i_q^2} \tag{3.10}$$

$$\Delta\theta_i(k) = \arctan\frac{i_q(k)}{i_d(k)} - \arctan\frac{i_q(k-1)}{i_d(k-1)} \tag{3.11}$$

$$\theta_i(k) = \sum_{n=0}^{k-1} \Delta\theta_i(n) \tag{3.12}$$

$$i_d = i_{\text{mag}} \cos\theta_i(k) \tag{3.13}$$

$$i_q = i_{\text{mag}} \sin\theta_i(k) \tag{3.14}$$

This class is used for data conversion in ANN training for induction motor control.

- *connect*: This header file occupies sublayer 2 of layer 4 in the hierarchy. It contains two classes — *single_conductor* and *three_phase_conductor* — which model a single-phase and a three-phase conductor respectively. The presence of the conductor classes makes it possible to have a standard interface for all higher level objects, ensuring any kind of connection between different objects. The conductors have error checking to ensure that, for example, two voltage sources are not tied to the same conductor leading to a short circuit. Also monitoring of currents, voltages and power flow is facilitated by member functions of these classes which return these respective values.

- *supply*: This header file occupies the third and highest sublayer in layer 4 of the hierarchy. It contains classes for the following:

  1. *dc_supply*: This class simulates a DC power supply, and it is used chiefly for supplying the DC bus of the *inverter*. The voltage level can be set externally.
  2. *three_phase_supply*: This class, as the name implies, simulates a three-phase power supply, which can be used to drive a three-phase rectifier or an induction motor directly.
  3. *current_amplifier*: This class simulates an ideal three-phase current source.
  4. *function_generator*: This class, as the name implies, simulates a function generator which can produce the following different kinds of waveforms — square, triangle, sine and sawtooth. For each waveform, the frequency,

amplitude and the DC offset can be set. It can also produce a balanced three-phase sinewave.

- *ann*: This header file occupies layer 4 of the hierarchy and defines a class by the same name. This is discussed in greater detail in the next section.

- *vector*: This header file occupies the fifth layer in the hierarchy and contains a class called *vector_controller*. This class implements the indirect rotor flux-oriented control strategy for a squirrel-cage induction motor drive. It has separate blocks for flux estimation, i_{mr} estimation, ramp reference generation for speed and flux reference generation. It uses PI controllers, which are declared as objects of the PI_controller class. The main vector controller function receives the current feedback, speed feedback and the reference speed setting as inputs. It computes the three-phase reference currents which are handed over to the current controller. The speed reference setting acts as an input to the speed reference generator block inside the vector controller, which ramps up the speed reference slowly, to prevent any instabilities.

  The main function first converts the speed to electrical rad/s. The conversion from three-phase to two-phase d-q axis quantities is automatically done in the *three_phase* class. Next, the flux estimator, the flux reference generator and the speed reference generator blocks are run. After this, the torque is computed in the synchronously rotating frame of reference. The various PI controllers are run next and the outputs from the PI controllers in the last stage, which are the reference currents in the rotating frame, are converted back to quantities in the stationary frame.

- *inv*: This header file occupies the fifth layer in the hierarchy and contains classes for the inverter and three different types of current controller.

  1. *inverter*: This is modeled in a class called *inverter* and simulates a three-

phase transistor inverter. At the input side, it interfaces to an object of the *dc_supply* class, which supplies the dc bus of the inverter. At the output, it interfaces to an object of the *three_phase_conductor* class. The main inverter function receives the six gate signal pulses as input, and computes the output voltages for the three-phase conductor at its output. It checks to see whether there is a DC short circuit, or if the voltages and currents exceed the specified transistor ratings.

2. *current controllers*: Three different types of current controllers have been modeled — a PWM voltage controller called *pwm_vc* , a PWM current controller called *pwm_cc* and a hysteresis current controller called *hyst_cc*. These classes emulate the well known current control strategies. The two current controllers need the three-phase reference and actual currents as inputs to their main function. The PWM voltage controller needs the magnitude and frequency of the modulating signal as inputs to its main function. All these functions supply the six gate drive pulses at the output, and these gate drive pulses can be given directly to an object of the inverter class. Dead-time (or *blanking* time) required between the top and bottom transistors in the same leg of the inverter has not been considered in this simulation because it is usually in the range of a couple of μs, and this would imply a reduction in the simulation step time causing a large increase in the actual run time of the simulation.

- *indmach*: This header file occupies the fifth layer in the hierarchy and models a class for the squirrel-cage induction machine and uses the Runge-Kutta method for solving the d-q axis induction motor dynamic equations, i.e. (1.1) ... (1.3). To make this set amenable to solution by Runge-Kutta method, we separate all the derivative terms and rearrange the equations in the state-space format, i.e.

$$\dot{I} = A_v I + B_v \tag{3.15}$$

where

$$I = \begin{bmatrix} i_{ds} & i_{qs} & i_{dr} & i_{qr} \end{bmatrix}^T \tag{3.16}$$

and

$$A_v = \frac{1}{L_r L_s - L_m{}^2} \begin{bmatrix} -R_s L_r & \omega_r L_m{}^2 & R_r L_m & \omega_r L_m L_r \\ -\omega_r L_m{}^2 & -R_s L_r & -\omega_r L_m L_r & R_r L_m \\ R_s L_m & -\omega_r L_s L_m & -R_s L_s & -\omega_r L_r L_s \\ \omega_r L_s L_m & R_s L_m & \omega_r L_r L_s & -R_r L_s \end{bmatrix} \tag{3.17}$$

$$B_v = \frac{1}{L_r L_s - L_m{}^2} \begin{bmatrix} v_{ds} L_r \\ v_{qs} L_r \\ -v_{ds} L_m \\ -v_{qs} L_m \end{bmatrix} \tag{3.18}$$

This model of the induction motor is called the voltage source model, since it assumes that the motor is supplied by a voltage source. It is also possible to supply the induction motor with a current source, and this model is also included in the induction motor class. For the current source model, it is assumed that the motor is supplied by a three-phase current source at the stator, and hence i_{ds} and i_{qs} in equation (1.1) are known and we have to solve for i_{dr}, i_{qr}, v_{ds} and v_{qs} and estimate the torque and motor speed. The rotor currents can be estimated by applying the Runge-Kutta method to the following equations

$$\dot{I} = A_i I + B_i \tag{3.19}$$

where

$$A_i = \frac{1}{L_r} \begin{bmatrix} -R_r & -\omega_r L_r \\ \omega_r L_r & -R_r \end{bmatrix} \tag{3.20}$$

and

$$B_i = \frac{L_m}{L_r} \begin{bmatrix} -\frac{di_{ds}}{dt} - \omega_r i_{qs} \\ -\frac{di_{qs}}{dt} + \omega_r i_{ds} \end{bmatrix} \tag{3.21}$$

The d-q axis stator voltages can then be estimated as follows:

$$v_{ds} = R_s i_{ds} + L_s \frac{di_{ds}}{dt} + L_m \frac{di_{dr}}{dt} \tag{3.22}$$

$$v_{qs} = R_s i_{qs} + L_s \frac{di_{qs}}{dt} + L_m \frac{di_{qr}}{dt} \tag{3.23}$$

At the input, the induction motor interfaces to an object of the *three_phase_conductor* class. The induction motor is automatically able to detect whether the three-phase conductor is being supplied by a voltage or a current source, and it switches to the correct model for solving the dynamic equations. The only input to the main function of the induction motor class (called *run*) is the load torque, and the output of this function is a structure which contains the motor speed in rad/s and the electromagnetic torque.

- *bpn*: This header file occupies the fifth layer in the hierarchy and contains a definition of the *backprop* class. This class implements the backpropagation algorithm and is discussed in greater detail in the next section.

At the highest, or the user level, there are main program(s) in which a user puts together objects from the classes discussed above.

3.4 Artificial neural network simulator

The ANN simulator has been developed using a hierarchical approach, and it can be used by a programmer to study non-standard network topologies and also create new neuron types with minimal modification to the remaining code [56]. It is possible to interconnect neurons in any fashion, without limiting the network to a layered architecture. Also, each neuron in the simulator can be of a different type and can have

a different learning rate. This approach allows new training algorithms, or the addition of a new type of neuron without affecting existing functionality. In addition, one of the biggest benefits of this scheme is that it encompasses a variety of different neural network types like Multilayer feedforward network, Hopfield, Radial Basis Function network and Elman network. Fig 3.2 shows a layout of the simulator.

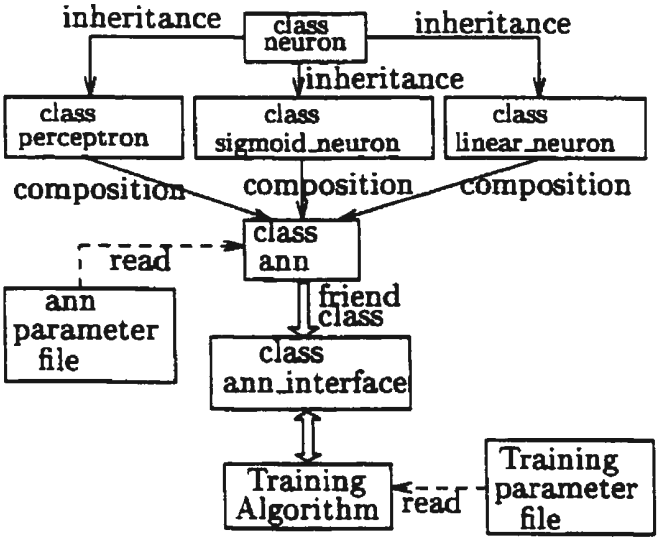


Figure 3.2: Structure of the simulator

Most types of ANNs can be subdivided into three constituent parts [9] — the *processing element (neuron)*, the *connections* and the *learning rule*. The following subsections describe this classification in greater detail.

3.4.1 The neuron

The fundamental unit in the ANN is the neuron. The neuron is implemented with a two-level design. At the root level, a *neuron* class is defined which implements the

common features of the different neuron types. These common features are

- a set of inputs
- a set of weights
- a synaptic function which might differ from neuron to neuron
- the output of the synaptic function (usually called *net*)
- an activation function which might differ from neuron to neuron
- the neuron output

The *neuron* class can then be inherited by the classes for the different neuron types e.g. *sigmoid neuron*, *linear neuron* or *perceptron*. The sigmoid and linear neurons have a sigmoid and linear activation function respectively, and the perceptron has a hard-limiting activation function. Most neurons use the linear synaptic function, and this is implemented in the *neuron* class. However, the inheriting class can redefine this function if need be, since the function is implemented using the *virtual* mechanism [57]. The activation function is implemented only as a dummy prototype in the *neuron* class, since it is expected that each new neuron would have a different activation function. Thus, the inheriting class must define an activation function.

3.4.2 The network

The creation of *neuron* objects and their connections is implemented in the constructor of the *ann* class. An object of this class reads from a parameter file which defines the structure of the ANN. The user has to create this parameter file which consists of three parts. The first part defines the number of neurons, the number of ANN inputs and

the number of ANN outputs. The second part describes each neuron type, the number of inputs for each neuron and any optional parameters like 'LR' (learning rate) or 'B' (β in the sigmoidal activation function). The third part of the parameter file consists of the connection description for each input of each neuron. It describes where each input of each neuron is connected to. An example of a parameter file is shown below. Any line which begins with a '%' is a comment line.

```
% Part 1
% Number of neurons
3
% Number of ann inputs
1
% Number of ann outputs
1
% Part 2 (neuron description)
% number     neuron type    inputs    optional
    0         TANSIG          1        LR=0.3    B=0.9
    1         LINEAR          1        LR=0.3
    2         LINEAR          3        LR=0.1
% Part 3 (connection description)
% type      from       to
  INPUT        0        0
  INPUT        0        1
  INPUT        0        2
  HIDDEN       0        2
  HIDDEN       1        2
  OUTPUT       2        0
```

Fig 3.3 shows a diagram of the network which is represented in the parameter file shown above. In this file, neuron 0 is a TANSIG neuron (with a tan-sigmoid activation function), and the other two neurons are LINEAR neurons (with a linear activation function).

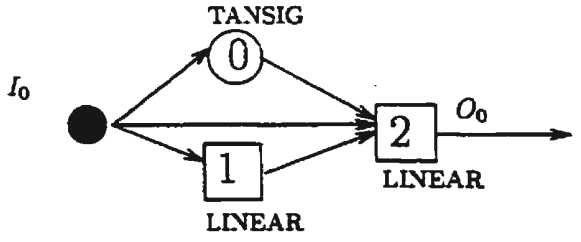


Figure 3.3: The network corresponding to the example parameter file

The last part of the parameter file details the neuron connections. Connections are of three types. If the connection is between an ANN input and a neuron input, the type is **INPUT**. If a connection is between a neuron output and a neuron input, it is of type **HIDDEN**. Finally, if a connection is between a neuron output and an ANN output, it is of type **OUTPUT**. The *from* and *to* fields refer to the input number, neuron number or output number as the case may be. It should be noted that with this scheme it is possible to define recurrent ANNs as well as neurons with self feedback.

Apart from creating the neurons and making the various connections, the *ann* class should also provide an interface for the training method. This interface is implemented as a *friend* class called *ann_interface* (Fig 3.2). This class contains functions for computing the output of a particular neuron, accessing any free parameter of a particular neuron and accessing the various neuron parameters (learning rate, number of inputs, threshold, local gradient and so on).

3.4.3 Training algorithm

The training method is written as a separate class which inherits the *ann_interface* class. As an example, a class *backprop* has been written to modify the weights of the network according to the backpropagation rule. It reads training parameters from a separate parameter file. There are two main functions — *forward_pass* and *backward_pass*, which implement the two main activities in backpropagation. The weights are saved periodically in a weight file as training progresses. The weight saving and weight reading functions are implemented in the *ann* class.

3.5 Putting it all together

At the top level in Figure 3.1, the block named "main program" makes use of the lower level blocks to build useful programs. Some of the programs which were built with the basic building blocks are briefly discussed below. It should be noted that the main program must run the system clock if it uses any lower level block which requires timing.

- *gendata*: This program is used for generating the data that is used for ANN training. The program uses the induction motor, vector controller, inverter and other relevant blocks, and runs the vector controller with step changes in the reference speed and load torque every couple of seconds. The inputs needed for the ANN and the desired outputs are stored on a disk file.

- *pd*: This program, which is an acronym for "peak detector", reads the datafile generated by gendata and finds the peak magnitudes for all the ANN inputs and outputs. These values, which are written onto the disk in a separate file, are used

as the normalizing and denormalizing gains for ANN training.

- *layerann*: This program is used for generating the parameter file which is used by the "ann" block. As has been discussed earlier, the ann block needs the network architecture in a specially formatted file, and this file can become very large for an average sized network. To save the user from typing this big text file containing the network description, the layerann program generates this file for a feedforward network with a given number of layers and neurons in each layer.

- *bpntrain*: This program is used for training a network using the backpropagation algorithm. It reads the training parameters like momentum, weight file name and so on from a disk file, and it also reads the normalizing and denormalizing gains from the same file. Next, it reads the complete data file, which contains the training vectors generated by gendata, and presents these vectors to the network for a specified number of epochs. The training vectors can be randomly shuffled if desired, because this helps the backpropagation algorithm in its search for the minimum.

- *nnim*: This program is used to run the induction motor using the trained network. It substitutes the vector controller with an ANN which has been off-line trained to mimic a vector controller.

3.6 Summary

This chapter describes a suite of software that was developed as a part of, and for facilitating, this research. The software is completely modular and can be easily extended. It models physical objects closely and thus greatly facilitates experimentation, since the user just has to put the various blocks together in the desired fashion to run an

experiment. This suite of software might benefit other researchers also, both at Memorial University of Newfoundland and elsewhere. The next two chapters discuss some simulation experiments that were carried out using this software, and the theoretical contribution made as a result of those experiments.

Chapter 4

ANN Based Induction Motor Speed Estimator

It was seen in chapter 2, that sensorless control is an important issue in present day research in the area of induction motor drives, and various techniques have been developed for the same. ANNs were shown to have a great potential for non-linear function approximation and control applications. They also have various benefits not present in other techniques. It was seen from the literature survey that almost no work has been done to develop sensorless drives using ANNs.

This chapter investigates the use of the function approximation property of ANNs for speed estimation of induction motor. This property was exploited by first considering the d-q axis dynamic equations of the induction motor and obtaining expressions for speed, based on the measurable stator quantities. These expressions were used to develop three different ANN based schemes for induction motor speed estimation. Though the schemes give a reasonable output, it was felt necessary to reduce the network sizes and improve performance at the same time. With this mind, another

approach for speed estimation was developed, which satisfies both these criteria. This chapter also presents simulation results for all the four schemes. For the purpose of simulation, extensive use was made of the software described in chapter 3.

4.1 Induction motor equations

The d-q axis dynamic equations for the squirrel-cage induction motor are very well known and are given in equation (1.1). We can see that in equation (1.1), if the stator voltages and stator currents are known along with the machine parameters, we have only 3 unknowns, viz. ω_r, i_{dr} and i_{qr}. We can thus solve for ω_r in terms of the stator quantities only. First, we obtain the rotor currents as functions of the stator quantities and ω_r from the first two rows of equation (1.1), since the rotor currents are not accessible in a squirrel-cage induction motor. The expressions obtained for i_{dr} and i_{qr} in the stationary reference frame are

$$i_{dr} = \frac{1}{L_m}[\int (v_{ds} - R_s i_{ds})dt - L_s i_{ds}] \tag{4.1}$$

$$i_{qr} = \frac{1}{L_m}[\int (v_{qs} - R_s i_{qs})dt - L_s i_{qs}] \tag{4.2}$$

When we substitute equations (4.1) and (4.2) in the last two rows of equation (1.1) we obtain the following two expressions for the rotor speed ω_r

$$\omega_r = \frac{-[\sigma^2 \frac{di_{ds}}{dt} - R_r L_s i_{ds} + R_r \int v_{dx}dt + L_r v_{dx}]}{\sigma^2 i_{qs} + L_r \int v_{qx}dt} \tag{4.3}$$

$$\omega_r = \frac{[\sigma^2 \frac{di_{qs}}{dt} - R_r L_s i_{qs} + R_r \int v_{qx}dt + L_r v_{qx}]}{\sigma^2 i_{ds} + L_r \int v_{dx}dt} \tag{4.4}$$

where, $v_{dx} = v_{ds} - R_s i_{ds}$, $v_{qx} = v_{qs} - R_s i_{qs}$ and $\sigma^2 = L_m^2 - L_r L_s$. The induction motor speed can be recovered from either of equations (4.3) or (4.4). However, both of these equations have singularities for regular induction motor operation. This can be seen

by plotting the numerator and denominator of either equation. Figure 4.1 gives this for the numerator and denominator of equation (4.3).

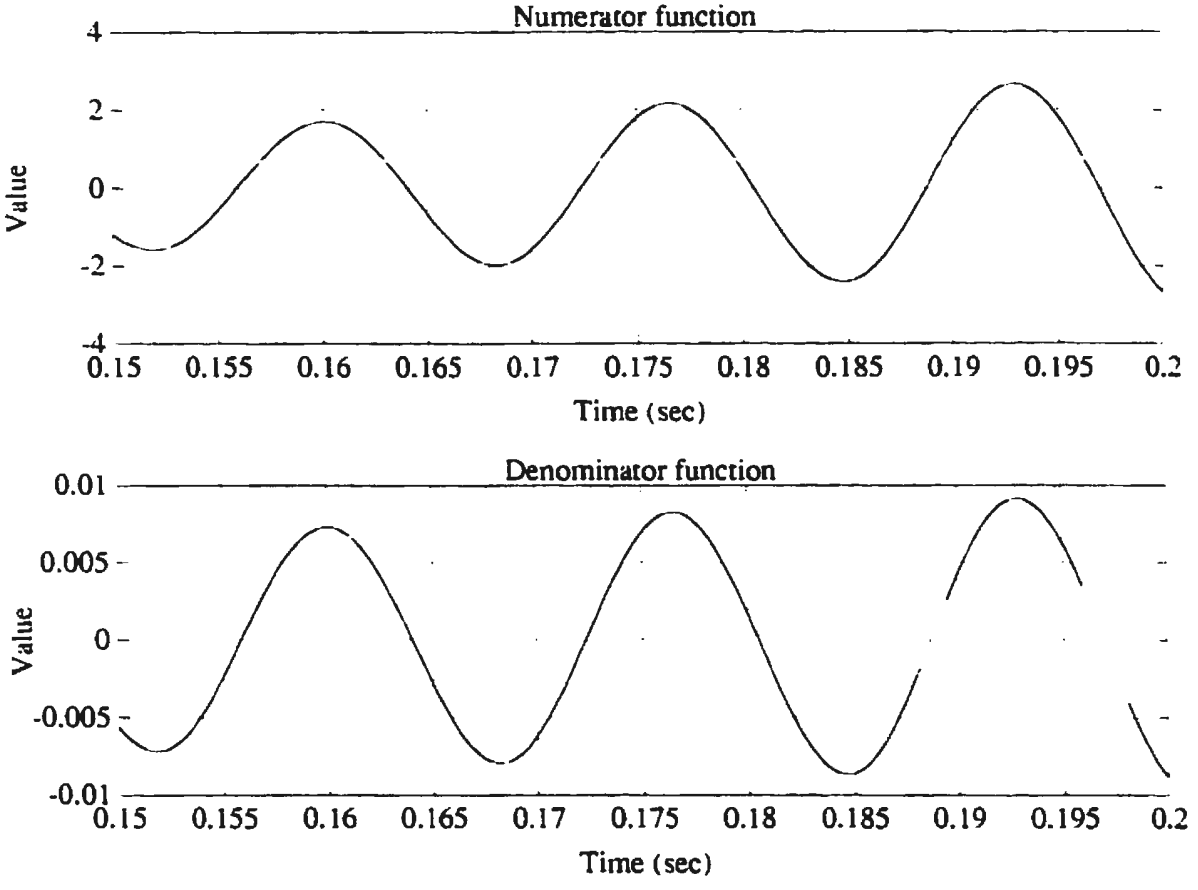


Figure 4.1: Numerator and denominator functions in the speed expression

It can be seen from Figure 4.1 that both waveforms are in phase, resulting in simultaneous zero-crossings, and hence, singular points. Equation (4.4) also produces a similar plot. A comparison of the numerators and denominators of equations (4.3) and (4.4) is shown in Figure 4.2.

From Figure 4.2 we can see that both the numerator and denominator of equa-

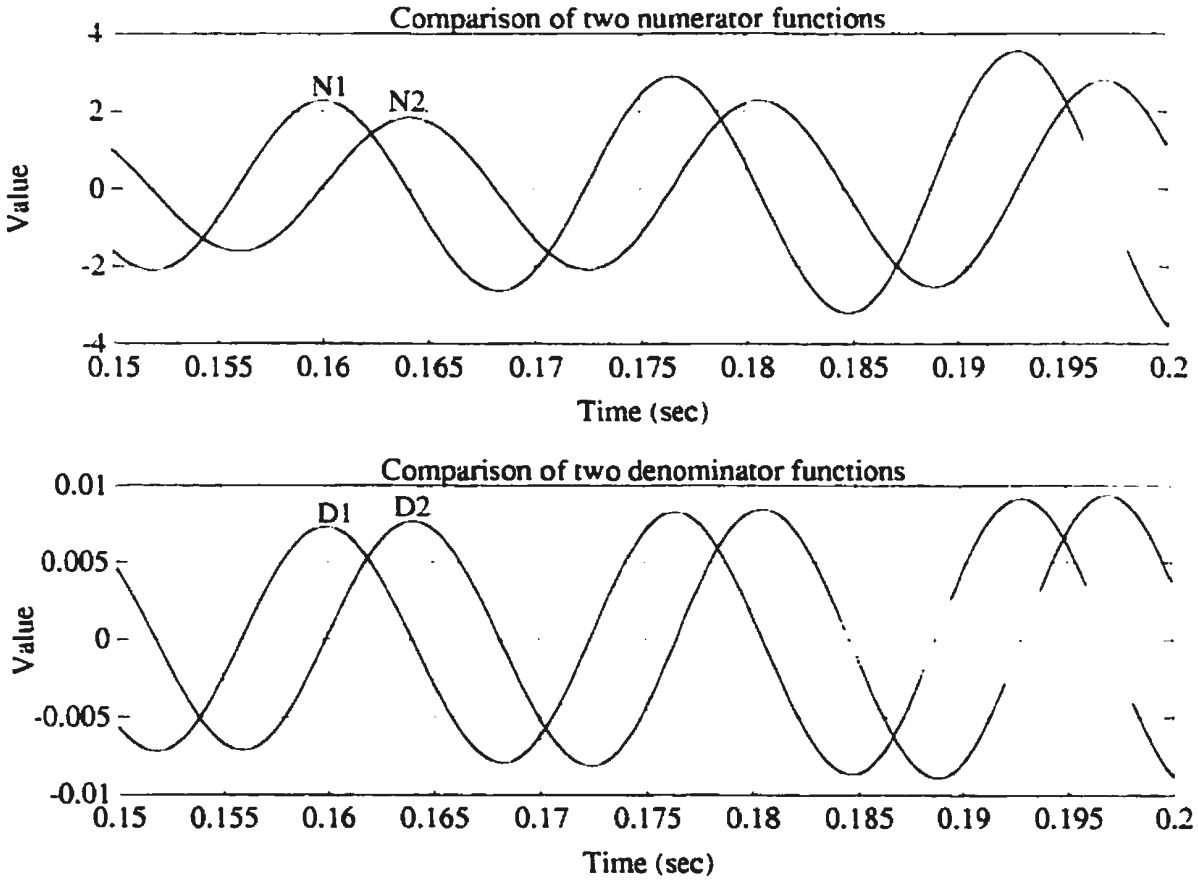


Figure 4.2: Comparison of the two speed expressions

tion (4.4) lead those of equation (4.3) by 90°. This suggests that we can treat them like d and q components of a vector or space phasor quantity. Thus equations (4.3) and (4.4) can be combined and expressed as

$$\omega_r[\sigma^2 i_{qs} + L_r \int v_{qx} dt] = -[\sigma^2 \frac{di_{ds}}{dt} - R_r L_s i_{ds} + R_r \int v_{dx} dt + L_r v_{dx}] \tag{4.5}$$

$$\omega_r[\sigma^2 i_{ds} + L_r \int v_{dx} dt] = [\sigma^2 \frac{di_{qs}}{dt} - R_r L_s i_{qs} + R_r \int v_{qx} dt + L_r v_{qx}] \tag{4.6}$$

Multiplying equation (4.6) by j and subtracting from equation (4.5) we can obtain another expression for the rotor speed as

$$\omega_r = \frac{(\sigma^2 p - R_r L_s - R_s L_r - \frac{R_s R_r}{p})\bar{\mathbf{i}}_s + (L_r + \frac{R_r}{p})\bar{\mathbf{v}}_s}{j[(\sigma^2 - \frac{R_s L_r}{p})\bar{\mathbf{i}}_s + \frac{L_r}{p}\bar{\mathbf{v}}_s]} \tag{4.7}$$

where $\bar{\mathbf{i}}_s$ and $\bar{\mathbf{v}}_s$ represent the stator current vector and stator voltage vector respectively. Note that the R.H.S. of equation (4.7) is complex and the speed would be given as the magnitude of the equation.

4.2 Speed estimation

The primary goal of building a speed estimator is to use it in an induction motor drive system which requires speed feedback for closed loop control. An inverter forms an essential component of an induction motor drive system, because control over the voltage magnitude and frequency is desired. The presence of an inverter results in a non-sinusoidal stator voltage waveform and ripples in the stator current also. This would impede ANN training, and thus it might be more desirable to study the ANN scheme under more ideal conditions and then try out its efficacy in the presence of an inverter.

One of the necessary conditions for an ANN to approximate a function is that the function be *square integrable* in the unit cube [58]. This condition is obviously

not satisfied by either equation (4.3) or equation (4.4). Thus, it would be futile to attempt to train the ANN using the quantities on the right hand side of either of these equations. This problem does not exist in equation (4.7), but it is a more complex function compared to the other two.

The basic problem in training an ANN to recognize induction motor speed is that the functional relationship between the speed and the stator parameters is quite complex with the result that the number of training vectors required is very large. Also, the training times involved on a PC become quite substantial. To circumvent the problems of large data files and substantial training times, it was decided that the whole drive system with the ANN be implemented in simulation, using the software outlined in chapter 3. The training vectors could be generated on-line, obviating any need for storing training data on the disk. For simulation, the induction motor model used has the parameters shown in Table 4.1.

4.2.1 Method 1: Using singular functions

As a first step, it was decided to use equations (4.3) or (4.4) for the purpose of ANN training. It has been mentioned in section 4.1 that both these equations have singularities. Thus, the basic idea in this method is to partition the main function having singularities (or *poles*) into smaller functions, each of which do not have any singularities, and to train multiple ANNs to identify these smaller functions. The desired output can be obtained from the outputs of these ANNs by avoiding the singular points or poles of the main function. In this case, one of the simplest ways to partition the functions is to consider their numerators and denominators separately. The numerators and denominators of equations (4.3) and (4.4) can be expressed respectively as

$$N_1 = -[\sigma^2 \frac{di_{ds}}{dt} - R_r L_s i_{ds} + R_r \int v_{dx} dt + L_r v_{dx}] \tag{4.8}$$

Table 4.1: Induction motor parameters used in simulation studies

Parameter	Symbol	Value
Power rating		1.5HP
Voltage		208V
Connection type		Y
Stator Resistance	R_s	0.49 Ω
Rotor Resistance	R_r	0.45 Ω
Stator Inductance	L_s	37.1 mH
Rotor Inductance	L_r	37.1 mH
Magnetizing Inductance	L_m	35.4 mH
Moment of Inertia	J	0.024 $N-m^2$
Damping Coefficient	B	0.0011 $N-m^2/s$
Pole Pairs	P	2

$$D_1 = \sigma^2 i_{qs} + L_r \int v_{qx} dt \tag{4.9}$$

$$N_2 = [\sigma^2 \frac{di_{qs}}{dt} - R_r L_s i_{qs} + R_r \int v_{qx} dt + L_r v_{qx}] \tag{4.10}$$

$$D_2 = \sigma^2 i_{ds} + L_r \int v_{dx} dt \tag{4.11}$$

ANNs can be trained to approximate either N_1 and D_1 in equations (4.8) and (4.9) or N_2 and D_2 in equations (4.10) and (4.11). The output of these ANNs can then be passed through a filter which performs the required division at points where both the numerator and denominator are non zero [59]. A block diagram of the speed recovery scheme using equation (4.4) is shown in Figure 4.3.

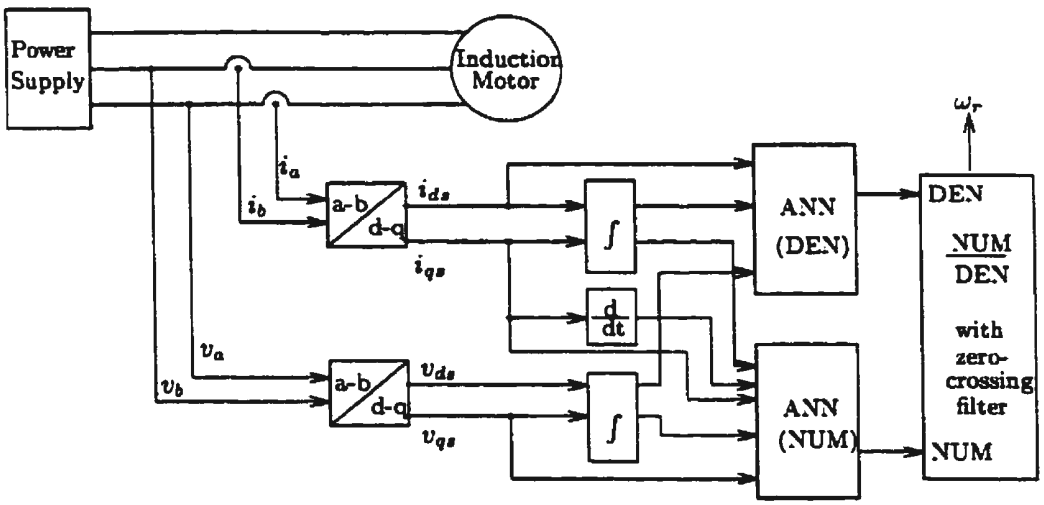


Figure 4.3: Block diagram of ANN speed recovery (method 1)

In the simulation study performed, both the ANNs have 2 hidden layers each. Inputs given to the numerator ANN were $i_{qs}(k)$, $\frac{di_{qs}}{dt}$, $v_{qs}(k)$, $\int v_{qs} dt$ and $\int i_{qs} dt$, based on the expression for N_2 and the network structure chosen was 5-26-13-1. Inputs given to the denominator ANN were $i_{ds}(k)$, $\int v_{ds} dt$ and $\int i_{ds} dt$, based on the expression for

D_2 and the structure chosen was 3-20-10-1. Training was initiated by giving random inputs to the ANNs. The same random inputs were also given to the numerator and denominator functions and the desired outputs for training the ANNs were obtained from these functions. The learning rate (η) was maintained at 0.25 and the value of the momentum parameter (α) was fixed at 0.2 throughout the training. After giving 5 million (5×10^6) vectors of random inputs, 1 million vectors of direct on-line start using a power supply, with step change in load torque were given. After this *off-line training*, the ANNs were able to mimic the numerator and denominator functions quite accurately. For the actual speed recovery, a *peak detector* was used as the filter in the block diagram. This detects the positive and negative peaks of the sinusoidal outputs of the two ANNs, and performs the required division at these points. The actual and ANN recovered speeds for *direct on-line* start and step change in load torque are shown in Figure 4.4. As can be seen from Figure 4.4, the performance of the speed estimator is quite accurate.

For inverter operation, this method can be used without the need for filtering the feedback signals [60]. However, a first order filter was used at the output, because the speed estimate obtained from the 2 ANNs had a lot of ripples. In the simulation study, a first order low pass filter (LPF) with the following transfer function was used

$$H(s) = \frac{80}{s + 80} \tag{4.12}$$

The two ANNs were trained for 3 million (3×10^6) iterations in the presence of a PWM current-controlled voltage source inverter, with step changes in magnitude and frequency of the voltage and load torque.

The resulting estimated speed with the LPF in series with the peak detector circuit is shown in Figure 4.5. It can be seen that this simple filter has resulted in the removal of most of the ripples during the startup period. However, there is a deviation from the actual speed at the start of the motor run, and there is a small steady state error

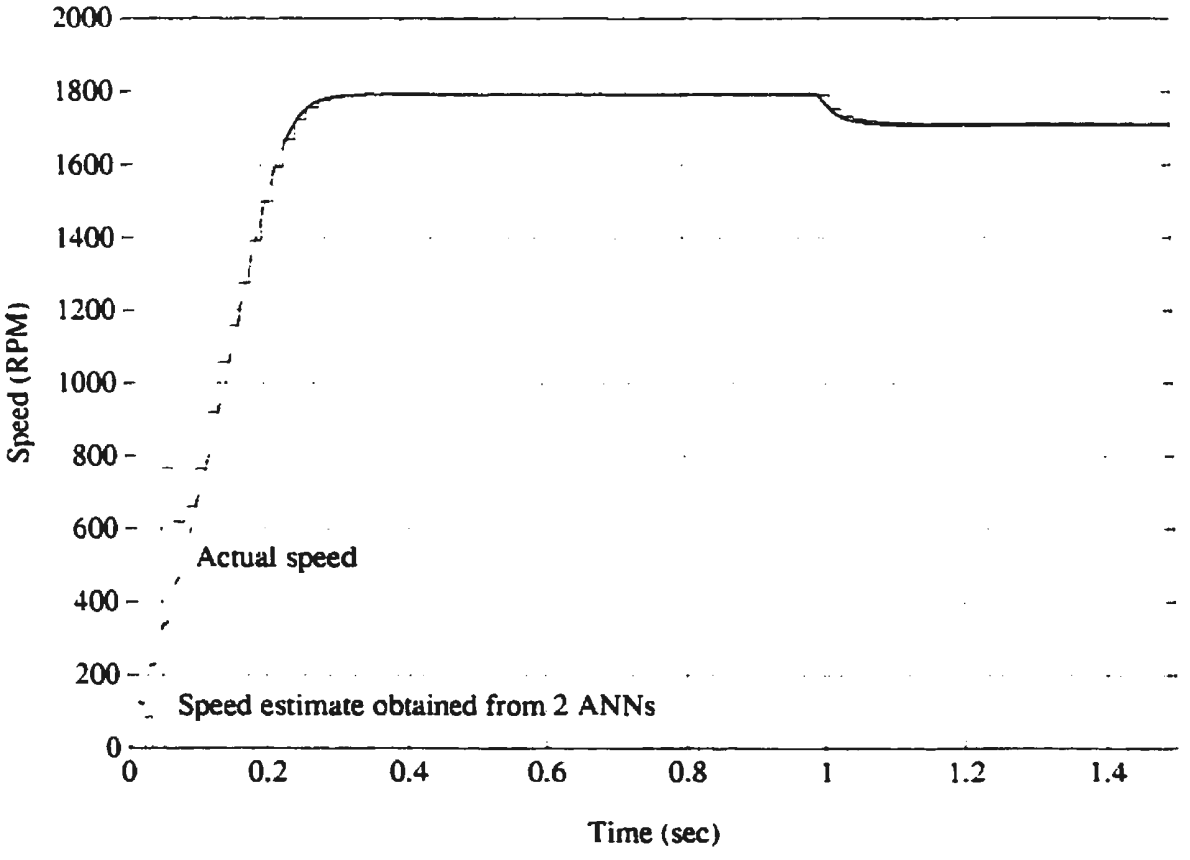


Figure 4.4: Actual and ANN recovered speed (method 1): Step change in load (10% to 150%) at t = 1.0 s

during the speed build-up.

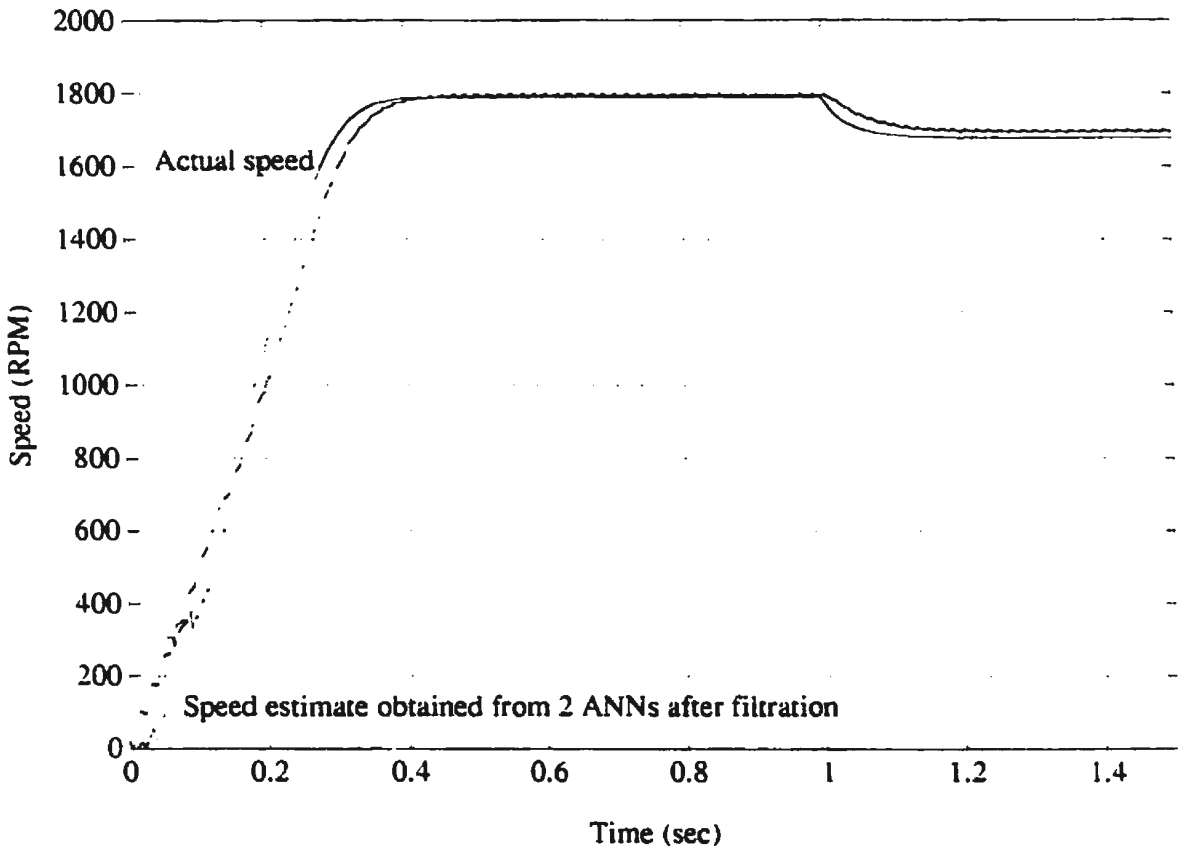


Figure 4.5: Actual and ANN recovered speed with inverter operation (method 1): Step change in load (10% to 150%) at t = 1.0 s

The estimate of the obtained speed is quite good in this method. However, since this scheme requires the computation of the numerator and denominator functions, a very small sampling time is required even though the speed feedback need not be computed very frequently. In the simulation study, the two ANNs compute their output every 50μsec and further increase in the sampling time results in loss of performance because

the peaks of the ANN outputs cannot be accurately determined. Also, the sizes of the two ANNs are quite large. Thus, this technique would require very high performance A/D converters and dedicated ANN hardware to implement it in real-time.

Another drawback of this scheme is that the actual motor speed is not used during training. For this reason, this scheme did not function acceptably in the presence of a vector controlled induction motor drive. Thus, it was decided to use equation (4.7) and see if it would be possible to obtain a more practical speed estimator.

4.2.2 Method 2: Using non-singular function

As seen in equation (4.7), there exists a non-singular function between the induction motor rotor speed and the stator quantities. Thus, if all the quantities which comprise the R.H.S. of equation (4.7) are given as inputs to an ANN, it should be able to estimate the speed, given sufficient training examples. Figure 4.6 shows the block diagram of the ANN speed estimator using this method. In this figure, the term T.D.L. represents a *Tapped Delay Line* which is used for obtaining the previous values of the inputs. The ANN is a four-layered 10-22-17-1 network. The inputs given to the ANN are $i_{qs}(k)$, $i_{ds}(k)$, $i_{qs}(k-1)$, $i_{ds}(k-1)$, $\int i_{qs}dt$, $\int i_{ds}dt$, $v_{qs}(k)$, $v_{ds}(k)$, $\int v_{qs}dt$, and $\int v_{ds}dt$. Here, unlike in method 1, training with random inputs was not performed, because applying random inputs to the induction motor would result in instability. Instead, the network was trained repeatedly on 16000 data vectors comprising of the starting response and step changes in load torque, frequency, etc. This set was given repeatedly 450 times resulting in a total of 7.5 million (7.5×10^6) iterations. Both the learning rate parameter and the momentum parameter were maintained at 0.2.

The output of the ANN had a lot of ripple in this case, and it had to be filtered in order to obtain a cleaner speed estimate. The response of the ANN to a direct

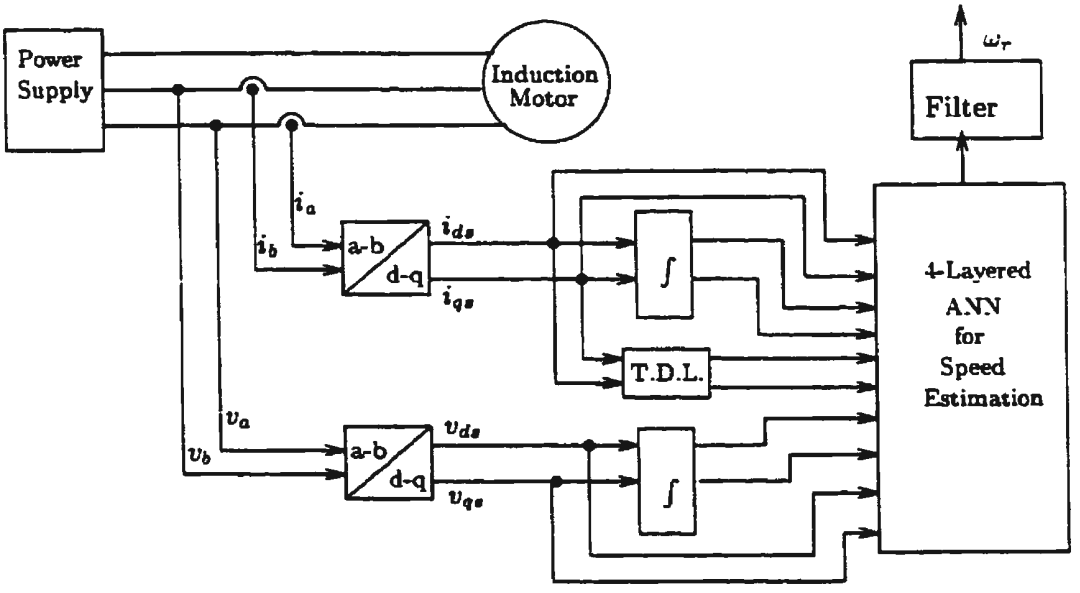


Figure 4.6: Block diagram of ANN speed estimator (method 2)

on-line start and step change in load torque is shown in Figure 4.7. It can be seen that this response is not as good as the response obtained in method 1 (section 4.2.1), even though there is a larger number of training examples. One of the main reasons for this is that training with random inputs was not performed in this case. Such a training spans a much larger area in the input space, with the effect that the ANN is able to generalize more effectively. However, if the training data is to be obtained from induction motor operation, giving absolutely random inputs to the motor would not be very effective, since the motor would not pick up speed. Thus, the ANN would be unable to generalize effectively. Another reason for the poorer performance is that the function to be estimated is more complex than the previous case.

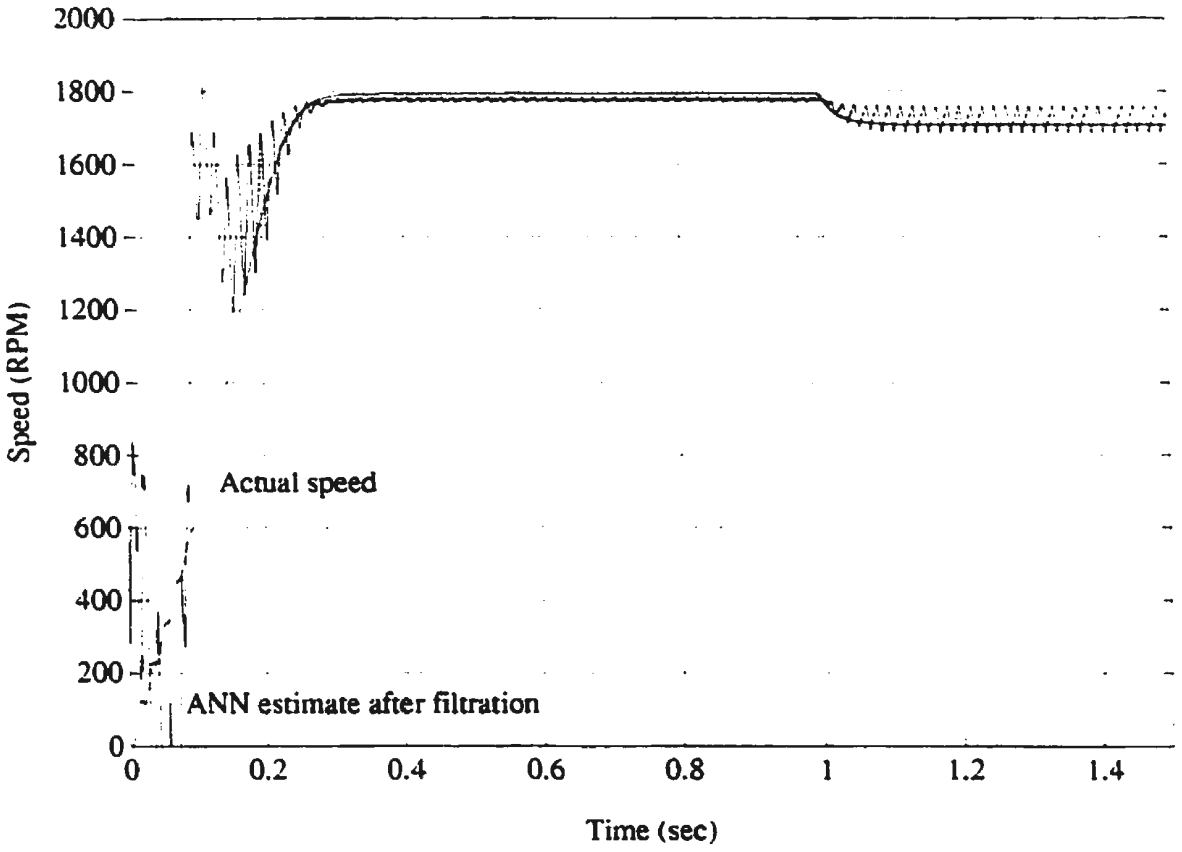


Figure 4.7: Actual and ANN recovered speed (method 2): Step change in load (10% to 150%) at t = 1.0 s

4.2.3 Method 3: Using non-singular function with magnitude and phase angle

The speed response of the ANN speed estimator using method 2 can be somewhat improved if the magnitudes and phase angles of all the quantities are given instead of their d-q components. This seems logical, because speed is obtained as the modulus of equation (4.7). The response of the ANN after filtration is shown in Figure 4.8. The filter uses the transfer function given in equation (4.12). As can be seen from the figure, the accuracy of the ANN output is improved even though it is still poorer than the estimation using two ANNs in section 4.2.1. However, one major advantage of this scheme and the one in method 2, is that the speed output can be computed with a much lower frequency as compared to the two ANN method. Thus, the hardware requirement would not be as stringent as in method 1. Unfortunately, methods 2 and 3 did not produce an acceptable output in the presence of an inverter, and thus these two methods are also not good candidates for a practical ANN speed estimator.

4.3 Importance of form in ANN training

The three speed estimators outlined in section 4.2 certainly prove the validity of the schemes presented, but as was seen earlier, the schemes cannot be used for obtaining a practical and real-time ANN speed estimator. In method 1, the two ANNs are large and have to be run at a very fast rate to approximate the complete sinusoid functions. Also, this method requires the induction motor parameters for the purpose of training, and this is undesirable. In methods 2 and 3, in spite of trying out numerous network sizes and extensive training, the networks were unable to produce a smooth speed estimate, and required filtration at the network output. Of bigger concern was the

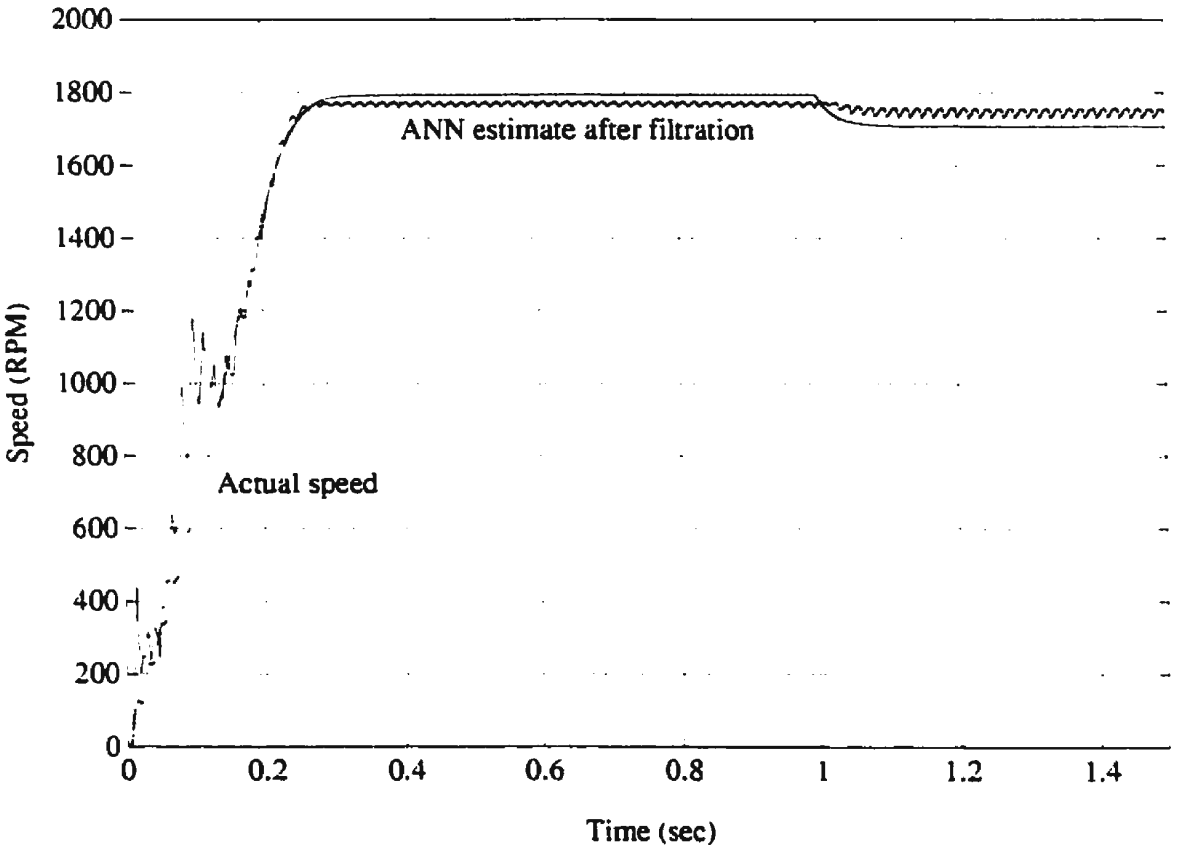


Figure 4.8: Actual and ANN recovered speed (method 3): Step change in load (10% to 150%) at t = 1.0 s

fact that in the presence of a vector controlled, inverter-fed drive, all the methods did not produce an acceptable output. Thus, it was felt that achieving the objective of experimental verification would require the development of another speed estimator which should be much smaller in size and should function more accurately, even in the presence of an inverter-fed drive.

With the above in mind, the whole idea of ANN speed estimation was revisited with the following fundamental question: "*Why does the ANN find it so difficult to obtain a mapping between the stator quantities and the speed, even though it has been established that there exists a functional relationship between the same?*" Some further investigation and thought led to the realization, that for an ANN speed estimator operating in steady state, the output would be a constant DC value, whereas the inputs would be sinusoidal in nature, though with a constant magnitude and frequency. For a different speed, the inputs would still be sinusoidal, but the magnitude and frequency would change. This implies that the ANN would need to extract the magnitude and frequency information from the sinusoidal inputs based on a few previous values, and then learn the relationship between the magnitude and frequency of the stator inputs and the speed. Thus, the sinusoidal nature of the inputs unnecessarily impedes the learning process, and modifying the form of the inputs would lead to a mapping which would be simpler for the ANN to learn. It should be noted that all the three methods outlined earlier have inputs which are not DC values. Method 3 uses magnitudes of the stator quantities, but it uses the phase angle also, which is not a DC quantity. Thus, the functional mapping remains quite complex and nonlinear.

This idea led to the development of a simple block which converts the instantaneous direct and quadrature axis sinusoidal quantities to instantaneous magnitude and frequency. Equations (3.10) to (3.14) are used to perform this conversion. This block uses the current and previous value of the d-q axis sinusoidal quantity to compute the

instantaneous magnitude and frequency. In the discrete case, the frequency is given by $f = \frac{\Delta\theta}{\Delta t}$, and it was decided that computing $\Delta\theta$ would suffice, since dividing by Δt would not provide any further useful information. This block will be called the "DQ to MF" or "DQ-MF" block. The block which performs the reverse transformation will similarly be called the "MF to DQ" or "MF-DQ" block. It should be noted that the speed estimator problem would not require the reverse transformation.

4.4 Method 4: Speed estimation using the DQ-MF block

This section outlines a simplified and more accurate speed estimator, which uses the "DQ-MF" block discussed in subsection 4.3 to obviate the problems associated with sinusoidal inputs. It was decided to try this scheme directly with an inverter-fed vector-controlled drive, by filtering the feedback quantities before using the "DQ-MF" block. Since it is impossible to filter out all the ripple, the instantaneous magnitude and $\Delta\theta$ also have ripples, and this necessitates the use of another filtration stage after the "DQ-MF" conversion. The block diagram of a 6-20-1 ANN estimator is shown in Figure 4.9. In this figure, T.D.L. stands for *Tapped Delay Line* which produces the previous value of the input. As can be seen from this figure, the current and voltage feedbacks for two phases are obtained and filtered to reduce the inverter ripple. They are then converted to d-q axis quantities and subsequently converted from DQ to MF. After this another filtration stage is required to smoothen the inputs for the ANN. The ANN receives the magnitude and $\Delta\theta$ values of the stator current along with one previous value of each. It also receives the magnitude of the stator voltage. Since the frequency of the stator voltage is the same as that of the stator current, $\Delta\theta$ value of the stator voltage would not provide any new information. Thus it is not used, making the ANN more

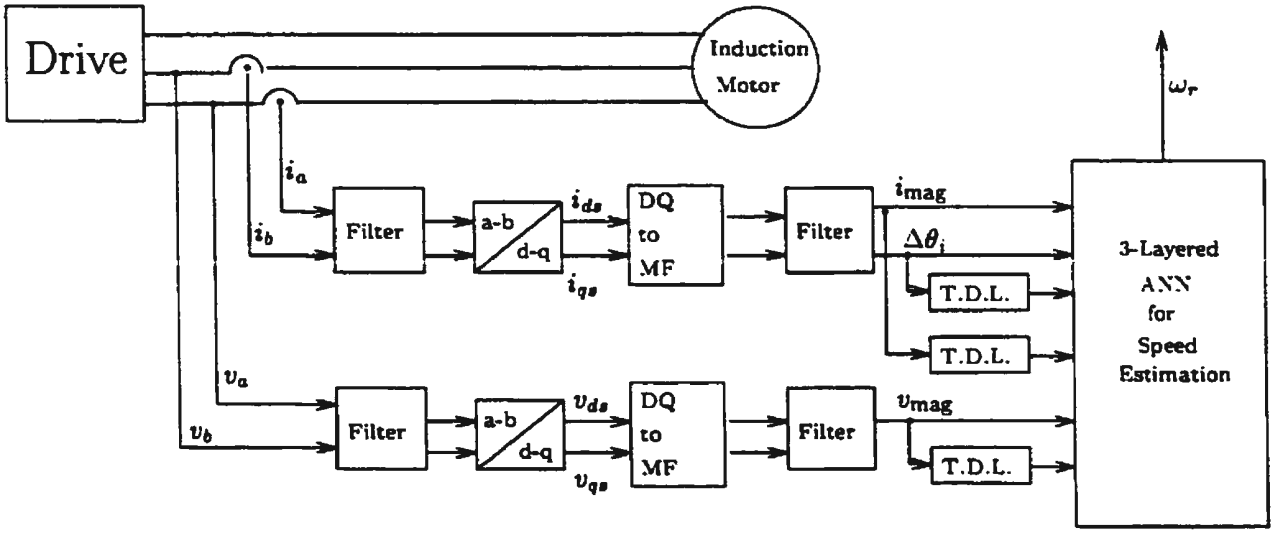


Figure 4.9: ANN speed estimator using the DQ-MF block (method 4)

compact. Also, like methods 2 and 3 and unlike method 1, the sampling requirements of this ANN speed estimator are not very stringent. However, it should be noted that the size of this ANN is significantly smaller than the two ANNs used in method 1 or the networks used in methods 2 and 3.

In section 4.2 it was noted that, for the purpose of training, no data need be stored on the disk, and the whole system, including the induction motor and the ANN, can be run on-line to provide training data to the ANN. Subsequent to that work, it was discovered that it is significantly more advantageous to store training data on the disk, and then provide the ANN with randomly shuffled data points during training. This not only increases the efficacy of the training process but also enables the computation of the "sum squared error" (SSE) per epoch, which is a useful indicator of the training progress. Thus, the training data for the speed estimator using method 4 has been stored prior to the start of ANN training.

For training, the induction motor was run in simulation with a vector controller and inverter setup, and a data set of training vectors was obtained. The network was trained with different learning rates, and different number of epochs. The chosen network was trained for 47 epochs with 0.03 as the output neuron learning rate and 0.50 as the learning rate of every other neuron. Figure 4.10 shows the performance of the ANN speed estimator with a step change in speed reference at t = 1.0 s and a step change in load torque from (9.3% to 93.0%) at t = 2.0 s. This speed estimator functions well under vector control conditions and in the presence of an inverter. Also, it does not require a filter at the output. Thus, this speed estimator is a very good candidate for experimental verification. The SSE for the ANN during training is shown in Figure 4.11.

4.5 Summary

In this chapter, four techniques for speed estimation of induction motors using artificial neural networks have been outlined. The mathematical model of the induction motor is considered and expressions for the rotor speed are obtained. Two of the expressions obtained have singularities and thus, ANNs cannot be used to obtain the speed directly by supplying the same inputs as the functions. A method is proposed in which two ANNs are trained to approximate the numerator and denominator functions in the speed expression. By training such ANNs and using a filter to avoid singular points, the speed can be recovered with a high degree of accuracy. In the second method, the two expressions with singularities are combined to obtain a single expression which does not have singularities for regular induction motor operation. A single ANN is trained to obtain the speed in this case by giving it all the quantities which act as input to the expression in equation (4.7). In the third method, which is a modification of the second,

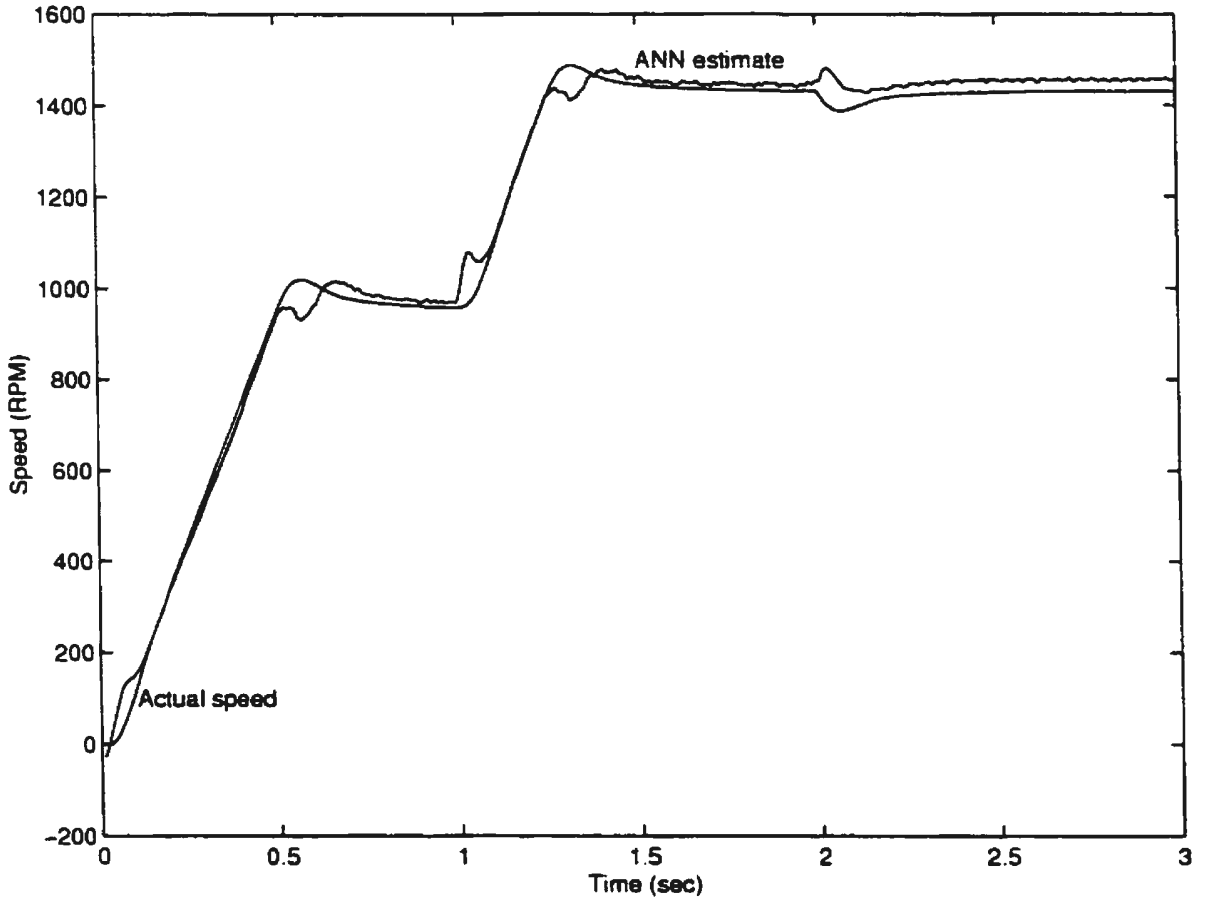


Figure 4.10: Actual and ANN recovered speed (method 4): Step change in speed reference at t = 1.0 s and step change in load torque at t = 2.0 s

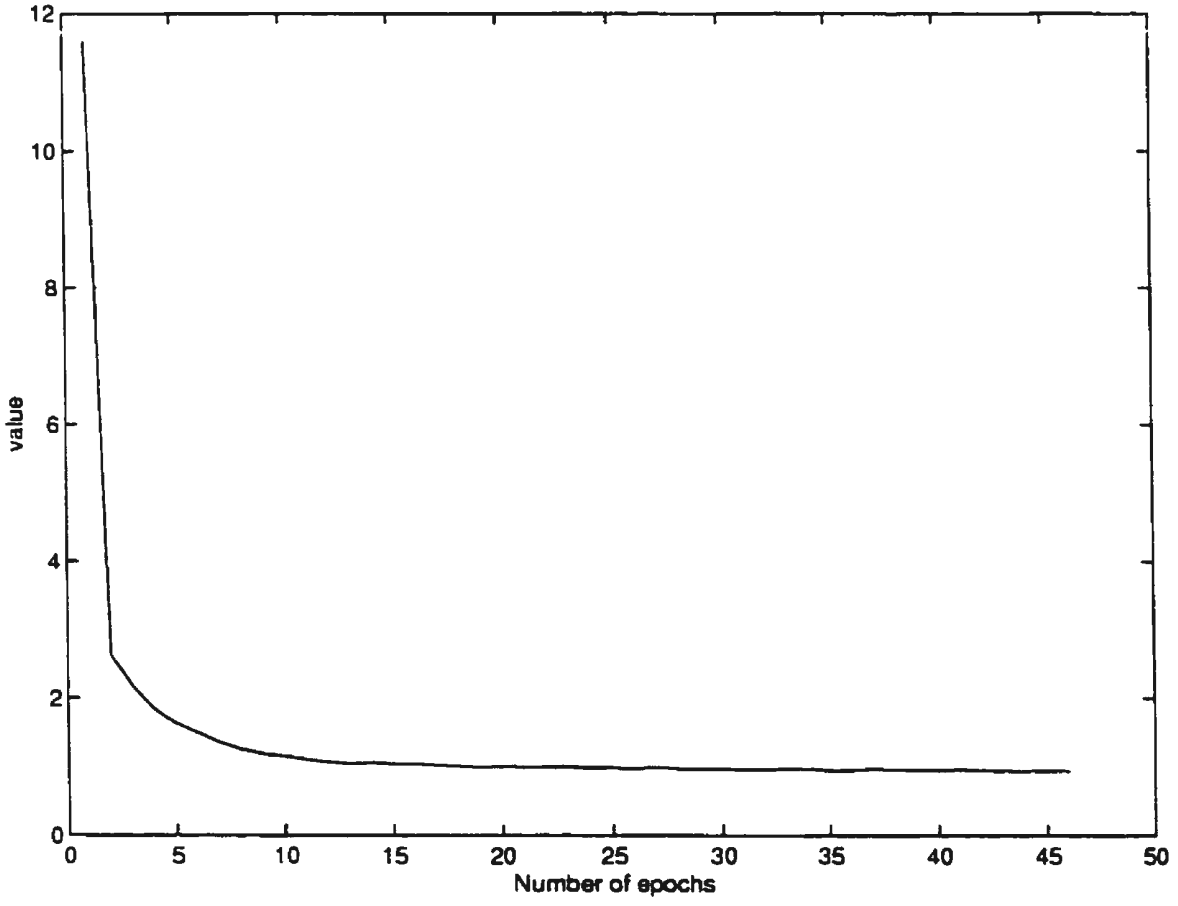


Figure 4.11: Sum squared error during ANN training (method 4)

the inputs are presented in a different form, i.e. using magnitude and phase angle, and this results in a slight improvement in performance. These three methods were found unsuitable for real-time implementation, and a fourth method was developed in which the inputs to the ANN are transformed in a way which simplifies the mapping that the ANN is required to learn. This method was found to be suitable and functions well, even in the presence of a vector controlled, inverter-fed drive. A real-time experimental implementation of the fourth method will be outlined in chapter 6.

Chapter 5

ANN Control of Induction Motor

It was seen in chapter 2 that while considerable progress has been made in the application of ANNs to induction motor drives, complete control of an induction motor using ANNs still eludes researchers. For ANN control of induction motor to become commercially feasible and economically viable, ANNs must provide a useful alternative to existing control strategies. This means that researchers must come up with strategies to implement induction motor control using ANNs, without the help of conventional controllers. Also, the resulting strategies must not be computationally overwhelming for the available hardware to implement in real-time.

The main thrust of this work was to move in this direction and try to come up with an induction motor control strategy using just one ANN. This chapter focusses on the problem of ANN control of induction motor drives, and demonstrates, for probably the first time, a strategy for complete control of induction motor using a single ANN. This ANN is able to control the induction motor in a satisfactory way, after being off-line trained to mimic a vector controller. A scheme for on-line training of this ANN is also presented, and this leads to improved steady state response and robustness in the

presence of motor parameter variations.

5.1 Issues involved with ANN control of induction motor

Most of the control theory developed so far deals with linear time-invariant systems, and powerful methods for designing controllers for such systems are currently available. However, as applications become more complex, the processes to be controlled are increasingly characterized by uncertainty in the system model, non-linearities, presence of noise and the effects of having distributed sensors and actuators with their associated delays and other problems. One approach used for handling a non-linear system has been to linearize it around an equilibrium point, and then use the well established linear control theory to study issues like stability, controllability and observability, and design controllers to function in an approximate linear region around the equilibrium point. Both *single-input single-output* (SISO) and *multiple-input multiple-output* (MIMO) systems have been studied using this approach, in which the non-linear dynamical system to be controlled can be described by the state equations

$$x(k+1) = f[x(k), u(k)], f(0,0) = 0 \quad (5.1)$$

$$y(k) = h[x(k)], h(0) = 0 \quad (5.2)$$

where $u(k)$, $y(k) \in \Re^m$ and $x(k) \in \Re^n$ and represent the input, output and state vectors. From a purely mathematical point of view, the precise control of a non-linear dynamical system is a formidable task [61]. It becomes substantially more difficult when uncertainty is also present in the system.

In most cases where ANNs have been used in the control of induction motors, on-line training has been preferred. On-line training has the potential to adapt to changing

motor parameters, but it is computationally very expensive, and it is very difficult to run an average sized ANN in real-time with on-line training.

As indicated earlier, Kung et al [48] have used a *two degree of freedom* controller (2DOF) to control an induction motor and also use an ANN to tune the parameters of this controller on-line. One of the most rigorous works on ANN control of induction motor has been done by Wishart and Harley [45], and here too we can see the presence of a PI controller which receives the speed error as an input and produces the magnitude of the reference currents needed for the current controlled induction machine. The control ANN produces only the frequency of the reference currents. Thus the ANN only partially controls the motor.

One way to control a plant using ANNs is to train the ANN off-line to mimic an existing controller. This implies that the ANN must have as input, all the quantities that are input to the existing controller (with a suitable number of previous values), including the reference value. The ANN is trained off-line to produce the same outputs as the controller and after sufficient training, the ANN should be able to replace the controller. A block diagram of this scheme is shown in Figure 5.1. Since it is impractical

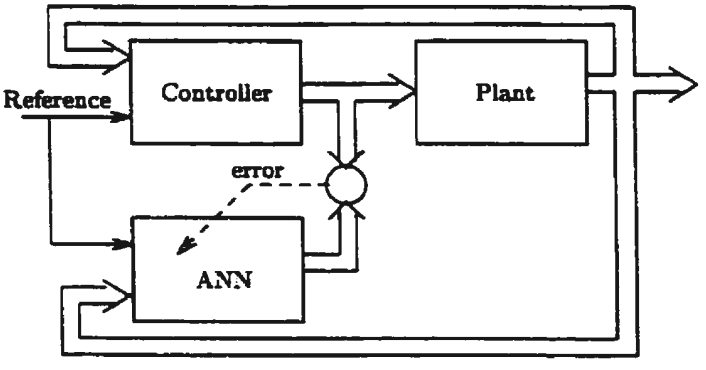


Figure 5.1: Training of ANN for controller mimicing

to provide all possible combinations of reference and load torque change to the ANN as training data, the ANN's generalization property will have to be relied upon to learn the important trends. It is a well known property of backpropagation nets that too much training increases the accuracy of the network on the given data set, but the network loses its ability to generalize effectively. On the other hand, a network that can generalize well will not produce extremely accurate results. Also, it must be remembered that a network will always have a small error in its output, no matter how well it is trained. Thus, it looks very likely that the steady state error in an off-line trained ANN may not be eliminated completely.

A second problem that is present in induction motor control is the inability to model the disturbance or load torque. As seen in Figure 5.2, the load torque acts as an external and unknown input. Some researchers have simplified the problem by

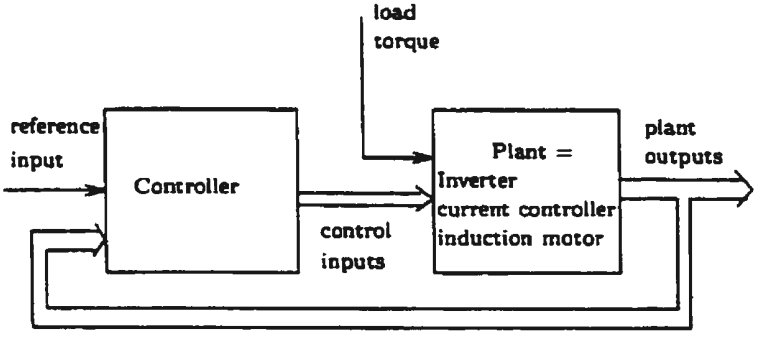


Figure 5.2: Induction motor control model

assuming that the load torque depends only on the speed, which is valid for pump and fan loads [45]. However, this factor severely limits the applicability of the drive. To be more general, the ANN controller must be able to handle unknown step changes in load torque.

One of the chief difficulties in controlling an induction motor with an ANN is that an induction motor requires sinusoidal inputs. This usually means that the ANN must produce smooth sine wave references which vary in magnitude and frequency to implement the control. For example, a vector controller produces stationary frame direct and quadrature axis reference voltages or currents, which are subsequently converted to three-phase references. If an ANN has to mimic the vector controller, it must also produce the same outputs. However, producing direct and quadrature axis references which are sinusoidal in shape and exactly 90° apart is no easy task. Thus it is very difficult to even run the induction motor at any speed using an ANN, let alone control the speed.

A second factor which contributes to this difficulty is the inverter which is required to supply the induction motor with three-phase voltages. The switching property of the inverter creates a lot of ripple and non-linearity which would severely hamper the operation of the ANN, since the ANN would require current or voltage feedback. This strategy was tried out in simulation, whereby an ANN was trained to mimic a vector control algorithm block, but failed to run the induction motor even after extensive training.

5.2 ANN based direct adaptive control of induction motor

Since the induction motor dynamic model is well known, it makes sense to use a control strategy which incorporates this knowledge, rather than treating the induction motor as a black box. Direct adaptive control offers a way of doing this, by using the plant Jacobian in ANN training. A block diagram of this strategy is shown in Figure 5.3 [62].

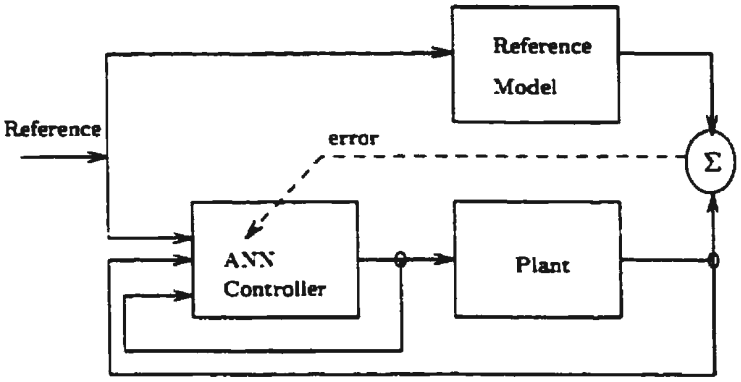


Figure 5.3: Direct adaptive control using ANN

For induction motor control, the plant consists of the induction motor and the inverter along with the current controller. Thus, the plant has just two inputs, which are the stationary frame, d-q axis reference currents. For simplicity, the inverter and current controller dynamics can be ignored, because in an ideal case the current controller and the inverter just supply the induction motor with the desired currents.

To derive the adaptive control strategy with ANNs, we have to consider the speed error which is given by

$$e_{\omega r} = \omega_r^* - \omega_r \tag{5.3}$$

where ω_r^* is the reference speed and ω_r is the actual speed. A *cost function* is computed from the speed error, and this can take various forms, though a standard form used is

$$\mathcal{J} = \frac{1}{2} e_{\omega r}^2 \tag{5.4}$$

For the output layer neurons, the weight increment is given by

$$\Delta w_{ji} = -\eta \left(\frac{\partial \mathcal{J}}{\partial w_{ji}} \right)$$

$$= -\eta(\frac{\partial \mathcal{J}}{\partial e_{\omega_r}})(\frac{\partial e_{\omega r}}{\partial u_j})(\frac{\partial u_j}{\partial w_{ji}}) \quad (5.5)$$

where η is the *learning rate*, w_{ji} is the i^{th} weight of the j^{th} neuron in the output layer and u_j is the j^{th} input to the plant.

Equation (5.5) can be further simplified as

$$\Delta w_{ji} = -\eta(e_{\omega_r})(-\frac{\partial \omega_r}{\partial u_j})(\frac{\partial u_j}{\partial w_{ji}}) \quad (5.6)$$

$$= -\eta(e_{\omega_r})(-\frac{\partial \omega_r}{\partial u_j})\Phi'(\text{net}_j)(y_i) \quad (5.7)$$

$$= \eta(\delta_j)(y_i) \quad (5.8)$$

where Φ represents the neuron activation function and δ_j is the *local gradient* given by

$$\delta_j = e_{\omega r}(\frac{\partial \omega_r}{\partial u_j})\Phi'(\text{net}_j) \quad (5.9)$$

and y_i is the output of the ith neuron. Once the local gradient for the output neurons is known, regular backpropagation can be used for computing the weight increments for the hidden layer neurons.

The main idea behind on-line training using direct adaptive control is to compute the local gradient, using the plant Jacobian $(-\frac{\partial \omega_r}{\partial u_j})$. This requires a knowledge of the plant dynamic model, in particular the dependence of the speed on the stationary frame d-q axis stator currents. From the earlier work on speed estimation, we have an expression for the induction motor speed of the form

$$\omega_r = \frac{1}{P}\sqrt{\frac{{g_1}^2 + {g_2}^2}{{f_1}^2 + {f_2}^2}} \quad (5.10)$$

where,

$$g_1 = -[\sigma^2 i'_{ds} - R_r L_s i_{ds} + R_r \int v_{ds} - R_r R_s \int i_{ds} + L_r v_{ds} - R_s L_r i_{ds}] \quad (5.11)$$

$$g_2 = [\sigma^2 i'_{qs} - R_r L_s i_{qs} + R_r \int v_{qs} - R_r R_s \int i_{qs} + L_r v_{qs} - R_s L_r i_{qs}] \quad (5.12)$$

$$f_1 = \sigma^2 i_{qs} + L_r \int v_{qs} - R_s L_r \int i_{qs} \quad (5.13)$$

$$f_2 = \sigma^2 i_{ds} + L_r \int v_{ds} - R_s L_r \int i_{ds} \quad (5.14)$$

In the above equations, $\sigma^2 = {L_m}^2 - L_r L_s$, where L_m is the magnetizing inductance and L_r, L_s are the rotor and stator inductances respectively. Also, i'_{ds} and i'_{qs} refer to the first time derivatives of i_{ds} and i_{qs}, i.e. $\frac{di_{ds}}{dt}$, $\frac{di_{qs}}{dt}$.

We need to obtain expressions for $\frac{\partial \omega_r}{\partial i_{ds}}$ and $\frac{\partial \omega_r}{\partial i_{qs}}$ for adaptive control because it is assumed that $i_{ds} \approx i^*_{ds}$ and $i_{qs} \approx i^*_{qs}$, assuming a good current controller and inverter combination.

Equation (5.10) can be represented as $\omega_r = \frac{1}{P}\sqrt{\frac{N}{D}}$, where both N and D are functions of i_{ds} and i_{qs}, given by $N = {g_1}^2 + {g_2}^2$ and $D = {f_1}^2 + {f_2}^2$. Applying the chain-rule differentiation, we get

$$\frac{\partial \omega_r}{\partial i_{ds}} = \frac{1}{2P}\sqrt{\frac{D}{N}}\frac{1}{D^2}(D\frac{\partial N}{\partial i_{ds}} - N\frac{\partial D}{\partial i_{ds}}) \tag{5.15}$$

$$\frac{\partial \omega_r}{\partial i_{qs}} = \frac{1}{2P}\sqrt{\frac{D}{N}}\frac{1}{D^2}(D\frac{\partial N}{\partial i_{qs}} - N\frac{\partial D}{\partial i_{qs}}) \tag{5.16}$$

Continuing the chain-rule process,

$$\frac{\partial N}{\partial i_{ds}} = 2g_1\frac{\partial g_1}{\partial i_{ds}} + 2g_2\frac{\partial g_2}{\partial i_{ds}} = 2g_1\frac{\partial g_1}{\partial i_{ds}} \tag{5.17}$$

$$\frac{\partial N}{\partial i_{qs}} = 2g_1\frac{\partial g_1}{\partial i_{qs}} + 2g_2\frac{\partial g_2}{\partial i_{qs}} = 2g_2\frac{\partial g_2}{\partial i_{qs}} \tag{5.18}$$

$$\frac{\partial D}{\partial i_{ds}} = 2f_1\frac{\partial f_1}{\partial i_{ds}} + 2f_2\frac{\partial f_2}{\partial i_{ds}} = 2f_2\frac{\partial f_2}{\partial i_{ds}} \tag{5.19}$$

$$\frac{\partial D}{\partial i_{qs}} = 2f_1\frac{\partial f_1}{\partial i_{qs}} + 2f_2\frac{\partial f_2}{\partial i_{qs}} = 2f_1\frac{\partial f_1}{\partial i_{qs}} \tag{5.20}$$

Thus, we need to find expressions for $\frac{\partial g_1}{\partial i_{ds}}$, $\frac{\partial g_2}{\partial i_{qs}}$, $\frac{\partial f_1}{\partial i_{qs}}$ and $\frac{\partial f_2}{\partial i_{ds}}$.

1. $\frac{\partial g_1}{\partial i_{ds}}$: g_1 is a function of i_{ds}, i'_{ds} and $\int i_{ds}$ and thus,

$$\frac{\partial g_1}{\partial i_{ds}} = [L_r R_s + R_r L_s] + (\frac{\partial g_1}{\partial i'_{ds}})(\frac{\partial i'_{ds}}{\partial i_{ds}}) + (\frac{\partial g_1}{\partial i_{dsi}})(\frac{\partial i_{dsi}}{\partial i_{ds}}) \tag{5.21}$$

where $i_{dsi} = \int i_{ds}$

$$\frac{\partial g_1}{\partial i'_{ds}} = -\sigma^2 \tag{5.22}$$

$$\frac{\partial i'_{ds}}{\partial i_{ds}} = (\frac{\partial i'_{ds}}{\partial t})/(\frac{\partial i_{ds}}{\partial t}) \tag{5.23}$$

$$= (\frac{\partial i'_{ds}}{\partial t})/(i'_{ds}) \tag{5.24}$$

$$\frac{\partial g_1}{\partial i_{dsi}} = R_s R_r \tag{5.25}$$

$$\frac{\partial i_{dsi}}{\partial i_{ds}} = (\frac{\partial i_{dsi}}{\partial t})/(\frac{\partial i_{ds}}{\partial t}) \tag{5.26}$$

$$= \frac{i_{ds}}{i'_{ds}} \tag{5.27}$$

2. $\frac{\partial g_2}{\partial i_{qs}}$: g_2 is a function of i_{qs}, i'_{qs} and $\int i_{qs}$

$$\frac{\partial g_2}{\partial i_{qs}} = -[L_r R_s + R_r L_s] + (\frac{\partial g_2}{\partial i'_{qs}})(\frac{\partial i'_{qs}}{\partial i_{qs}}) + (\frac{\partial g_2}{\partial i_{qsi}})(\frac{\partial i_{qsi}}{\partial i_{qs}}) \tag{5.28}$$

$$\frac{\partial g_2}{\partial i'_{qs}} = \sigma^2 \tag{5.29}$$

$$\frac{\partial i'_{qs}}{\partial i_{qs}} = (\frac{\partial i'_{qs}}{\partial t})/(\frac{\partial i_{qs}}{\partial t}) \tag{5.30}$$

$$= (\frac{\partial i'_{qs}}{\partial t})/(i'_{qs}) \tag{5.31}$$

$$\frac{\partial g_2}{\partial i_{qsi}} = -R_s R_r \tag{5.32}$$

$$\frac{\partial i_{qsi}}{\partial i_{qs}} = (\frac{\partial i_{qsi}}{\partial t})/(\frac{\partial i_{qs}}{\partial t}) \tag{5.33}$$

$$= \frac{i_{qs}}{i'_{qs}} \tag{5.34}$$

where $i_{qsi} = \int i_{qs}$

3. $\frac{\partial f_1}{\partial i_{qs}}$: f_1 is a function of i_{qs} and $\int i_{qs}$

$$\frac{\partial f_1}{\partial i_{qs}} = \sigma^2 + (\frac{\partial f_1}{\partial i_{qsi}})(\frac{\partial i_{qsi}}{\partial i_{qs}}) \tag{5.35}$$

As in the previous case,

$$\frac{\partial f_1}{\partial i_{qsi}} = -L_r R_s \tag{5.36}$$

$$\frac{\partial i_{qsi}}{\partial i_{qs}} = \frac{i_{qs}}{i'_{qs}} \tag{5.37}$$

4. $\frac{\partial f_2}{\partial i_{ds}}$: f_2 is a function of i_{ds} and $\int i_{ds}$

$$\frac{\partial f_2}{\partial i_{ds}} = \sigma^2 + (\frac{\partial f_2}{\partial i_{dsi}})(\frac{\partial i_{dsi}}{\partial i_{ds}}) \tag{5.38}$$

$$\frac{\partial f_2}{\partial i_{dsi}} = -L_r R_s \tag{5.39}$$

$$\frac{\partial i_{dsi}}{\partial i_{ds}} = \frac{i_{ds}}{i'_{ds}} \tag{5.40}$$

From the above equations, we can see that we have all the terms required to compute equations (5.17) ... (5.20), and hence obtain expressions for equations (5.15) and (5.16). Thus, the plant Jacobian can be computed for direct adaptive control using ANNs. However, as is quite obvious from the above, this method is computationally quite involved. Furthermore, this method was tried out in simulation, but the system became unstable and the induction motor was unable to run at a sustained speed and kept oscillating around zero speed. Thus, this method was not investigated further, though it is being reported for the sake of completeness.

5.3 Off-line control of induction motor using ANN

This section discusses a hitherto unreported scheme of induction motor control — one using a single ANN for control with only off-line training. To date, the author has not come across a single satisfactory scheme in which complete control of an induction

motor has been executed using only one or more ANNs and without any conventional controller. In the relatively few papers that discuss induction motor control using ANNs, the authors have used the ANNs to assist an existing controller, or at best, implement a portion of the control. In the off-line control strategy which is considered here, the ANN is trained to mimic a rotor field-oriented control strategy, which produces stationary frame direct and quadrature axis current references at its output. These references are converted to three-phase quantities by a simple transformation. Thus, for all practical purposes, the control problem is converted to a system identification problem.

The main benefit of using off-line control is that for running the system in real-time, the computationally intensive backpropagation algorithm does not have to be executed. This results in a major saving in execution time, thereby enabling bigger networks to be implemented. The next benefit is that the network can, in theory, be trained to model a very efficient control algorithm like field-oriented control. In schemes which backpropagate the speed error for on-line training, the resultant control may not be as efficient as field-oriented control and might be closer to simple PI based control or V/f control.

It has been pointed out in section 4.3 that it is very important to present the inputs to an ANN in a form that simplifies the mapping that the ANN is required to learn. In the case of the speed estimator discussed in section 4.4, the sinusoidal inputs were transformed from DQ to MF to simplify the mapping for the ANN. For the control problem, both the inputs and outputs are sinusoidal, even in steady state. However, it was felt that here also the mapping could be simplified if this sinusoidal nature could be suppressed, since control is actually executed by varying the magnitude and frequency of the reference outputs, depending on the magnitude and frequency of the inputs to the controller. Thus, in this work the following block diagram is used for

training purposes: As seen in this block diagram, the outputs of the vector controller,

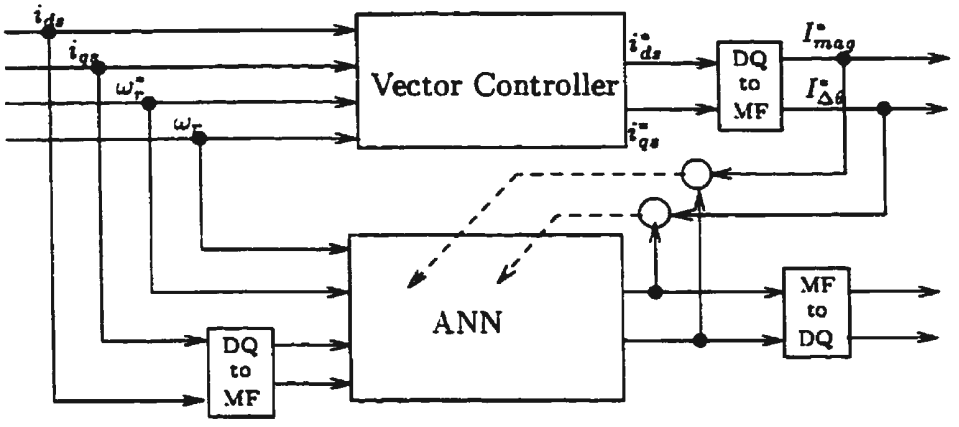


Figure 5.4: ANN training for off-line control

which are the direct and quadrature axis stator current references, are converted from DQ to MF and then used as desired outputs for ANN training. The ANN receives all the signals that the vector controller receives. However, to facilitate training, the current feedback is converted from DQ to MF. After training, the ANN is run in the feedforward mode and its outputs are converted from MF to DQ and then handed over to the inverter-motor block.

5.3.1 Training considerations

Since an inverter forms a necessary component of a drive system, it was decided to implement a three-phase current amplifier for reasons mentioned in section 4.2. This is simply a block which supplies the motor with three-phase currents identical to the current references. This block is implemented as the class *current_amplifier* in the simulator and has been discussed in section 3.3. With this arrangement, the current controller and the inverter can be bypassed, because the three-phase current amplifier

acts like an ideal current source which injects currents into the induction motor. In effect, it functions like a power amplifier, boosting the power of the reference signals and then supplying the induction motor with these boosted signals.

The next thing to consider is that the ANN must receive all the inputs that the vector controller receives, plus some previous values. The vector controller is first run with the current amplifier to collect data for training the neural network and a block diagram of this process is shown in Figure 5.5. If the ANN receives current feedback,

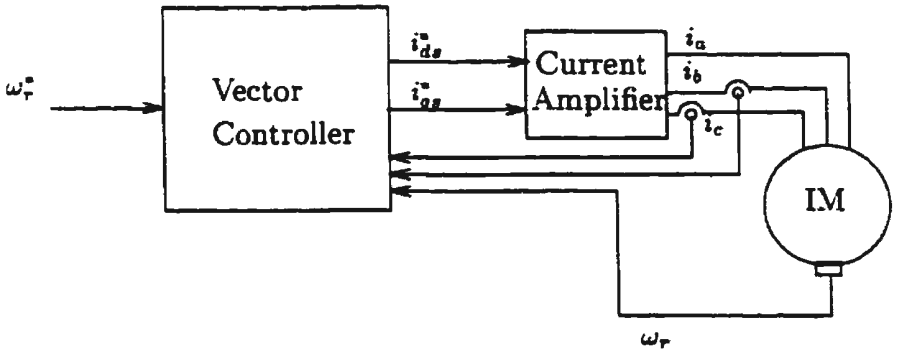


Figure 5.5: Data collection for ANN training

like a vector controller, then the network would have to be a feedback network and not simply a feedforward one, because the network output would be fed back to the input after a unit delay. Thus, the network would have to be trained in the *series-parallel* mode and, after training, run in the parallel mode. It is known that the series-parallel mode is preferrable to the parallel mode [50], and it would be desirable to be able to run the ANN in a series-parallel mode after training. This would entail avoiding the output feedback. This, however, is not possible in most cases, since, after training, the ANN no longer has access to the training data (or the "right answers") and hence the ANN outputs must be fed back after a delay to obtain the previous values of the

outputs. This would make it a parallel system while running, increasing the chances of instability, because of the feedback present.

In the current reference model of the induction motor, the motor receives current inputs from a current source and the stator voltages are determined by the currents flowing into the motor. Thus, the stator voltages should contain information about the stator currents. This, in turn, implies that stator voltages can be provided as inputs to the ANN instead of stator currents, making the ANN a strictly feedforward one.

Something else to consider is that whereas $\Delta\theta$ attains both positive and negative values, the magnitude is always positive. Thus, the output neuron which produces the magnitude of the reference current could have an activation function which does not permit negative values, thereby reducing the range of possible incorrect values produced by the network. The other output neuron which produces the $\Delta\theta$ of the current reference should have a different activation function such that negative outputs are permitted.

For training purposes, it should be noted that a single output network is usually easier to train than a multiple output network, because in a multiple output network, there is interference from other outputs when the error is being backpropagated. Thus, instead of having a dual output neuron, it might be more desirable to have two single output networks. However, having two ANNs in the simulation would imply that a backpropagation algorithm block would be required for each of them, and, if they both receive the same inputs, then these would have to be supplied individually to both, increasing the program length and complexity.

5.3.2 Fully-connected network training

The number of weights in a three-layer fully-connected network is given by

$$N_{\text{weights}} = (I + O)H \tag{5.41}$$

where I is the number of ANN inputs, H is the number of neurons in the hidden layer and O is the number of ANN outputs.

A 13-75-2 ANN was tried out for the purpose of control. The magnitude output neuron has a "LOGSIG" activation function for reasons outlined earlier in subsection 5.3.1. and this function is given by

$$f_{\text{LOGSIG}}(\text{net}) = \frac{1}{1 + e^{-\beta \text{net}}} \tag{5.42}$$

Every other neuron in the network has a "TANSIG" activation function given by

$$f_{\text{TANSIG}}(\text{net}) = \frac{1 - e^{-\beta \text{net}}}{1 + e^{-\beta \text{net}}} \tag{5.43}$$

The parameter β was chosen to be 0.8 for the 2 output neurons and 0.9 for every other neuron. This network structure was chosen after a lot of trial and error, in an attempt to optimize the performance. It should be noted that it is not possible to implement this kind of a network using a commercially available ANN simulator like the MATLAB neural network toolbox, because of the possibility of having different neurons within the same layer, each with a different learning rate. The inputs to the ANN are

- Voltage magnitude and $\Delta\theta$ and 2 previous values of each
- Speed feedback and 3 previous values
- Actual speed error (ramp generator output minus speed feedback) and 1 previous value

- Speed reference

This set of inputs was chosen carefully, based on the approximate NARMAX model of the vector controller. Usually, in ANN training, there are no fixed rules for choosing the network structure, learning rates, number of training epochs, training data set and so on. The researcher must try different combinations and use various heuristics and experience in an attempt to come up with an optimum network, which is efficient in terms of size and performance. The structure of the above network is by no means the best one, and theoretically, one should be able to get a network which approximates the desired function arbitrarily closely.

The data set was obtained by running the induction motor under vector control with step changes to reference speed and load torque. The induction motor parameters are given in Table 5.1 [41]. An important point to note is that the ANN itself does not need any machine parameters for the purpose of training, because it observes only the motor inputs and outputs. The machine parameters have been used only for the purpose of simulating the induction motor.

The vector control algorithm runs at 500μs sampling interval, and the rest of the simulation runs at a 10μs sampling interval. To nullify the effects of inverter ripple, a current amplifier block was used instead of the current controller and inverter combination. Apart from the speed reference, the output of a *ramp generator* has also been provided to the ANN to improve learning. This ramp generator is identical to the block which ramps up the speed reference inside the vector controller. A block diagram of the scheme is shown in Figure 5.6.

For generating training data, the switch "S" is thrown to the top position, connecting the vector controller to the current amplifier. After the data is collected, training of the ANN is performed off-line, and the switch "S" is then thrown to the bottom

Table 5.1: Induction motor parameters used in motor control simulation studies

Parameter	Symbol	Value
Power Rating		2.0 kW
Voltage		208V
Connection type		Y
Stator Resistance	R_s	0.60 Ω
Rotor Resistance	R_r	0.40 Ω
Stator Inductance	L_s	72.7 mH
Rotor Inductance	L_r	72.7 mH
Magnetizing Inductance	L_m	69.8 mH
Moment of Inertia	J	0.0357 $N-m^2$
Damping Coefficient	B	0.0030 $N-m^2/s$
Pole Pairs	P	2

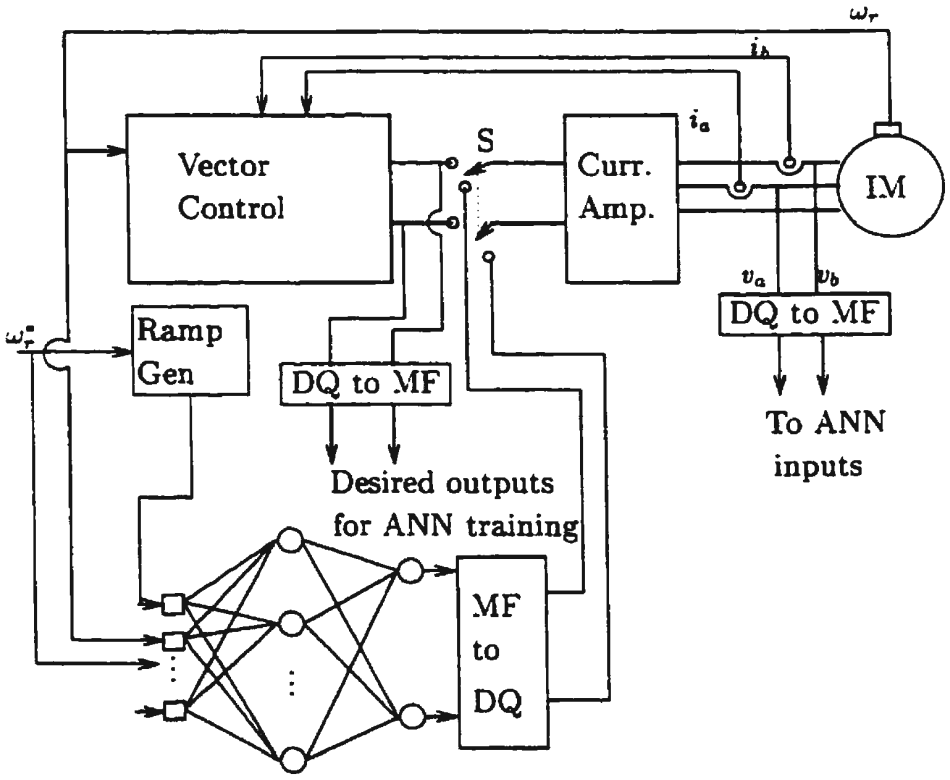


Figure 5.6: Current amplifier based scheme for ANN control of induction motor using voltage feedback

position, handing over the reins to the ANN and disconnecting the vector controller. It must be remembered that the inputs and outputs have to be normalized to a range of ± 1 before being handed over to the ANN.

The ANN was trained for 50 epochs on a training data set with 1690 points which were randomly shuffled during training. The learning rate of all neurons was 0.20. The performance of the ANN controller is shown in Figure 5.7. Here, the reference speed undergoes a step change at t = 1.0 s, and there is a step change in load torque at t = 2.0 s. As can be seen from the figure, the performance of the ANN is unacceptable as a controller. The SSE during ANN training and the actual ANN outputs are shown in Figures 5.8 and 5.9 respectively.

5.3.3 Split-ANN training

One of the main problems with using a fully-connected ANN is that the training for the two outputs cannot be performed independently, thereby leading to less effective learning. To circumvent this problem and still attain the convenience of using just one ANN, a *dual output split-ANN* is proposed as shown in Figure 5.10. This network is a special kind of *sparse ANN* architecture. As can be seen from the figure, both the subnetworks can be trained separately, by controlling their learning rates individually. For example, if it is desired to train just one subnetwork, then the learning rates of all the neurons in the other subnetwork can be set to zero, thereby preventing any modification of the weights. Also, it can be seen from the figure, that both the subnetworks may or may not have the same set of inputs. Thus, for all practical purposes, we have two separate networks which can be trained separately if desired, for optimum performance, without any interference from each other, and with the convenience of having just one training algorithm block. Also, extending the principle

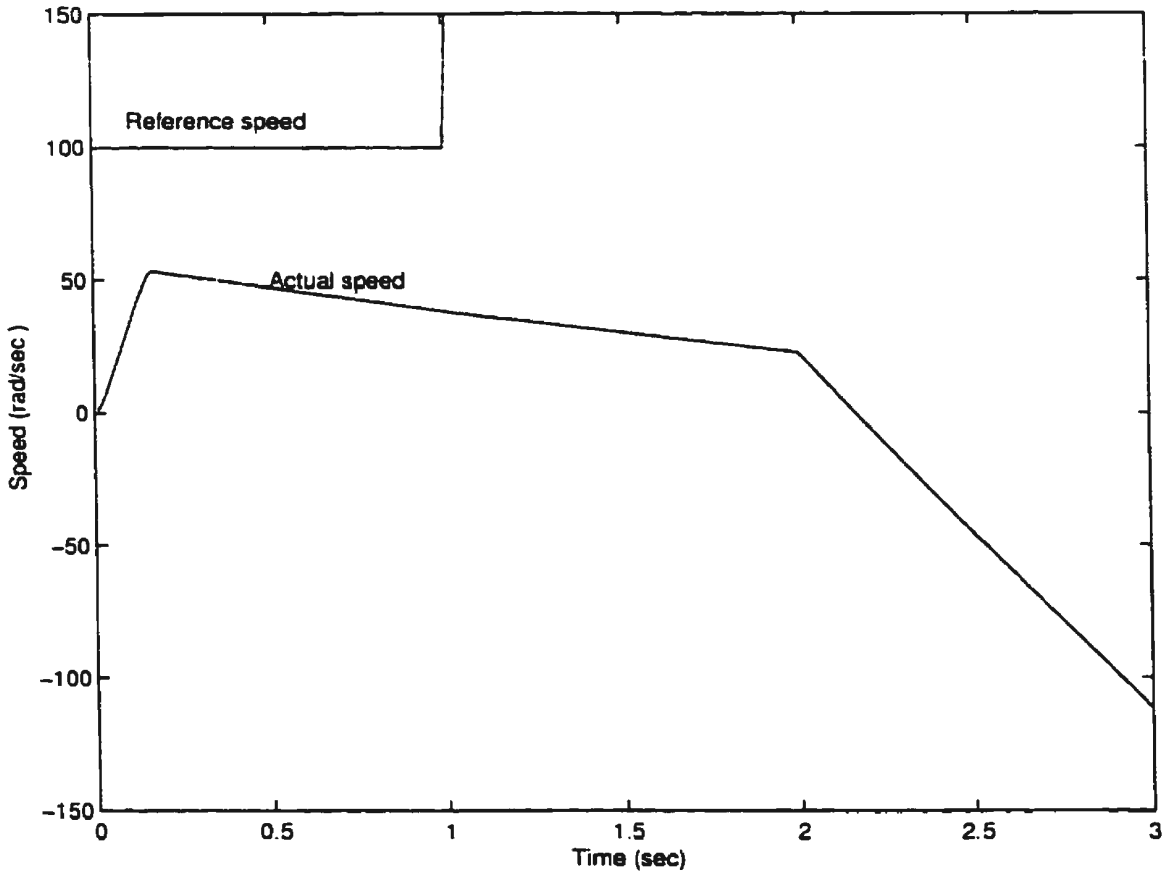


Figure 5.7: Performance of the fully-connected 13-75-2 ANN controller using a current amplifier: Step change in speed reference at t = 1.0 s and step change in load torque at t = 2.0 s

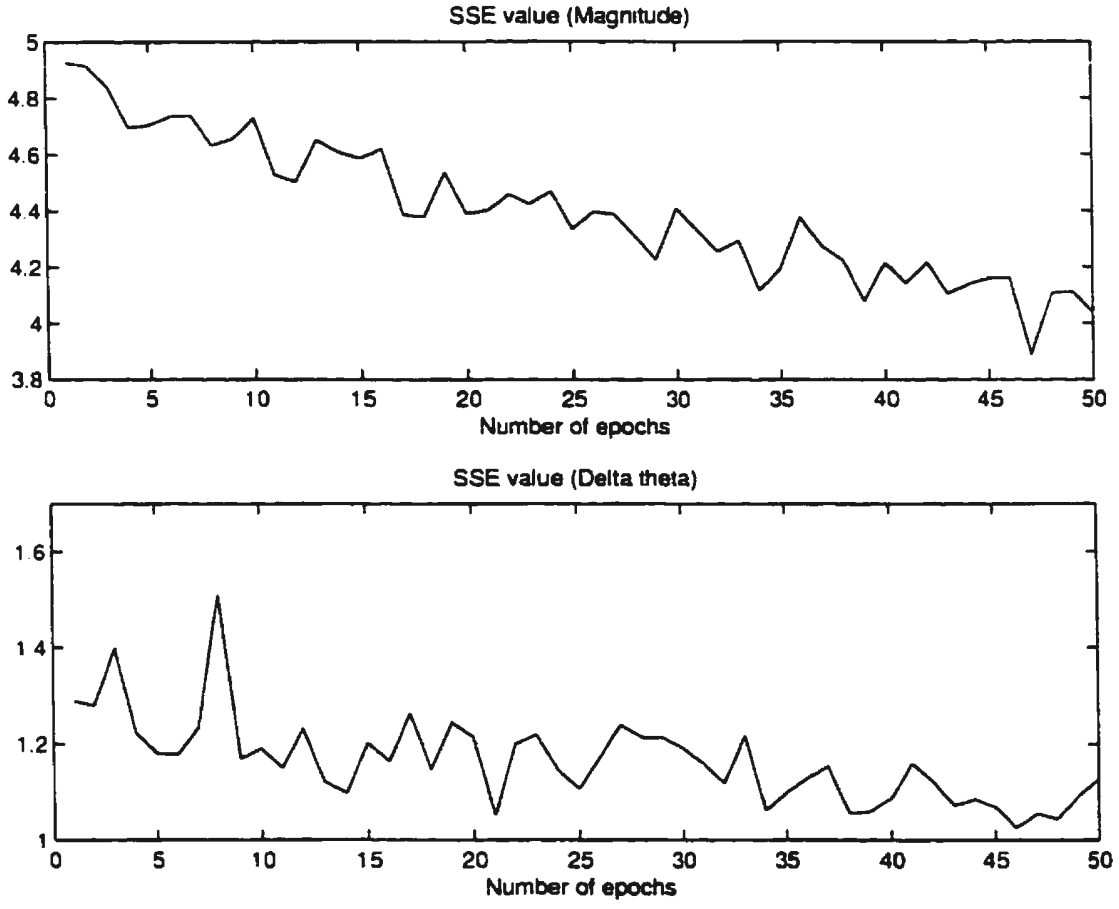


Figure 5.8: Sum squared error during training of 13-75-2 ANN controller using a current amplifier

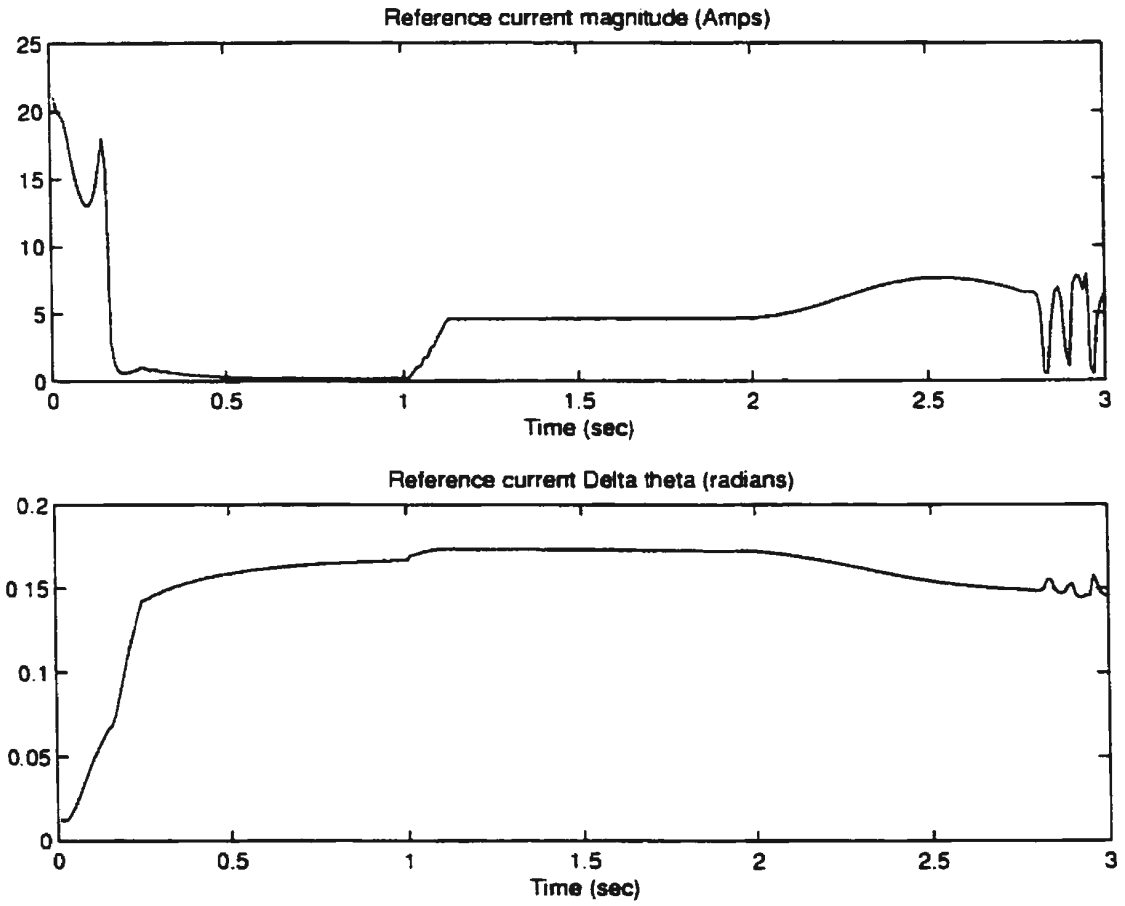


Figure 5.9: Outputs of the fully-connected 13-75-2 ANN controller using a current amplifier: Step change in speed reference at t = 1.0 s and step change in load torque at t = 2.0 s

further, more single output networks can be added in just one physical ANN.

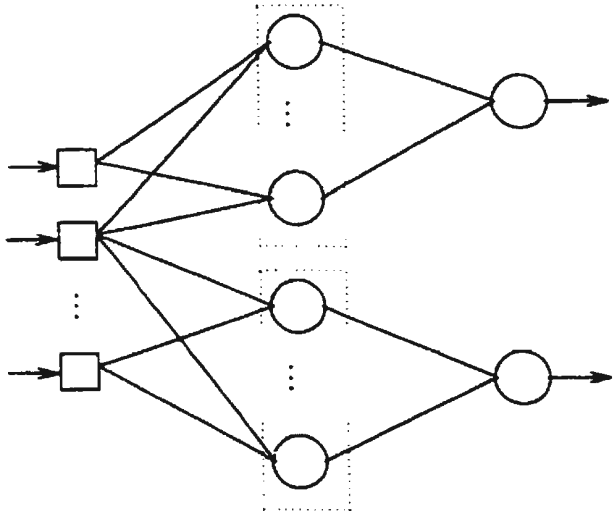


Figure 5.10: Dual output split-ANN for induction motor control

5.3.4 Voltage feedback scheme with current amplifier

Keeping in mind the various constraints discussed in the previous section, a dual output split-ANN has been trained to mimic the vector controller. The number of weights for a three-layer split-ANN is given by

$$N_{\text{weights}} = (I + 1)H \tag{5.44}$$

where I is the number of ANN inputs, and H is the number of neurons in the hidden layer. It should be noted that in a split-ANN, the number of weights does not depend on the number of ANN outputs.

The structure of the ANN chosen was 13-80-2, so that the number of weights is roughly equal to that of the network presented in subsection 5.3.2. The inputs to the

ANN are the same as those in subsection 5.3.2. The scheme is similar to the one shown in Figure 5.6, except for the fact that a split-ANN is used instead of a fully-connected network. The network was initially trained on a data set comprising of 1690 points, for 60 epochs with a learning rate of 0.01 for all neurons. After this, only the subnetwork producing the magnitude output was trained for another 100 epochs with the same learning rate and learning was blocked for the $\Delta\theta$ network. The plot of the "sum squared error" during training is shown in Figure 5.11. This quantity is the sum of the squares of the normalized output errors for a single network output, computed over the entire epoch.

Figure 5.12 shows the ANN performance for step changes in speed reference. The step changes at zero and 1 s have been included in the training data set, and as can be seen from the plot, the ANN responds well to these changes in speed reference. However, the next step change is one that hasn't been included in the training data set. The network performance is quite good in this case also, thereby proving that the network is able to generalize effectively. The actual network outputs for the above case are shown in Figure 5.13.

Next, the response of the ANN to a step change in load torque was tested by applying a step change in load torque from 9.3% to 93.0% rated torque at time = 2.0 secs. The response of the ANN controller is shown in Figure 5.14. As can be seen from the figure, the ANN response is very quick, leading to a speed recovery that is faster than regular vector control. The ANN outputs for this case are shown in Figure 5.15.

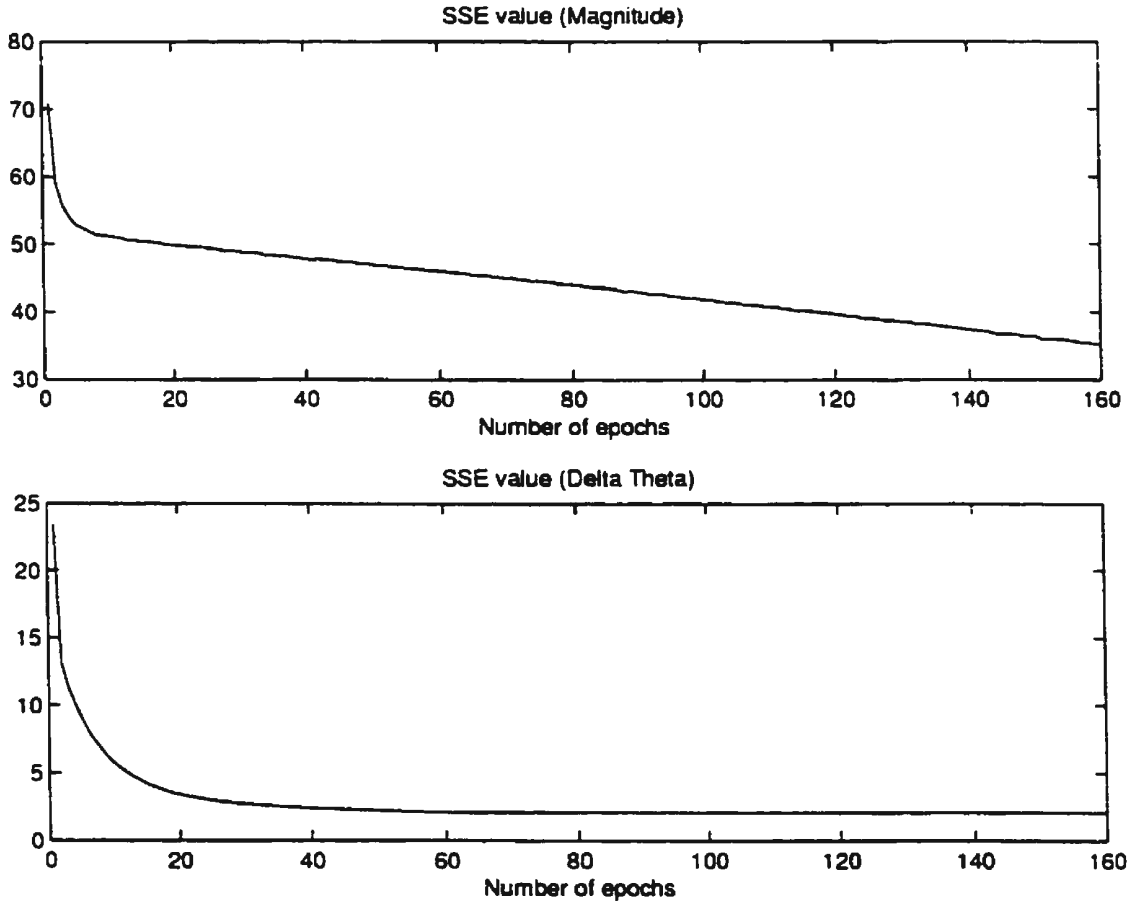


Figure 5.11: Sum squared error during training of the 13-80-2 split-ANN controller using a current amplifier

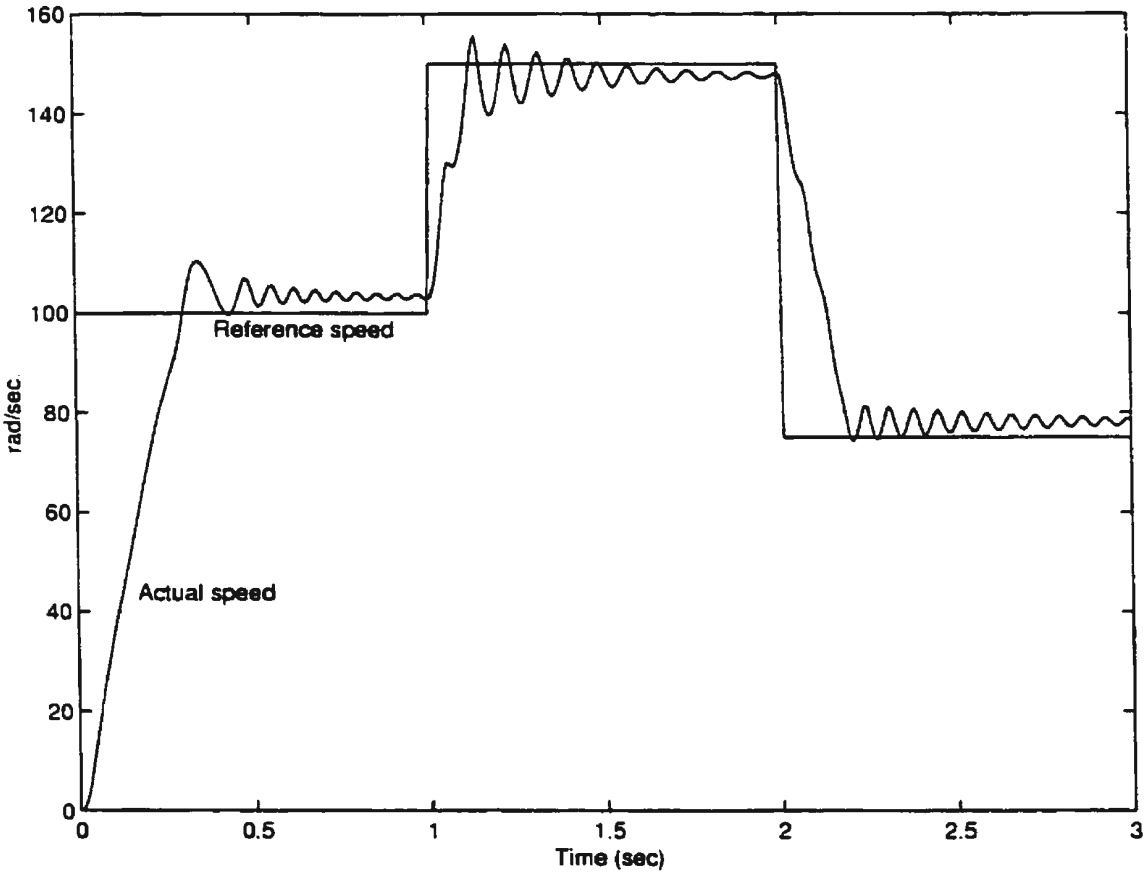


Figure 5.12: Performance of the 13-80-2 split-ANN controller using a current amplifier: Step changes in speed reference at t = 1.0 and 2.0 s

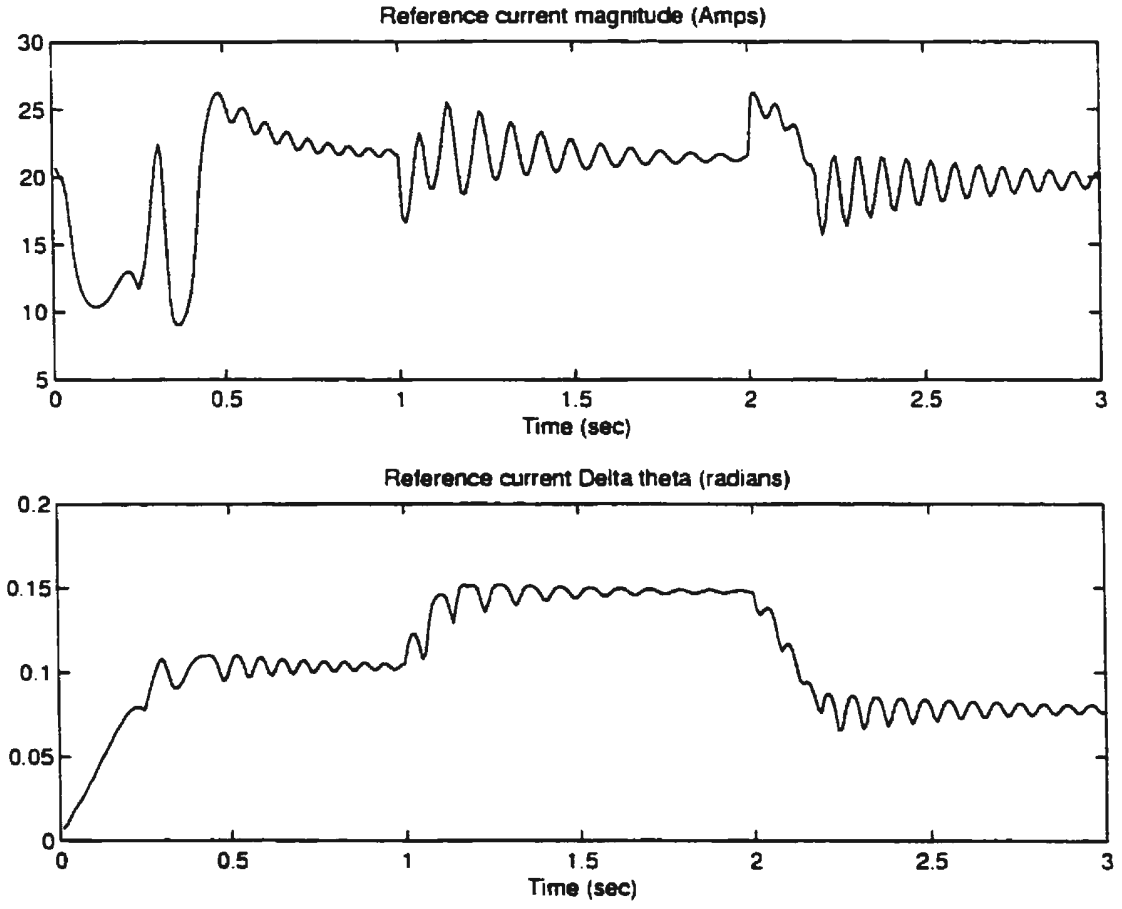


Figure 5.13: Outputs of the 13-80-2 split-ANN controller using a current amplifier: Step changes in speed reference at t = 1.0 and 2.0 s

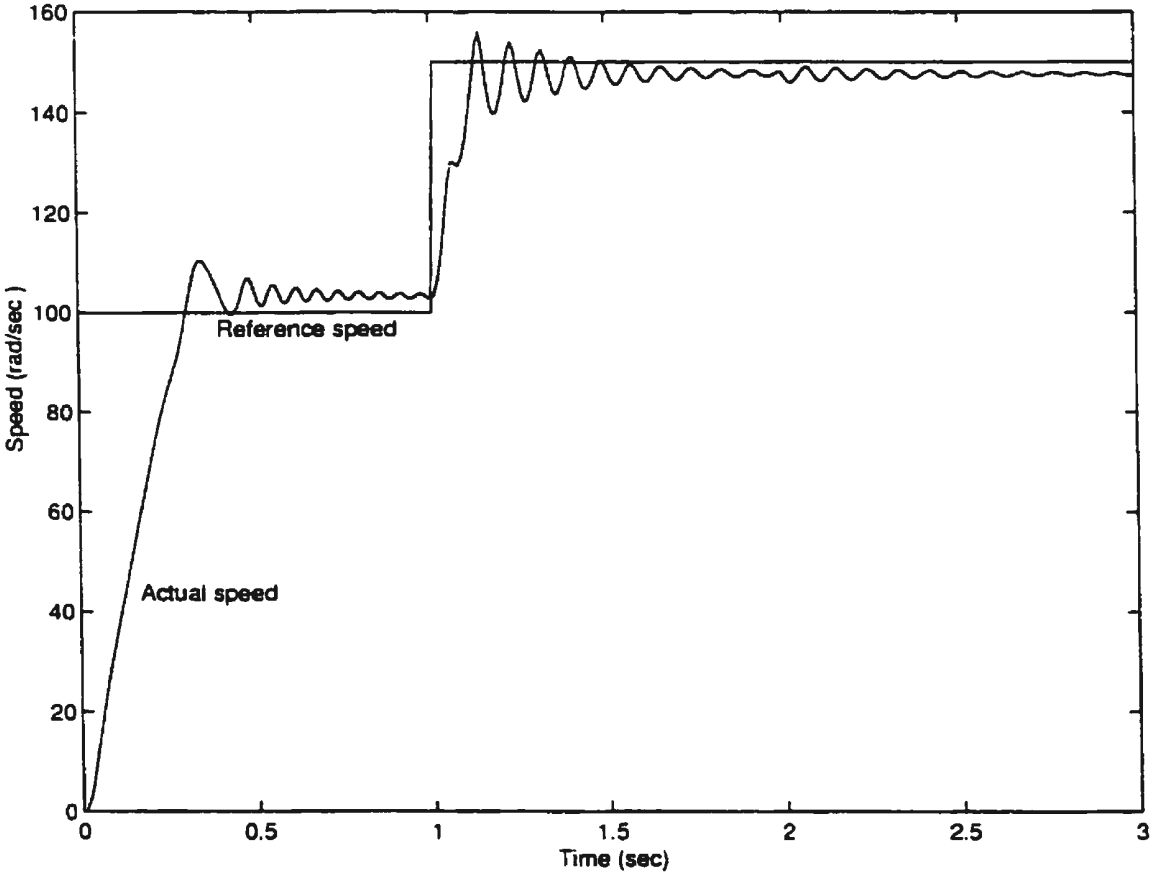


Figure 5.14: Performance of the 13-80-2 split-ANN controller using a current amplifier: Step change in speed reference at t = 1.0 s and step change in load at t = 2.0 s

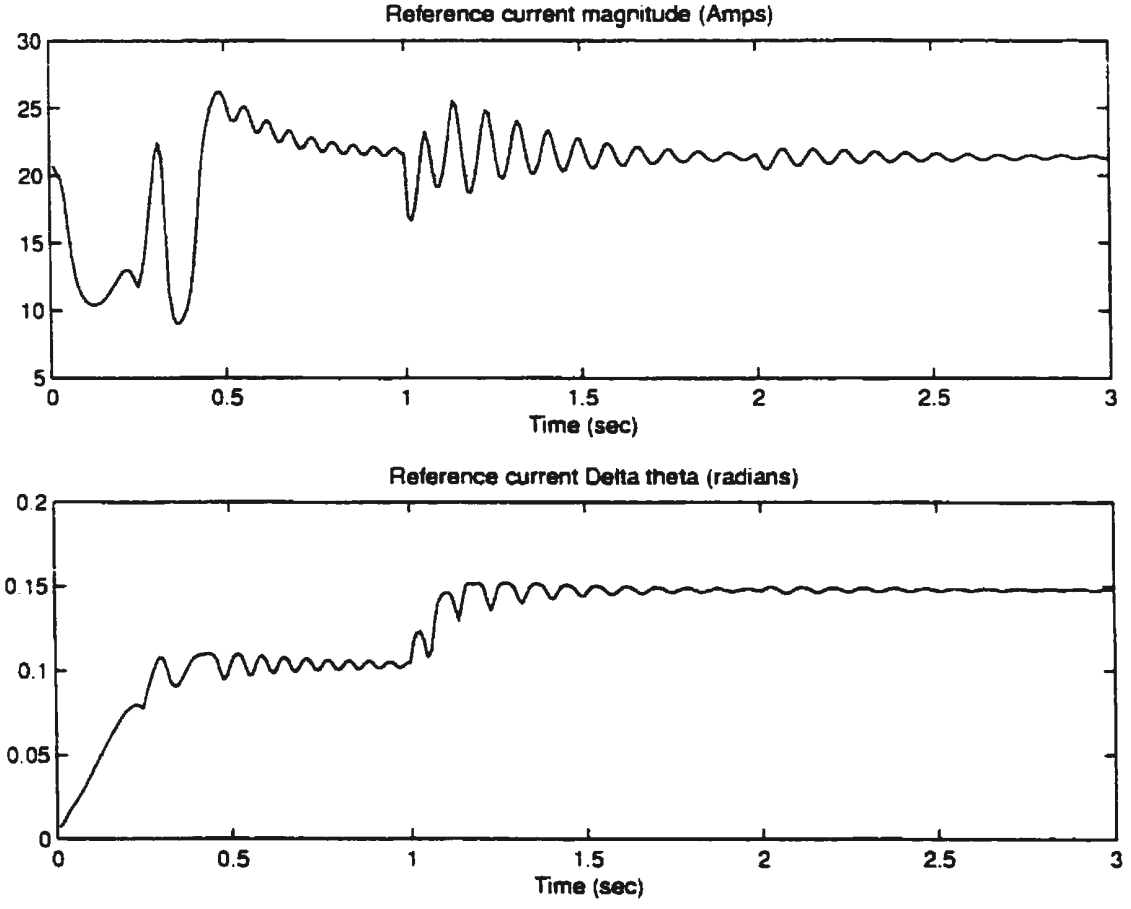


Figure 5.15: Outputs of the 13-80-2 split-ANN controller using a current amplifier: Step change in speed reference at t = 1.0 s and step change in load at t = 2.0 s

5.3.5 Voltage feedback scheme with PWM voltage source inverter

The scheme described in section 5.3.4 used a current amplifier to simplify the system for establishing the theoretical foundation of the scheme. The scheme in this subsection is similar to the previous scheme except for the fact that the current amplifier is replaced by a current-controller and an inverter combination [63]. A PWM current controller is chosen because it has a constant switching frequency. This feature is very important because the voltage feedback needs to be filtered since it is in the form of inverter output pulses and cannot be used directly for converting to the magnitude-$\Delta\theta$ format. A block diagram of the PWM controller is shown in Figure 5.16.

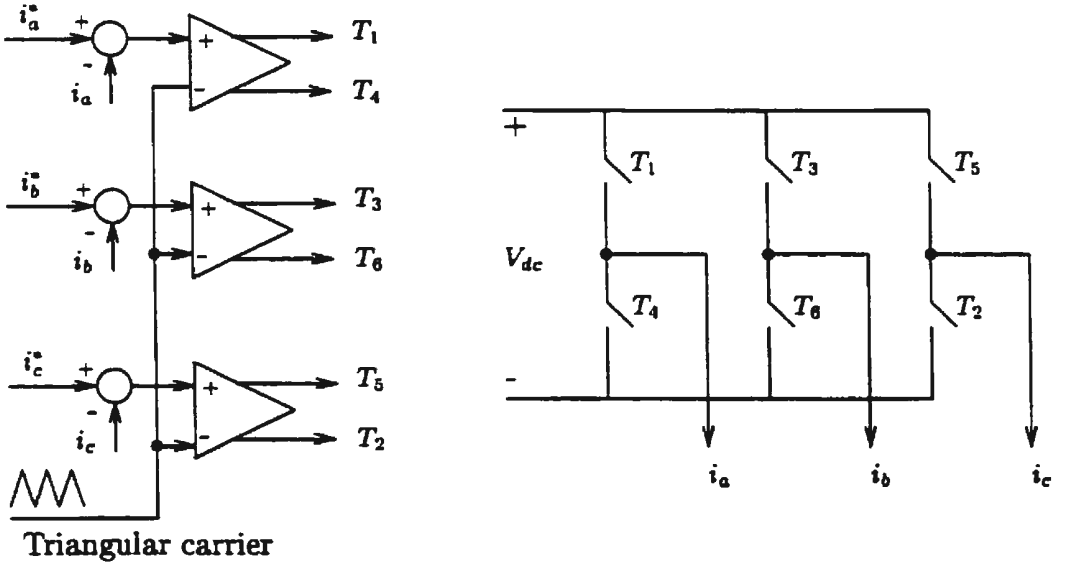


Figure 5.16: PWM current controller

The vector controller is run for step changes in load and speed reference, and all the ANN inputs and desired outputs are collected in a data file. The vector controller

performance is shown in Figures 5.17 to 5.22. The following speed and torque changes are used: A speed ramp from 0 to 100 rad/s in 0.5 seconds, a constant speed of 100 rad/s for 0.5 seconds, a speed ramp from 100 to 150 rad/s in 0.25 seconds, a constant speed of 150 rad/s for 0.75 seconds and a step change in load torque from 1N-m to 10N-m at t = 2 s. Figure 5.18 shows the reference and actual torque (as computed within the vector control algorithm), and the effect of the PWM voltage source inverter is clear in this figure. The ripples in the torque are due to the fact that the current is not a smooth sinusoid. Figure 5.19 shows the reference and actual value of i_{mr}, which is the flux component of the current. It should be noted that under ideal field-oriented conditions, the flux and torque should be completely decoupled, but this is not the case as can be seen in this figure. A step change in speed reference and a step change in load torque cause the flux component of the current to deviate from its reference value, indicating a coupling between the flux and the torque. The main reason for this is that the sampling time for vector control is 500μs. Reducing the sampling time would lead to more effective decoupling.

The inverter voltage for phase a, before and after filtration, is shown in Figure 5.20. Filtration plays a very important role in inverter based ANN training, because the inverter voltage has discrete values, and it is difficult to see any trend from a few previous values. A first order digital filter is used for filtering out the ripple, in this case. The reference and actual current for phase a are both shown in Figure 5.21. The distortion in the actual current waveshape is probably due to the fact that the PWM current controller functions in the over modulation range, for a part of the cycle.

Figure 5.22 shows the filtered and unfiltered versions of the reference currents converted to magnitude-$\Delta\theta$ using the "DQ-MF" block. As can be seen in this case, $\Delta\theta$ has a lot more ripple than the magnitude, necessitating another filter at this stage. Figure 5.23 shows the same quantities for the voltage feedback. Here again, it can be

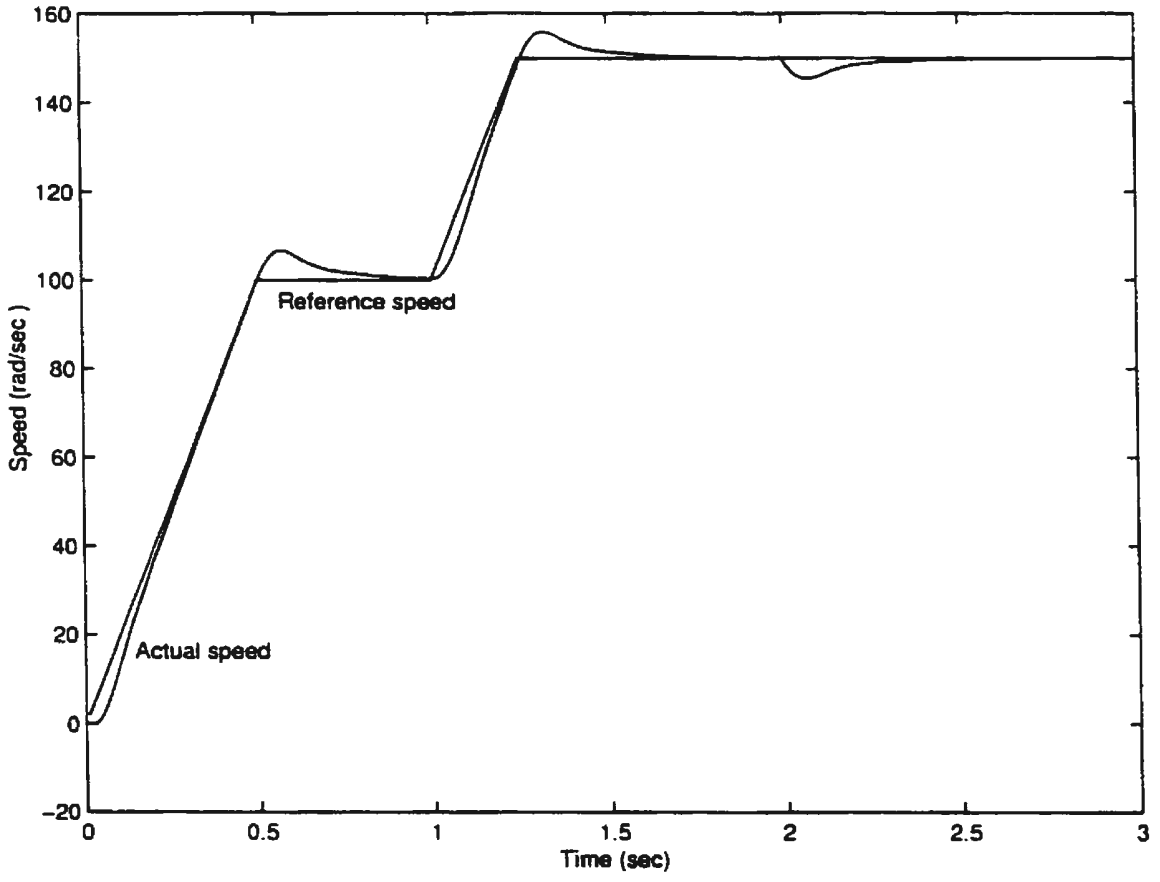


Figure 5.17: Performance of induction motor drive with field-oriented control (speed response)

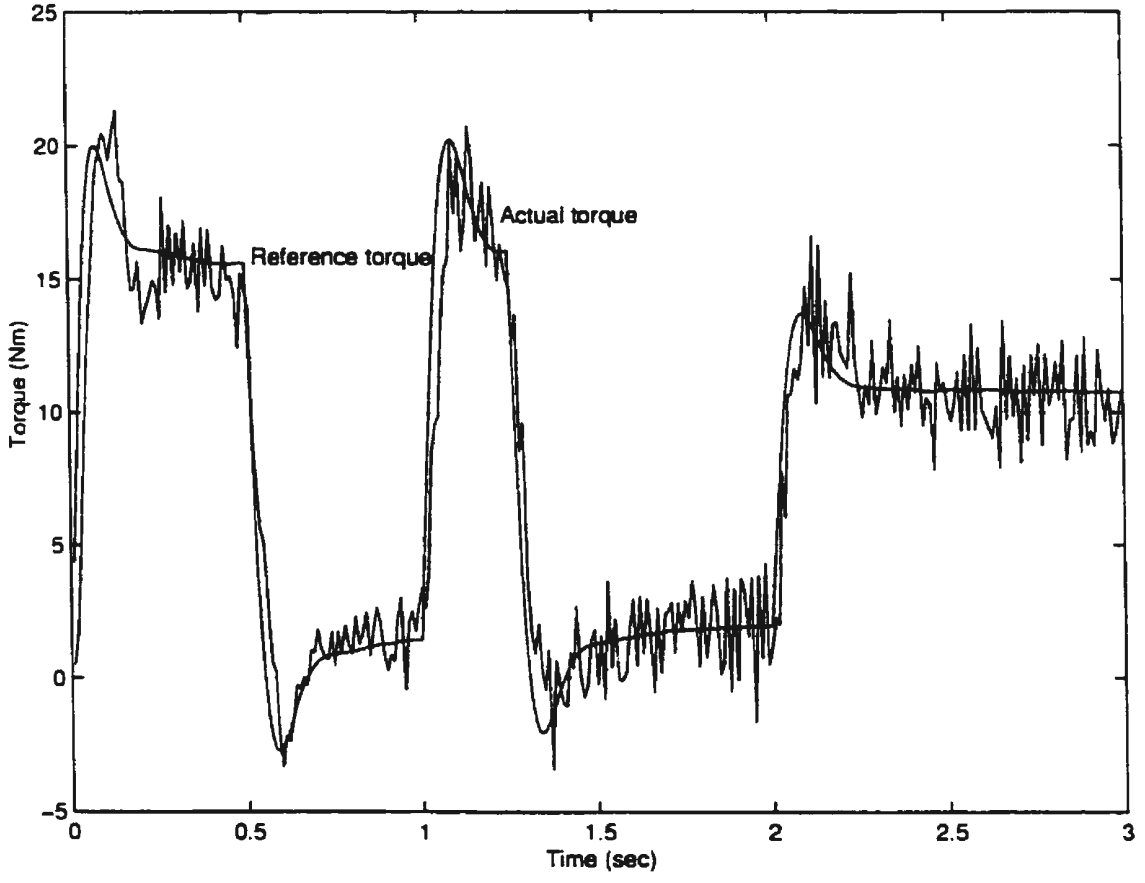


Figure 5.18: Performance of induction motor drive with field-oriented control (torque response)

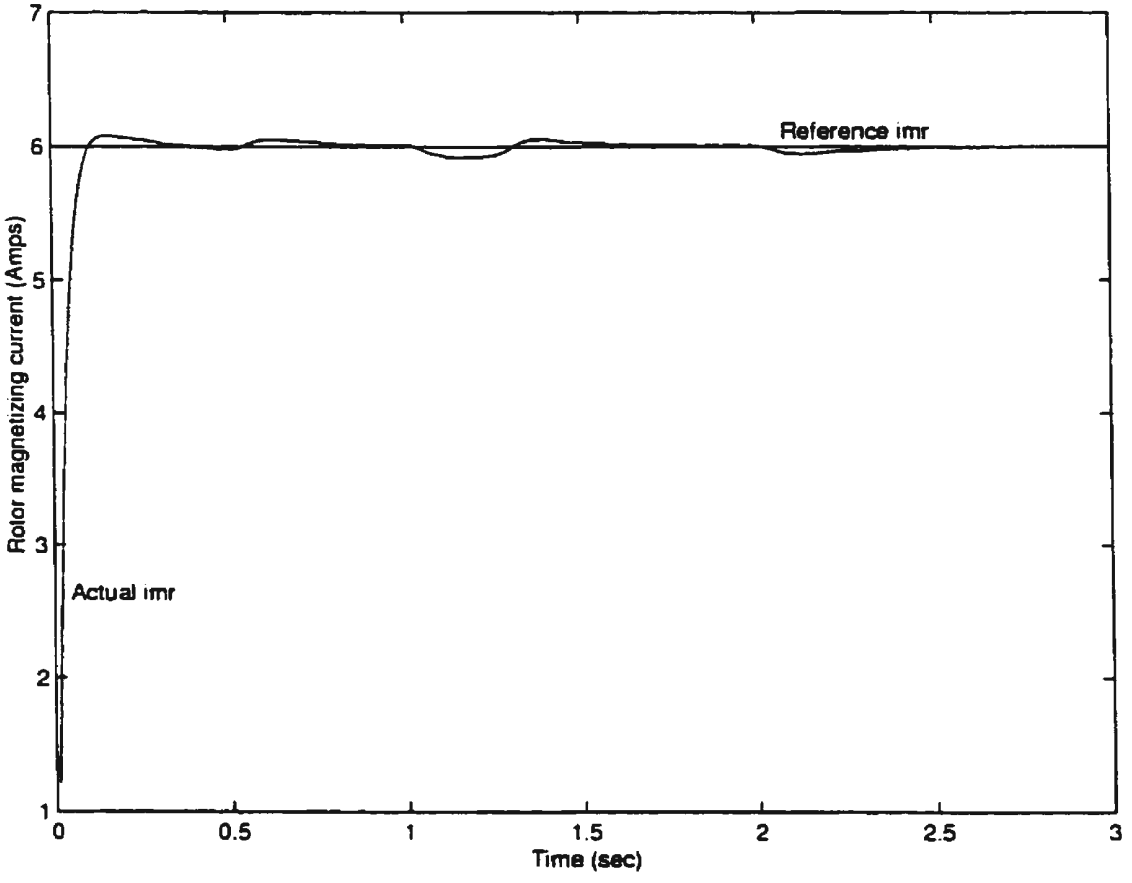


Figure 5.19: Performance of induction motor drive with field-oriented control (i_{mr} response)

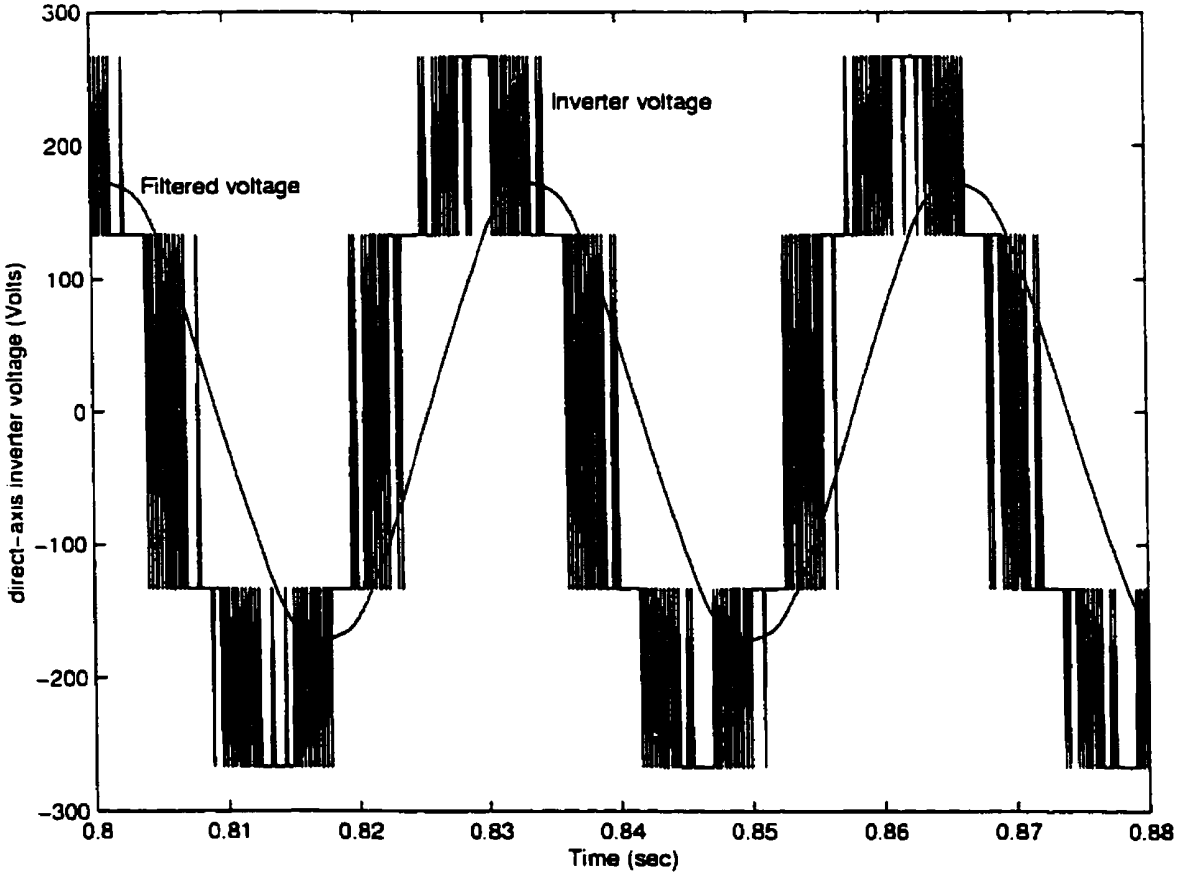


Figure 5.20: d-axis inverter and filtered voltages

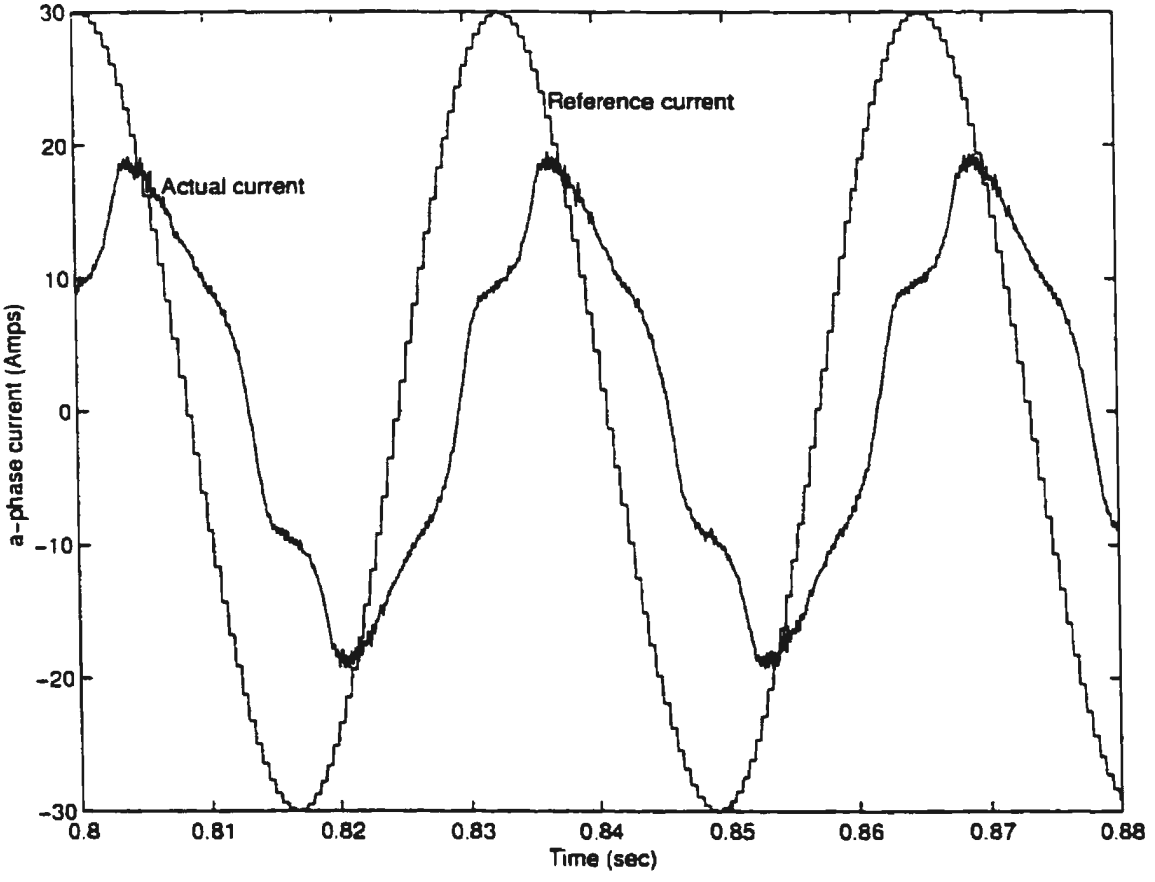


Figure 5.21: a-phase reference and actual currents with rotor field-oriented control

seen that another stage of filtering is required to make the voltage feedback suitable as ANN input. A block diagram of this scheme is shown in Figure 5.24.

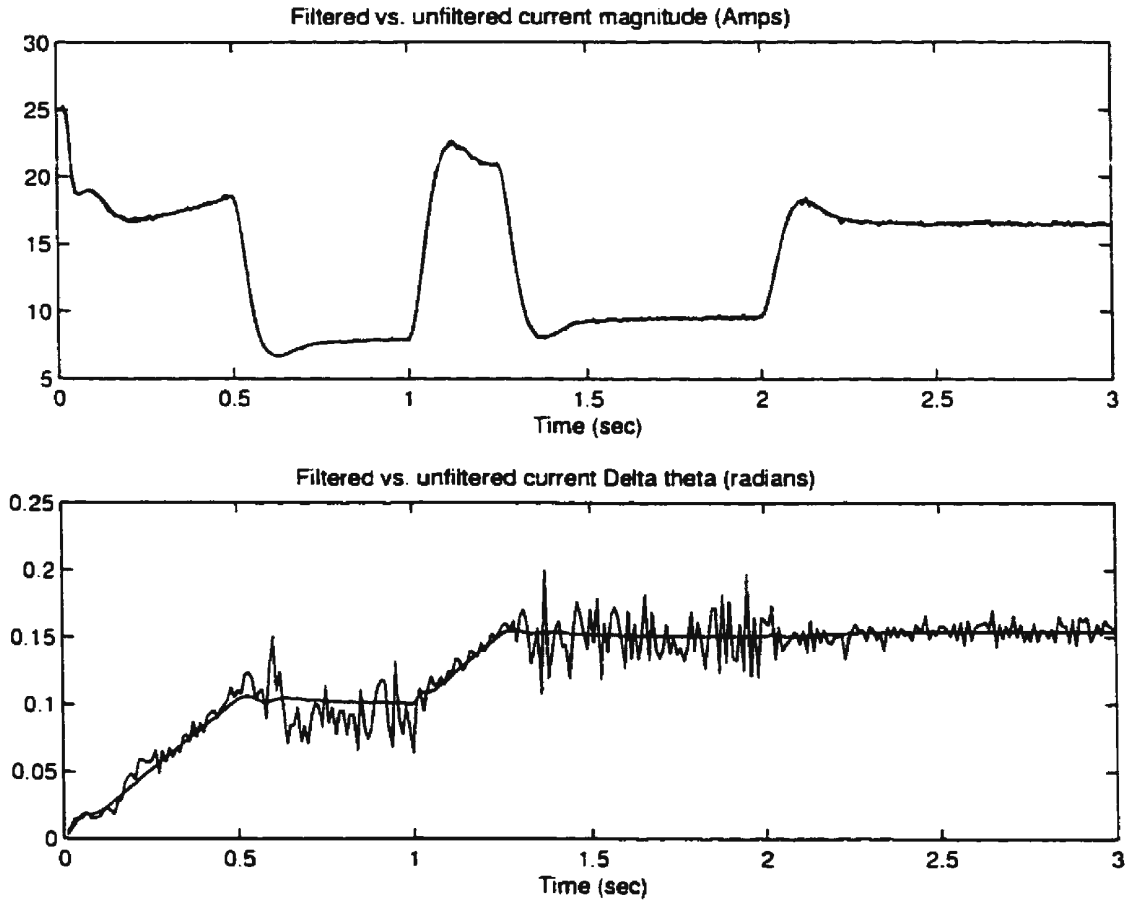


Figure 5.22: Generation of desired outputs for ANN training

The PWM voltage source inverter is run at a switching frequency of 7 kHz, and the amplitude of the triangular carrier signal is 10V. The transistors used in the simulation have a current rating of 150A, and a voltage rating of 600V. If the current through, or the reverse voltage across, any transistor exceeds these ratings, then the inverter block invokes the error module and terminates the simulation. It should be noted that the

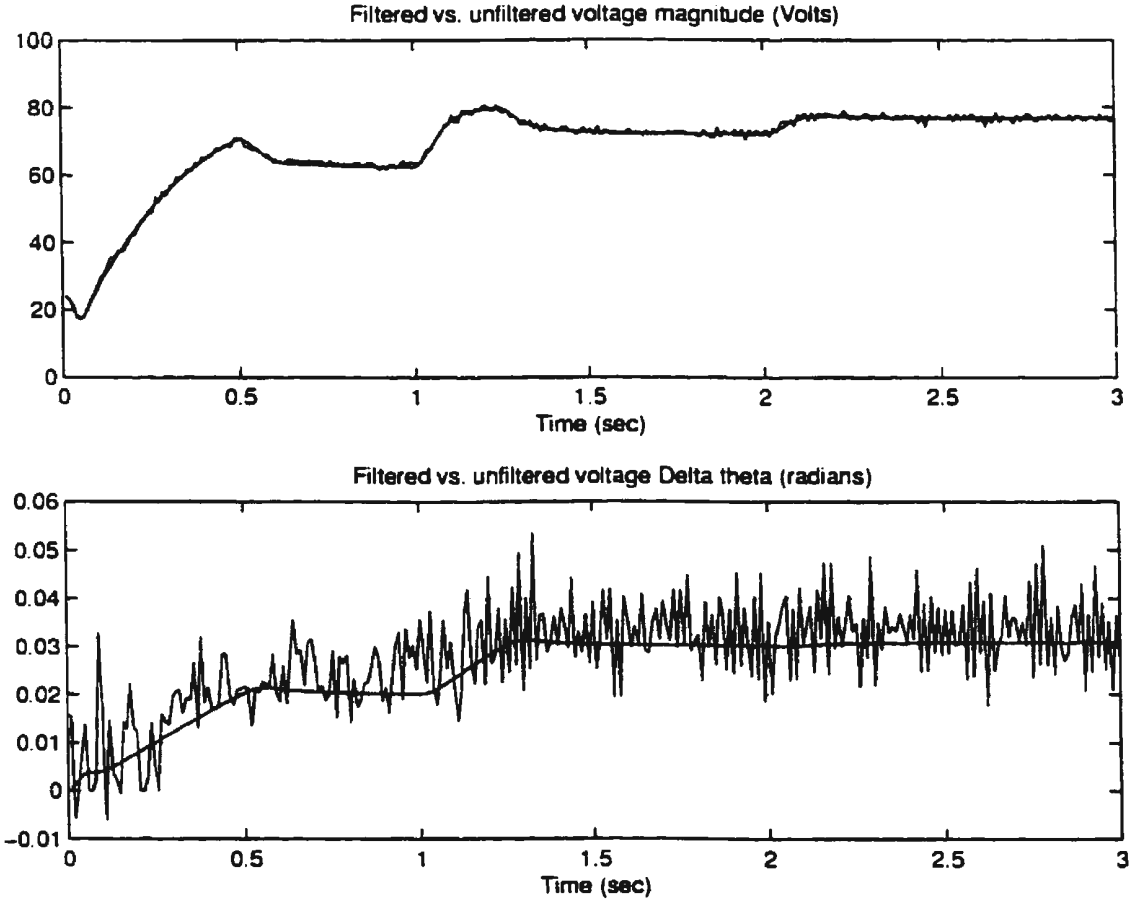


Figure 5.23: Voltage feedback for ANN training

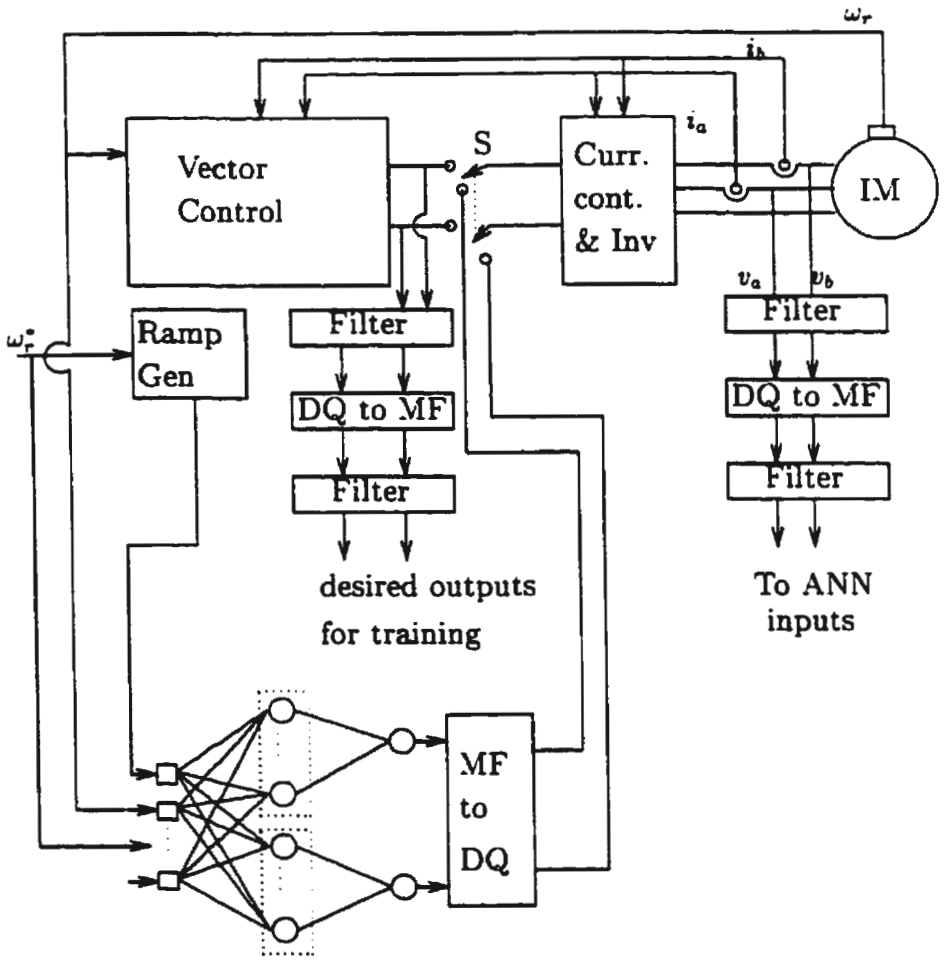


Figure 5.24: Current controller and inverter based scheme for ANN control of induction motor using voltage feedback

dead-time needed between the two transistors of the same leg is not considered because this would drastically slow down the simulation. Also, the transistor is considered an ideal switch and small non-idealities like the forward voltage drop are not modeled in the simulation.

The network was initially trained on a data set comprising of 1630 points, for 20 epochs with a learning rate of 0.01 for both output neurons, 0.50 for the magnitude subnetwork and 0.05 for the $\Delta\theta$ subnetwork. The plot of the "sum squared error" during training is shown in Figure 5.25.

After training, the network performance was tested by letting the ANN run the induction motor for various changes in speed reference and load torque. The ANN output is passed through the "MF-DQ" block to convert it to d and q-axis currents, which are then subsequently converted to three-phase quantities. There is no filtration required at this stage. Figure 5.26 shows the ANN performance for step changes in speed reference. The step changes at zero and 1 s have been included in the training data set, and, as can be seen from the plot, the ANN responds well to these changes in speed reference. However, the next step change is one that has not been included in the training data set. The network performance is quite good in this case also, thereby proving that the network is able to generalize effectively. However, the network has a peak steady state error of about 5.5%. This may not be precise enough for a high performance drive, but it certainly fares well in comparison to traditional open-loop V/f drives. It should be noted that this network is by no means the most optimum one, and better training and a more extensive training data set might lead to much better performance. The actual network outputs for the above case are shown in Figure 5.27.

Next, the response of the ANN to a step change in load torque was tested by applying a step change in load torque from 9.3% to 93.0% rated torque (1Nm to 10Nm)

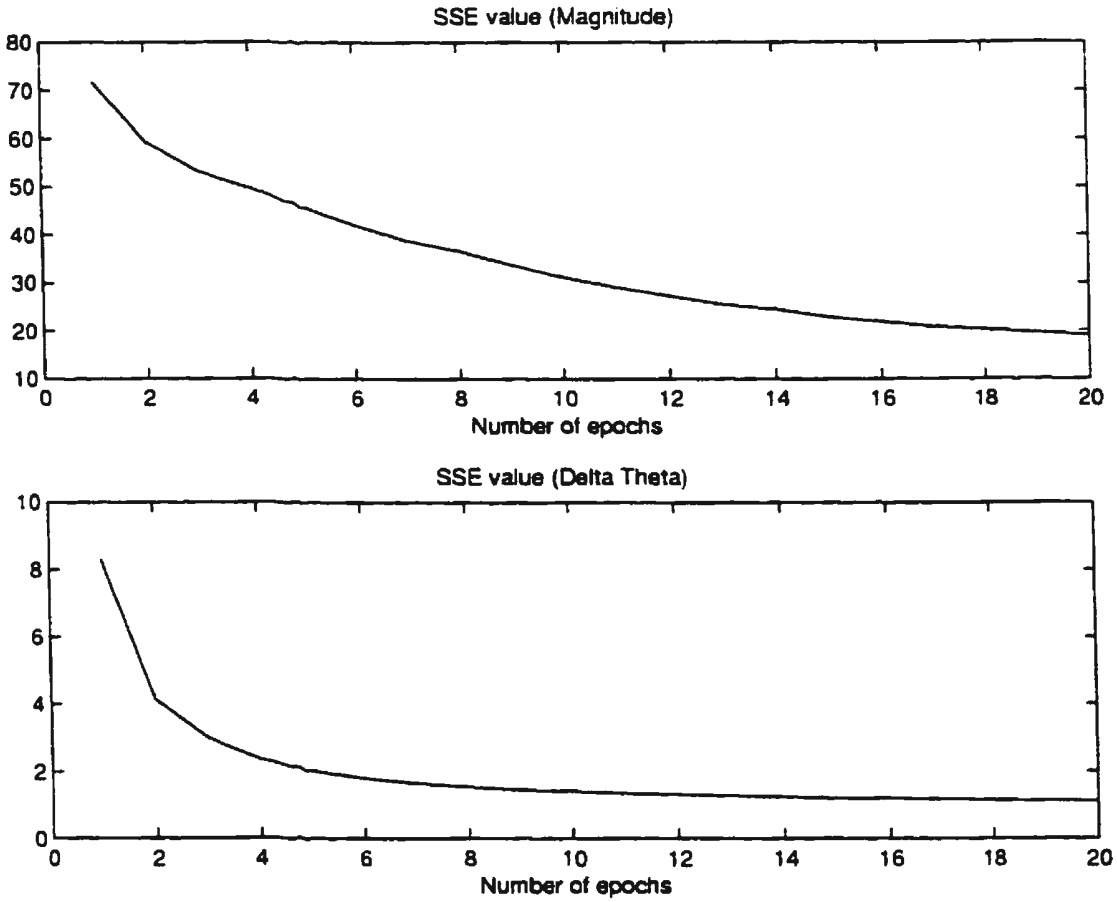


Figure 5.25: Sum squared error during training of the 13-80-2 split-ANN controller using an inverter

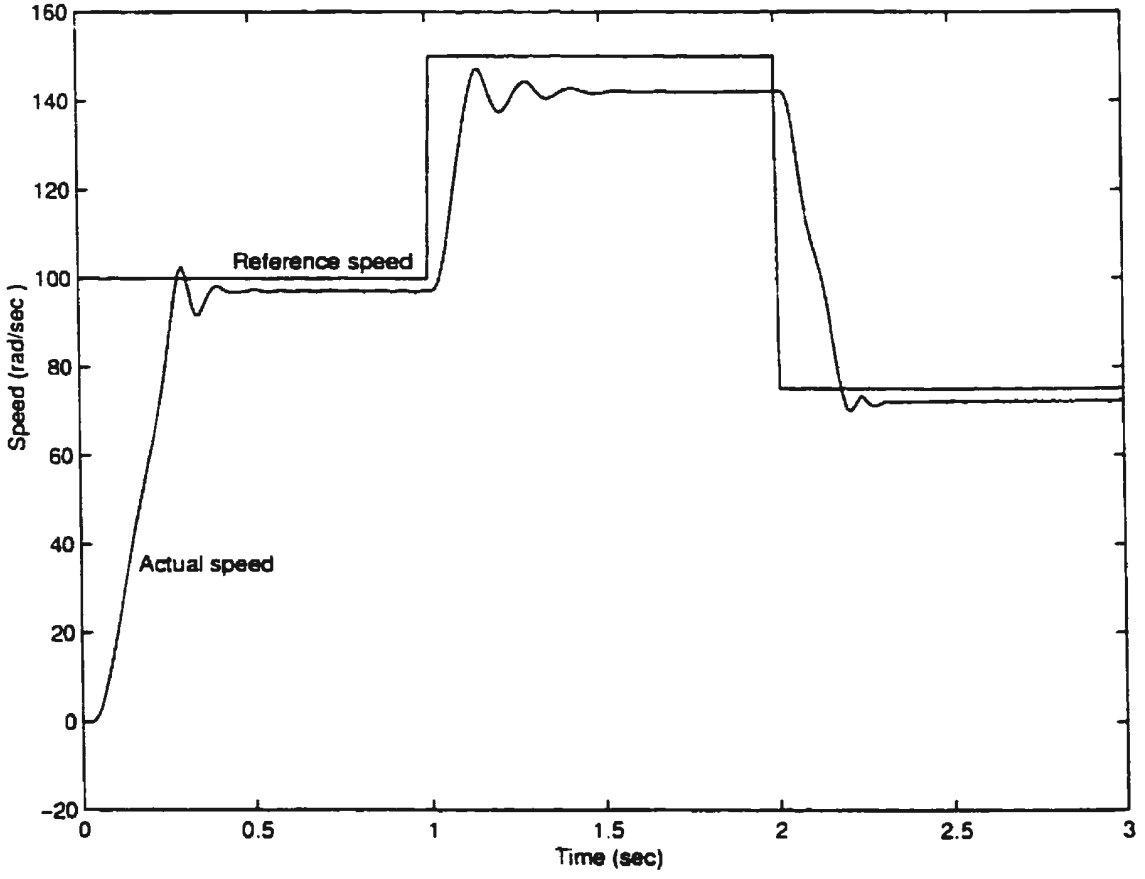


Figure 5.26: Performance of the 13-80-2 split-ANN controller using an inverter: Step changes in speed reference at t = 1.0 and 2.0 s

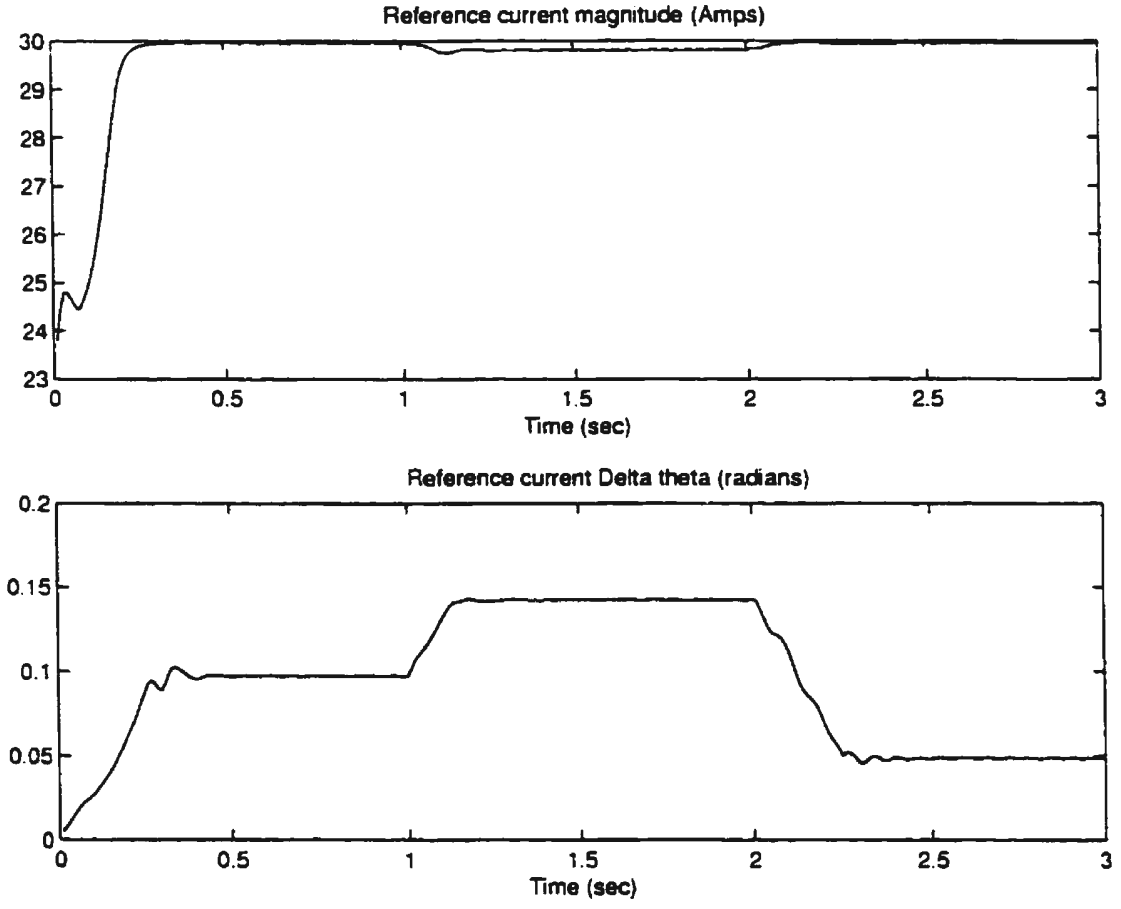


Figure 5.27: Outputs of the 13-80-2 split-ANN controller using an inverter: Step changes in speed reference at t = 1.0 and 2.0 s

at time 2.0 secs. The response of the ANN controller is shown in Figure 5.28. As can be seen from the figure, the ANN response is very quick, leading to a speed recovery that is faster than regular vector control. The fast recovery is due to the fact that the ANN is running the motor in "overdrive". The ANN outputs for this case are shown

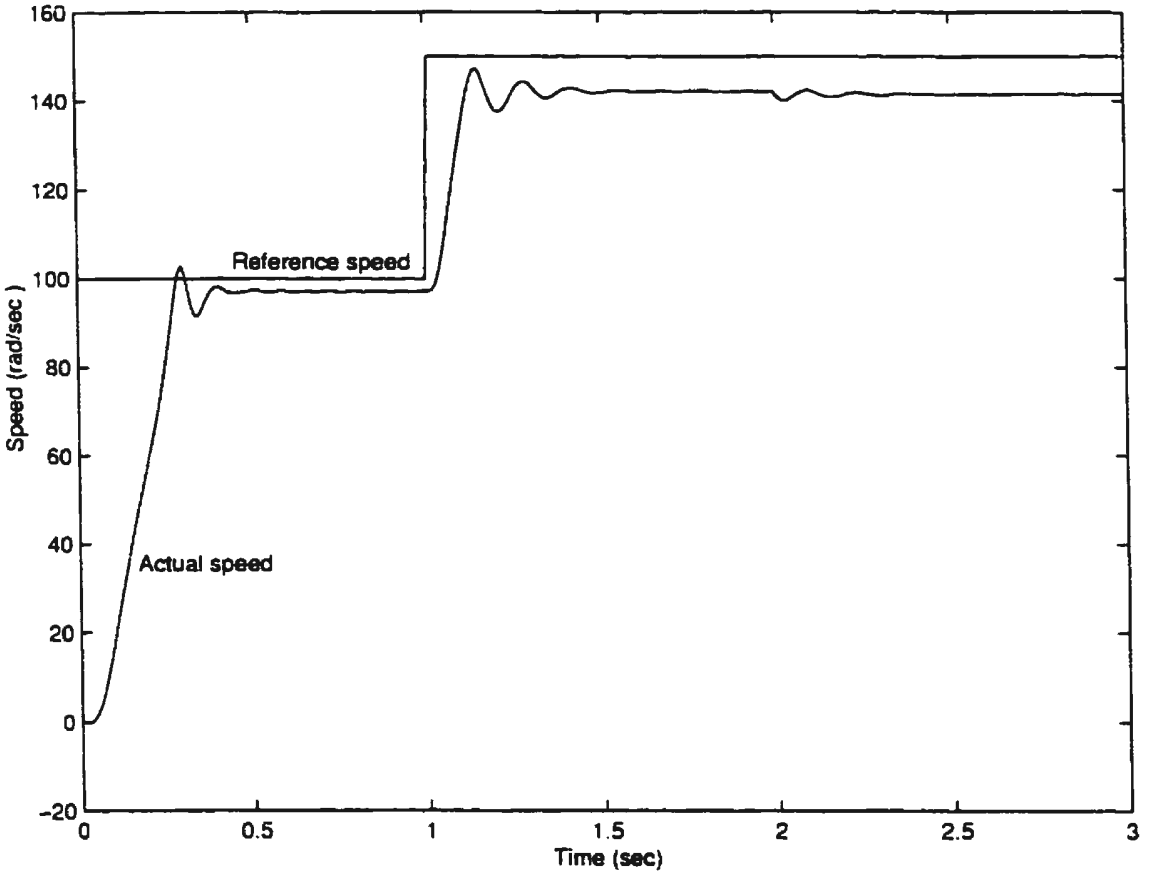


Figure 5.28: Performance of the 13-80-2 split-ANN controller using an inverter: Step change in speed reference at t = 1.0 s and step change in load torque at t = 2.0 s

in Figure 5.29.

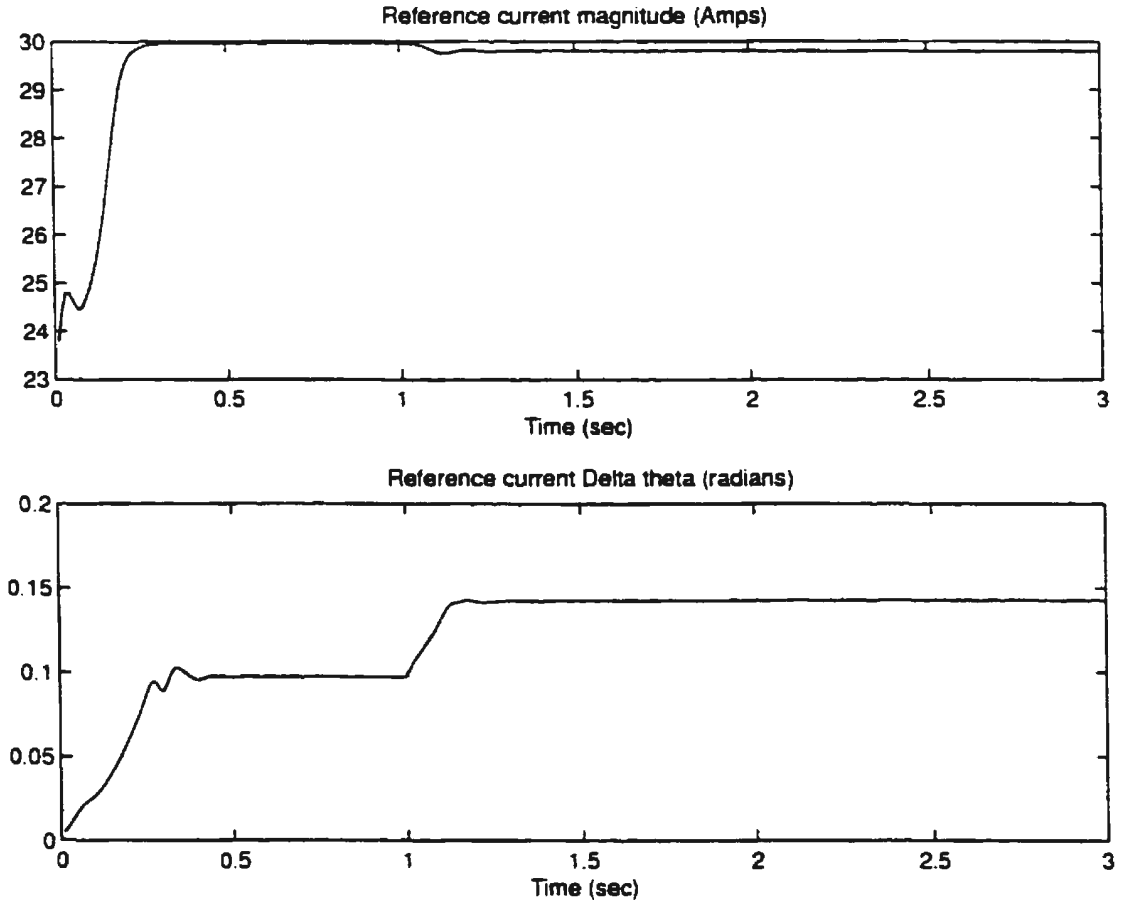


Figure 5.29: Outputs of the 13-80-2 split-ANN controller using an inverter: Step change in speed reference at t = 1.0 s and step change in load at t = 2.0 s

5.4 On-line training

The basic idea behind on-line control is to use the speed feedback and reference speed to get the speed error, and use this speed error for adjusting the weights of the ANN on-line using backpropagation. This requires estimating the plant Jacobian using a method similar to the one outlined in section 5.2. Since the ANN controller produces the current reference magnitude and $\Delta\theta$, partial derivatives of the motor speed ω_r with respect to these quantities would be the two elements of the Jacobian vector. The process of computing these quantities is quite involved, and as seen in section 5.2, would result in a large increase in computation time for real-time implementation. However, even if the sign of the Jacobian is known, it is enough for running backpropagation, because the direction of the gradient is then known. For this work, both elements in the Jacobian vector were assumed to be +1, resulting in a lot of simplification.

5.4.1 Voltage feedback scheme with PWM voltage source inverter

The response of the ANN controlled induction motor drive with a current controlled PWM VSI is shown in Figure 5.30. In this figure, the drive is run with off-line control till the motor reaches steady state. It has been seen that there is a steady state error with off-line control, and it is difficult to nullify this error. On-line training is switched on at t = 1.0 s, and it can be seen that the drive reaches its desired steady state at about t = 1.7 s. On-line training is switched off at t = 2.0 s, after the speed has reached its steady state value. The learning rate is kept very small (0.0001 for all the neurons) to prevent oscillations about the reference speed. The actual ANN outputs for the above case can be seen in Figure 5.31. Here, it can be seen that the magnitude

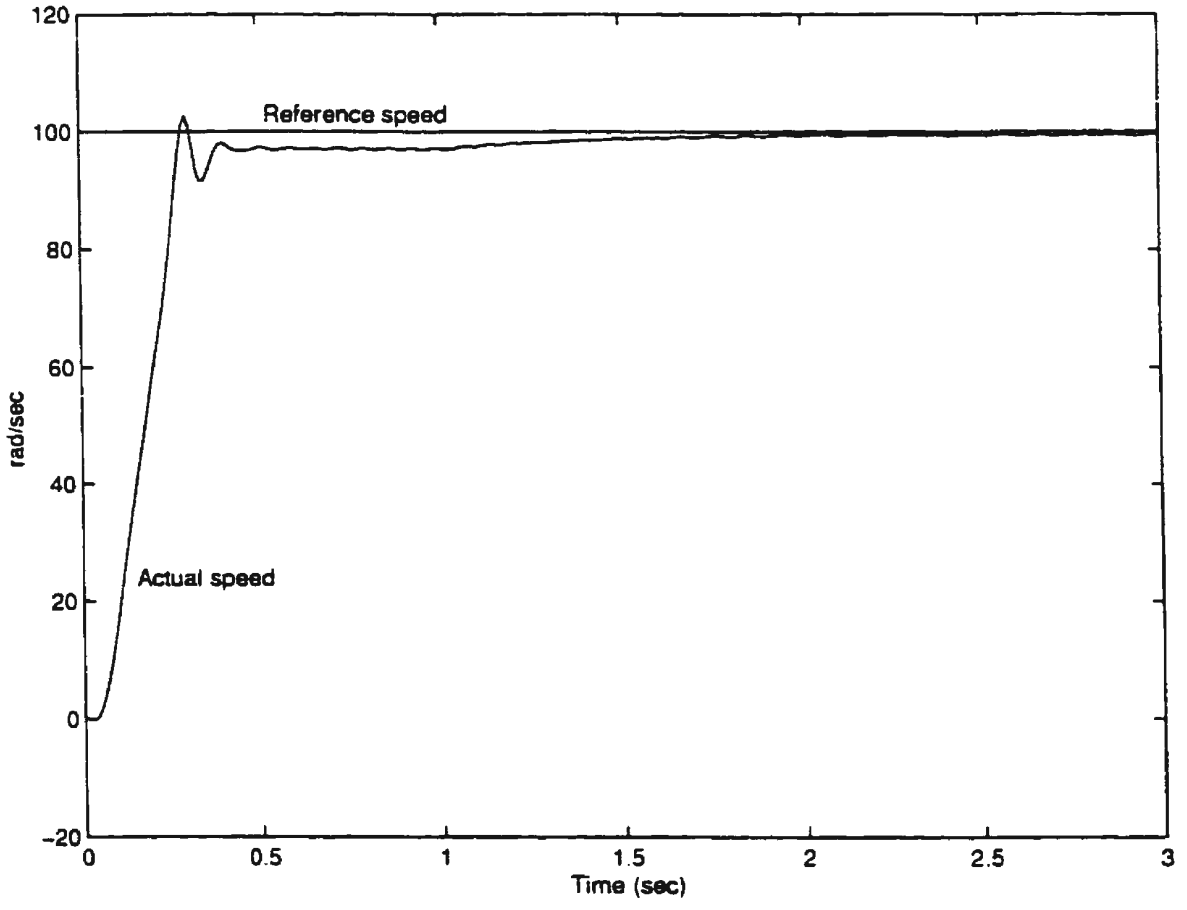


Figure 5.30: Performance of the 13-80-2 split-ANN controller using an inverter: On-line training implemented from t = 1.0 s to t = 2.0 s

output saturates at its peak value, and the ANN essentially runs on frequency control during on-line training.

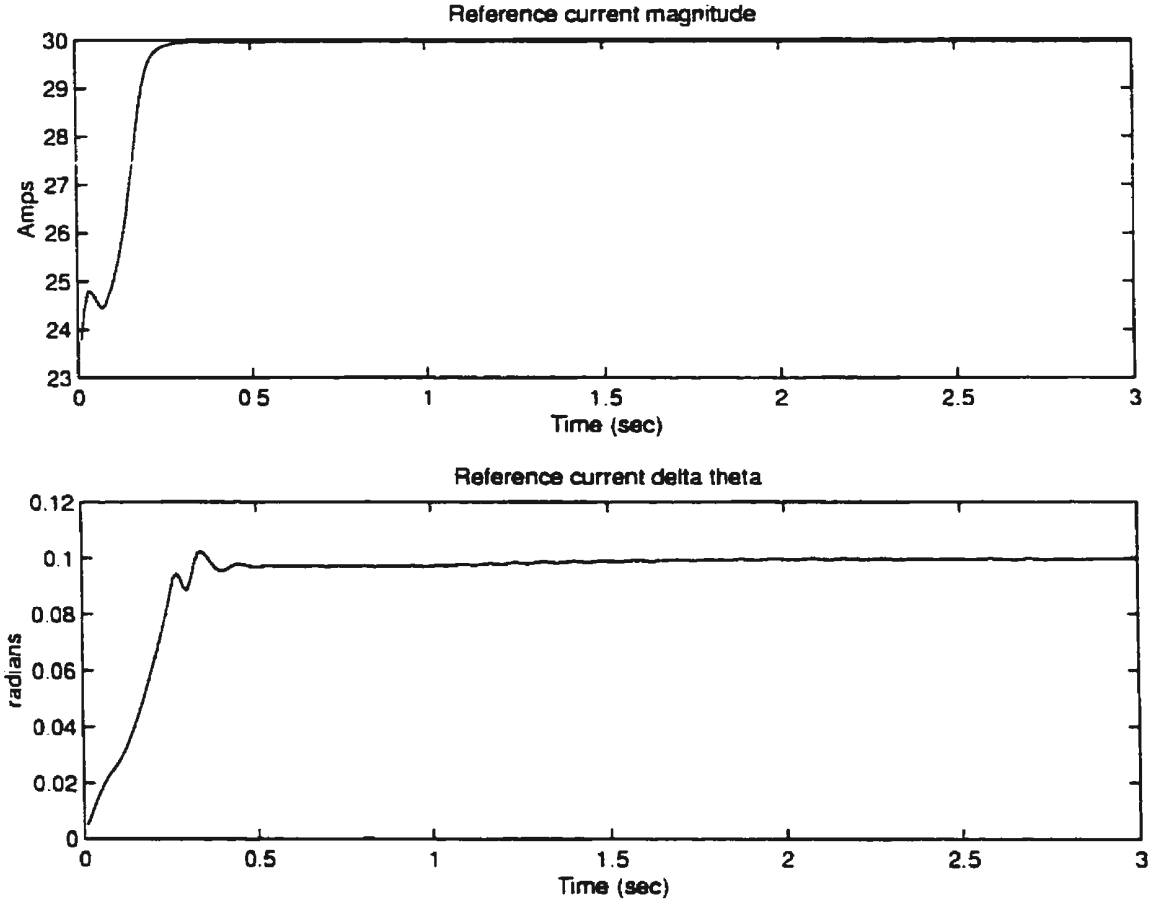


Figure 5.31: Outputs of the 13-80-2 split-ANN controller using an inverter: On-line training implemented from t = 1.0 s to t = 2.0 s

5.4.2 Effect of parameter variation

This subsection studies the effect of induction motor parameter variation when the ANN controller is functioning under on-line control. For this simulation experiment,

on-line training was initiated at t = 1.0 s like before, and was terminated at t = 2.5 s. However, this time both the stator and rotor resistances are increased by 100% linearly from t = 2.0 s to t = 2.5 s. The performance of the ANN is shown in Figure 5.32, and the actual ANN outputs are shown in Figure 5.33. As can be seen from these figures,

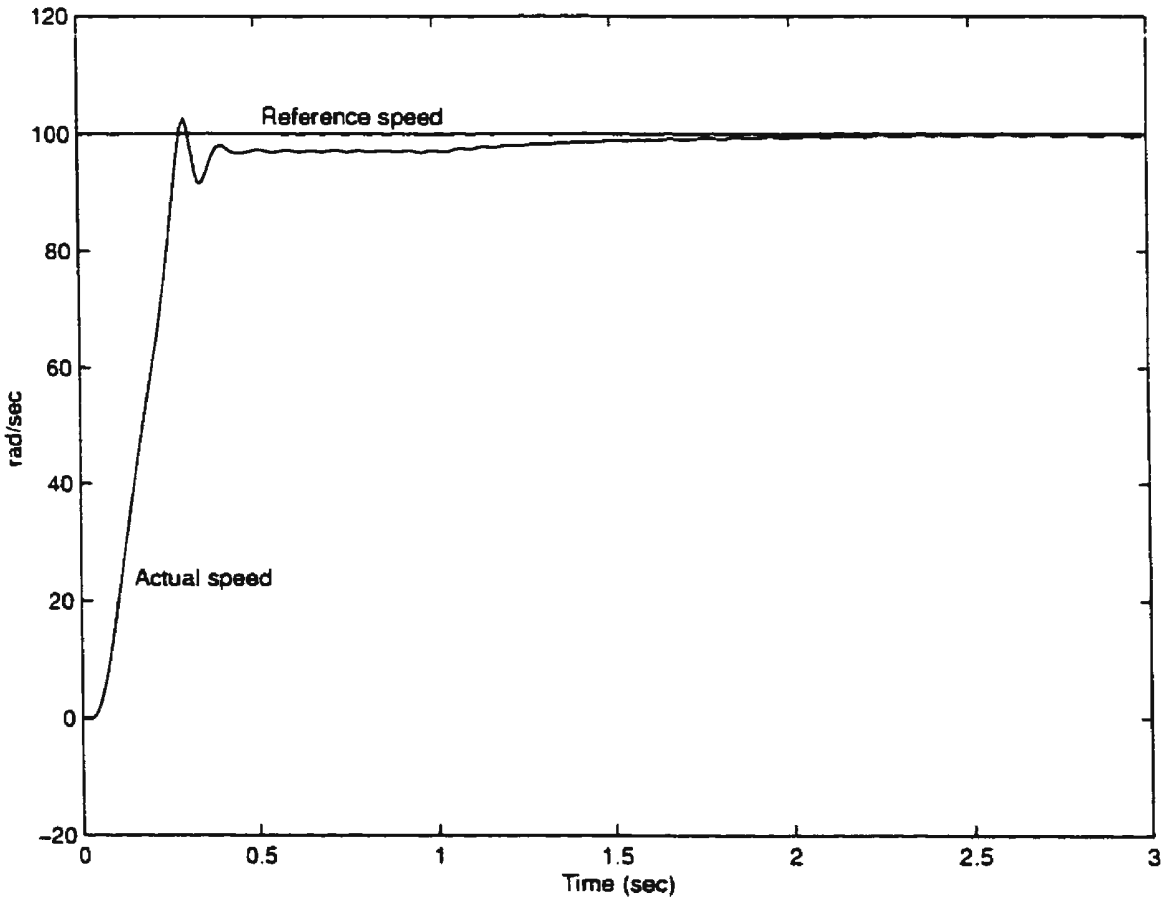


Figure 5.32: Performance of the 13-80-2 split-ANN controller using an inverter: On-line training implemented from t = 1.0 s to t = 2.5 s and linear change in motor parameters from t = 2.0 s to t = 2.5 s

the ANN performance is quite good even under such a drastic change in parameters. It should be noted that in real applications, one would usually not encounter such a large

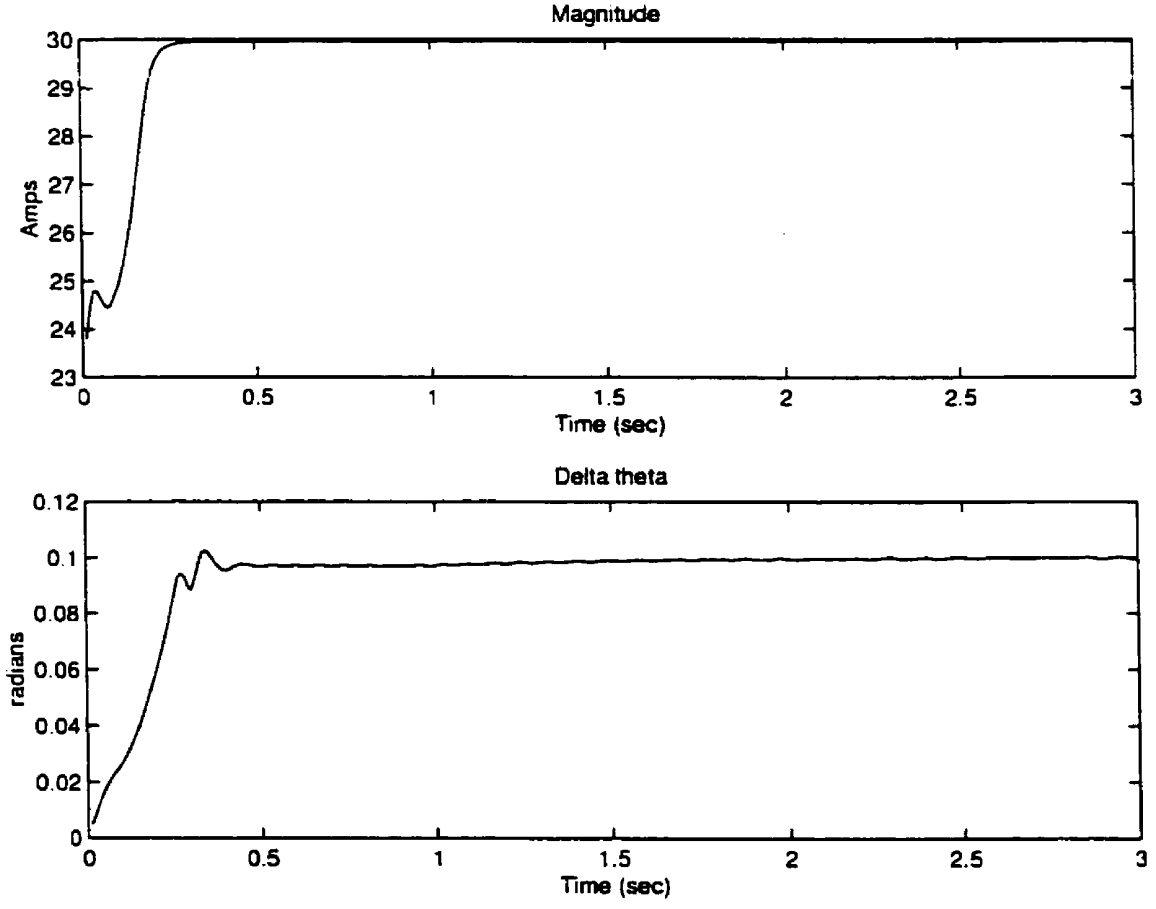


Figure 5.33: Outputs of the 13-80-2 split-ANN controller using an inverter: On-line training initiated at t = 1.0 s and linear change in motor parameters from t = 2.0 s to t = 2.5 s

change in motor parameters, and thus the performance would be better than shown here.

5.5 Summary

This chapter outlines a major portion of the theoretical work done for this thesis. The issues involved with ANN control of induction motor have been discussed first, and the basic equations for a traditional direct adaptive control method using ANNs have been derived. These equations are cumbersome and need a lot of computation time. The scheme itself is unstable and an ANN trained with this scheme was unable to even run the motor, let alone control the speed.

The next method proposed in this work is off-line control of induction motor using ANNs. The problems with off-line control are discussed, along with some benefits. After this, a simplified off-line control strategy is presented in which an ANN has been trained to mimic a vector controller. Simulation results for the same are also presented. Next, a full fledged ANN based induction motor control using a PWM voltage source inverter is presented along with simulation results. On-line control is discussed, and a computationally simple strategy for direct adaptive on-line control is presented. Simulation results for the same are also presented. It is shown that the on-line strategy is quite robust even in the presence of large changes in motor parameters.

The control schemes outlined in chapter fulfill a major objective of this work, viz. to control an induction motor using only an ANN and no other controller. It would be desirable to verify this strategy in real-time, but this could not be tried out due to limitations in available resources. However, keeping in mind the objective of experi-

mental verification of the ANN based schemes, the next chapter discusses a real-time implementation of an ANN based speed estimator discussed in the chapter 4.

Chapter 6

Experimental Verification of ANN-based Speed Estimation

The previous two chapters outlined the simulation studies which were undertaken as part of this research work. The simulations helped to establish a theoretical foundation for ANN based induction motor speed estimation and control. It has been noted earlier that most of the work done on the application of ANNs to induction motor drives has been simulation work, and very little experimental verification has been done. This could be in part due to the fact that dedicated ANN hardware is expensive and difficult to acquire, and most sequential machines cannot handle in real-time the large number of computations required for an average sized ANN. However, as part of this work, it was decided that some of the simulation studies done earlier must be verified experimentally. This chapter begins with a review of some of the commercially available ANN hardware and discusses the nature and scope of the proposed experiments. It then outlines the setup which was built for the experimental work. Finally, the experimental results are presented to demonstrate close conformity between the simulation and the practical

implementation [64].

6.1 A review of available ANN hardware

Some important benefits of using ANNs, which are cited by various researchers as justification for choosing ANN based methods, are inherent parallelism and robustness. For many practical applications, however, these benefits and other requirements can hardly be met by conventional algorithms running on sequential machines. Even though the simulations help to validate the theory and can effectively demonstrate the ability of the ANNs to map unknown nonlinear functions, the parallelism and robustness can only be exploited with dedicated ANN hardware.

Over the last few years, the field of ANNs has matured considerably and has prompted the development of both custom-built laboratory test benches as well as some commercial hardware. These systems are extremely heterogeneous and range from small systems for on-board applications to large computational servers often call *neurocomputers* [65]. Figure 6.1 shows a classification of available ANN hardware. As can be seen from the figure, implementations could be analog, digital, hybrid (mixed analog and digital) and optical [66]. The last one is relatively new and still in the early stages of development.

6.1.1 Optical implementation

Optoelectronic circuits that use optics for inter-connection provide the fastest way of implementing ANNs. They do not suffer from connectivity problems, because light emitted from different sources can cross without interfering, and thus the third dimen-

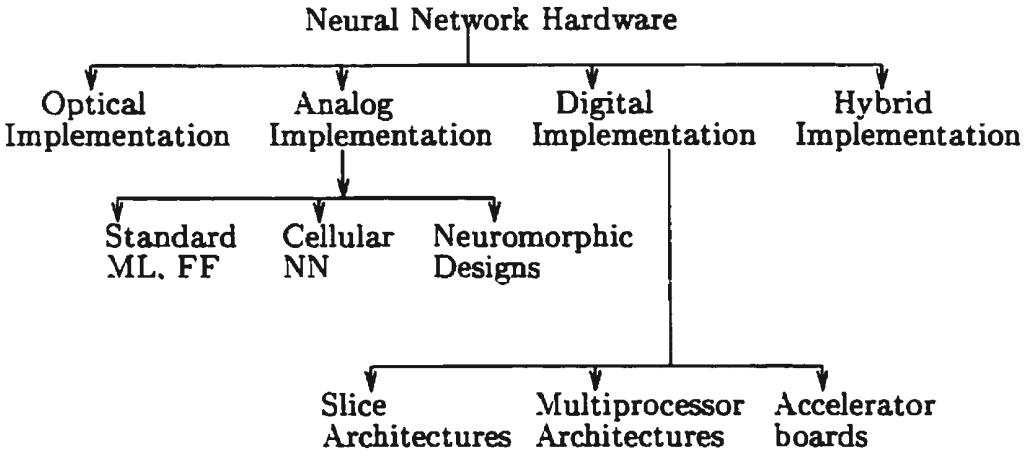


Figure 6.1: Classification of artificial neural network hardware

sion can be exploited to implement neural network functions, e.g., nonlinearity and weight storage. Besides light sources and receptors, spatial light modulators are the main component of optical systems. These can be used for storing weights, by changing the transparency or opacity of liquid crystal devices in proportion to the value of the weight. An interesting application is the GaAs-based retina chip which provides parallel image sensing and processing capability [66].

6.1.2 Analog implementation

Analog hardware is several orders of magnitude faster than digital, and can exploit some of the inherent qualities of ANNs, e.g. large scale parallelism and nonlinearity. However, analog hardware is very tricky to implement because of parameter spread in large circuits. Creating an analog synapse involves the complications of analog weight storage and the need for a multiplier linear over a wide range. Thus, like optoelectronic implementations, analog hardware has not found very widespread use.

Analog designs come in various forms. Some designs implement standard multilayer feedforward networks. The Intel 80170NW ETANN (Electrically Trainable Analog Neural Network) is one example. The design is quite flexible and allows for multiple configurations including 3-layers of 64 neurons/layer, and 2-layers with 128 inputs and 64 neurons. The ETANN is not designed for on-chip training, so a *chip-in-the-loop* mode with a PC is necessary. Following off-line training, the network configuration and weights are downloaded to the chip. Though this was probably the first commercially available analog ANN, Intel has since sold off its ANN division to Nestor Inc.

Cellular Neural Networks (CNN) are another kind of ANN, which are amenable to analog implementation. In CNN, the individual neurons are influenced only by a local neighbourhood of contiguous cells. This is a very attractive feature for hardware implementation, as it alleviates the connectivity problem present in large fully-connected multilayer networks. These are quite useful in certain image processing applications. A few implementations of CNN chips have been reported in literature [66].

Neuromorphic designs attempt to closely mimic the biological model of the neuron. The Synaptics Silicon Retina [67] is an example of this design.

6.1.3 Digital implementation

Digital implementations of ANNs are the most common, primarily because of a large existing digital infrastructure and compatibility with PCs and workstations. However, digital implementations are much slower than analog or optical implementations.

Digital implementations of ANNs can be broadly classified as slice architectures, multiprocessor architectures and accelerator boards [67]. Slice architectures provide building blocks for constructing networks of arbitrary structure and size.The Micro

Devices MD1220 was probably the first commercial neural network chip[8]. Each chip has eight neurons with hard-limit thresholds and eight 16-bit synapses with 1-bit inputs. Multiprocessor architectures have multiple small processors on the same chip, and each could perform the same or different computation depending on the hardware architecture and the degree of programmability. Accelerator boards typically have a single or couple of high performance processors (DSPs, RISC processors and so on) and can conveniently plug in into the PC GPIB. The degree of programmability is typically quite high, and many systems come with their own compilers for program development.

Digital systems can be further classified according to various sub-criteria like *numerical representation*, *degree of parallelism* and *inter-processor communication* network. To simplify the dedicated ANN hardware, fixed point numerical representation with reduced precision is often chosen. The degree of parallelism can vary widely from very fine grain to coarse grain or even sequential in some cases. Massively parallel systems, with a very fine grain of parallelism, can even map each synapse onto a simple processor (also called a *node* occasionally). Most moderately parallel systems map one or more neurons onto each processor. Single instruction-stream, multiple data-stream (SIMD) architectures often model one neuron per processor. In systems with a higher degree of programmability, the mapping of neurons onto processors depends on the user's choice. The inter-processor communication network could take many forms — *bidimensional mesh*, *systolic ring*, *broadcast bus*, *linear array* and so on.

6.1.4 Hybrid implementation

Hybrid designs attempt to combine the best of analog and digital techniques. Typically, the external inputs/outputs are digital to facilitate integration into digital sys-

tems, while internally some or all of the processing is analog. Examples of hybrid architectures are the AT&T ANNA chip and the Ricoh RN-200 chip among others. Table 6.1 [67] outlines some of the commercially available ANN hardware.

Typically, computation performance of hardware ANNs is measured in connections per second (CPS), or connection updates per second (CUPS). Each of these might be preceded by M or G, which stand for *million* and *giga* respectively. However, these benchmark values may not be fair when comparing different systems. One reason is that using a simpler numerical representation might speed up the number of CUPS, but the training process as a whole might take longer, or it may not be as effective as with a full floating point or double representation. Thus, it may not be easy to compare the performance of ANN hardware based on a few simple metrics.

Another difficulty posed by ANN hardware is that it is not very standardized. As the programmability of the system increases (to make it more flexible), the speed and degree of parallelism generally reduce. In fact, as Ienne et al [65] point out, in some cases, dedicated ANN hardware might actually be slower or moderately faster than a high performance serial workstation, or a high-end PC. Also, even though the prices of ANN hardware have come down in recent years, they are still considerably more expensive than serial hardware. In addition, the commercial market is still quite volatile, and some ANN hardware has disappeared from the market after a brief stint. Thus, buying dedicated ANN hardware is still a risky proposition, and the user must weigh all the available options before spending time, money and effort in developing ANN hardware based systems.

Table 6.1: Available ANN hardware [67]

Type	Name	Architecture	Learning	Neurons	Synapses	Speed
Analog	Intel ETANN	FdFwd, ML	no	64	10280	2GCPS
	Synaptics Silicon Retina	Neuromorphic	no	48x48	Resistive net	na
Digital	NeuraLogix NLX-420	FdFwd, ML	no	16	off-chip	300 CPS
	HNC 100-NAP	GP,SIMD,FP	program	100PE	512K off-chip	250MCPS 64 MCUPS
	IBM ZISC036	RBF	ROI	36	64x36	250k pat/s
	Micro Devices MD-1220	FdFwd,ML	no	1 PE	8	8.9MCPS
	Nestor/Intel NI1000	RBF	RCE,PNN	1PE	256x1024	40k pat/s
	Philips Lneuro-1	FdFwd,ML	no	16 PE	64	26MCPS
	Siemens MA-16	matrix ops	no	16 PE	16 x 16	400MCPS
Hybrid	AT&T ANNA	FdFwd,ML	no	16-256	4096	2.1GCPS
	Mesa Research Neuroclassifier	FdFwd,ML	no	6	426	21GCPS
	Ricoh RN-200	FdFwd,ML	BP	16	256	3.0GCPS

6.1.5 Choice of hardware for real-time ANN implementation

The preceding subsections discuss some specialized hardware used for implementing ANNs in real-time. As pointed out, commercial ANN hardware has not yet been standardized and is quite expensive. A DSP based accelerator board was also considered for the experimental work. However, a typical DSP based system these days is about 4 times as expensive as a high-end PC which can provide more computation power. On the other hand, DSP systems are optimized for real-time applications and usually come with a suite of software that makes the development of a control or estimation algorithm very easy and trouble free. In the end, it was decided that using a very fast sequential machine might be a better idea, because it would lead to a more general purpose hardware setup and the development time for the software would be very small, because most of the programs could be recompiled using a DOS based C++ compiler and used with minor modifications for real-time application. A high-end PC would be able to handle smaller sized networks with relative ease, providing the added flexibility of being able to use a regular C++ compiler. Since all of the simulation was implemented using C++, the transition from simulation to real-time implementation would be smoother.

6.2 Scope of the experimental work

Since a sequential machine was being used, it was decided to implement an ANN which would not be very large. The ANN speed estimator using method 4 outlined in chapter 4 seemed to be an ideal choice, because of the reasonable size, and also because it would permit the verification of the important idea of training with the help of the "DQ-MF" block, which is used for ANN control discussed in the previous chapter.

It was decided that if the experimental results agree with the simulated results to a good extent, then it can be said with a high degree of certainty that the ANN control methods outlined in chapter 5 should work as predicted. Furthermore, it was decided to use a commercially available drive to run the induction motor.

6.3 Description of the experimental setup

A block diagram of the experimental setup for real-time ANN speed is shown in Figure 6.2, and a photograph of the same is shown in Figure 6.3.

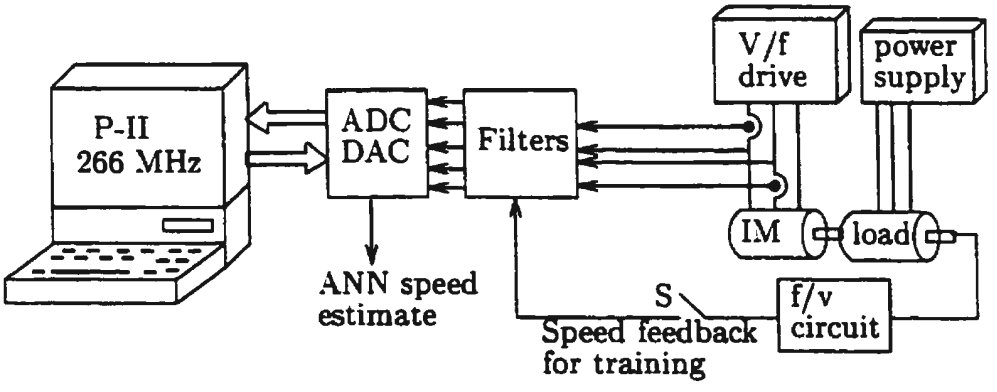


Figure 6.2: Schematic of the experimental setup

6.3.1 Hardware components

A Pentium-II computer running at 266 MHz was chosen for implementing the ANN speed estimator in real-time. The operating system for this machine is MS-DOS version 6.0. The reason for using a DOS based system was the simplicity it offers, in terms of

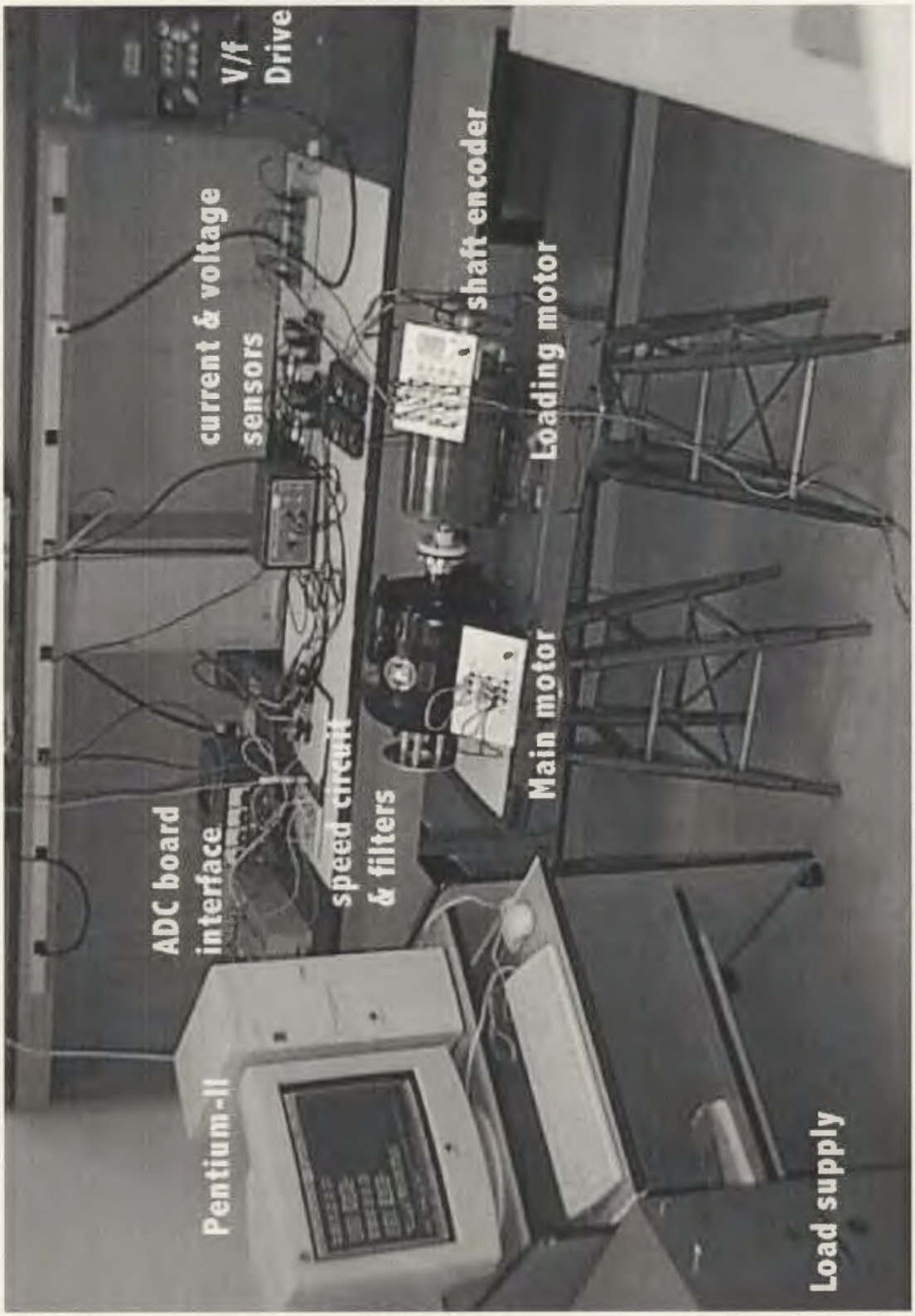


Figure 6.3: Photograph of the experimental setup

accessing the input and output ports, as well as program portability. A Borland Turbo C++ compiler was used for program development. Because of this arrangement, the actual program development time was drastically reduced.

For obtaining the current, voltage and speed feedback, an RTI-815-F board was used [68], which plugs in conveniently into the PC's AGP 100 MHz bus. The RTI-815-F is a multifunction analog/digital I/O board that has capabilities for analog input, analog output, digital input and output, and time-related digital I/O functions (through the AM9513A Counter/Timer chip). It has 16 analog input channels with 12 bit A/D resolution, and A/D ranges of 0 to +10V, ±5V, or ±10V can be selected. The A/D conversion time is typically 8μs. There are two analog outputs with a 12 bit D/A resolution and a 20μs settling time for a +10V step. The board also has an 8-bit digital input port and an 8-bit digital output port.

The current feedback was obtained with two Hall effect LEM current modules with a ratio of 1:1000. Voltage feedback was obtained with SI 9000 differential probes with an attenuation ratio of 1:50. An optical, incremental shaft encoder was used for obtaining speed feedback. The encoder produces 1000 pulses per revolution and has two separate channels which are phase shifted by 90° to allow estimation of the direction of rotation. There are various ways of obtaining the actual speed from the encoder pulses. Since the encoder produces high frequency pulses (about 30 KHz for a 4 pole induction motor running at rated speed), conversion from frequency to voltage produces a very accurate estimate of the motor speed. Such a circuit is very easy to implement, using one of many commercially available frequency-voltage converters. Some extra circuitry has to be used to estimate the direction. The circuit used in the experimental setup is shown in Figure 6.4. This produces two separate signals, one for the speed magnitude and the other for the speed direction. The direction signal is a digital signal, and both the signals can be read into the Pentium-II using the RTI-815-F ADC/DAC card. It should

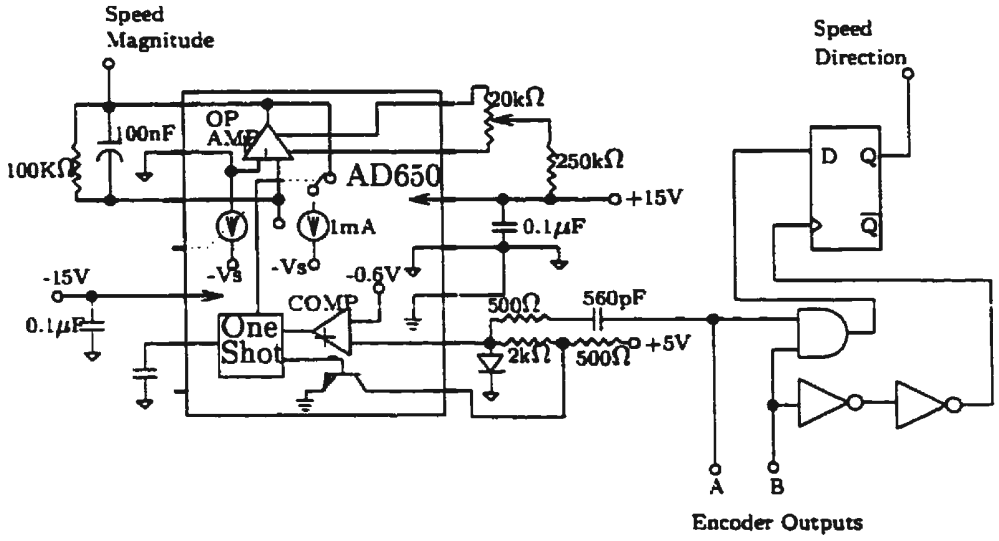


Figure 6.4: Speed measurement circuit

be noted here that a shaft encoder has been used in the setup only for the purpose of obtaining ANN training data. In a sensorless scheme, the trained ANN should be able to provide a good estimate of the speed, obviating the need for a shaft encoder. If a shaft encoder cannot be mounted for the purpose of ANN training, it is possible to use a contact-less tachometer for this purpose, though the accuracy of the ANN speed estimate would be limited by the accuracy of the measured speed in the training data.

The induction motor runs on a PWM voltage supplied by the IGBT inverter in the V/f drive, and this voltage signal cannot be used directly, since the instantaneous magnitude and $\Delta\theta$ values cannot be obtained from such a signal. Thus, the fundamental component of the voltage signal must be extracted from the PWM signal. Also, the current waveform is not a pure sinusoid, because of the PWM nature of the voltage. This necessitates the use of a filtration stage prior to the analog/digital conversion. Also, the speed signal obtained from the frequency-to-voltage conversion circuit has noise. This noise probably results from the internal operation of the frequency-to-

voltage converter. Thus, it is necessary to filter all the five feedback quantities before use. For purposes of filtration, a *Sallen and Key* second order active filter was chosen, because of the simplicity and efficiency of the circuit. A circuit schematic of the filter is shown is Figure 6.5. The transfer function of the filter is given by

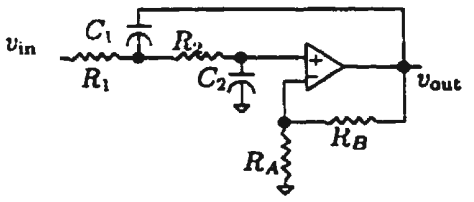


Figure 6.5: Circuit schematic of the Sallen and Key filter

$$H(s) = \frac{K\omega_c^2}{s^2 + \frac{\omega_c}{Q}s + \omega_c^2} \tag{6.1}$$

where,

$$K = 1 + \frac{R_B}{R_A} \tag{6.2}$$

$$\omega_c = \sqrt{\frac{1}{R_1 R_2 C_1 C_2}} \tag{6.3}$$

$$\frac{1}{Q} = [\sqrt{\frac{R_2 C_2}{R_1 C_1}} + \sqrt{\frac{R_1 C_2}{R_2 C_1}} + (1 - K)\sqrt{\frac{R_1 C_1}{R_2 C_2}}] \tag{6.4}$$

To simplify the filter design, it was assumed that $R_1 = R_2$ and $C_1 = C_2$, and the circuit values were chosen to obtain a cut-off frequency (ω_c) of about 300 rad/s. The gain was adjusted such that the output signal occupies most of the available range of the ADC ($\pm$10V). Identical filters were used for the current and voltage signals and for the circuit components chosen, the total harmonic distortion (THD) of the filter output was about 3.5%, which implies that the output signal was quite clean. For the speed feedback signal, a lower cutoff frequency of 100 rad/s was chosen, because high frequency components are not expected in the speed due to the inertia of the motor

and the fact that the drive accelerates and decelerates the motor at a predetermined rate.

It should be noted that the use of the filters would lead to a delay in the signals. For the current and voltage signals, this delay manifests itself as a phase shift. However, after conversion from DQ to MF, the phase information is lost, and thus phase shifted current and voltage signals make no difference to the ANN operation. For the speed signal, the filter has to be carefully chosen, because a delay here would result in a delay in the ANN output. Fortunately, the speed dynamics are significantly slower than the current and voltage dynamics and no appreciable delay results due to filtering.

For controlling the induction motor, a commercially available 1 HP V/f drive was used, which can operate in all four quadrants. The drive has an IGBT inverter running at 3.3kHz. An open loop frequency control method was used, since it is simpler and does not affect in any way the generality of the ANN based speed estimation scheme. Closed loop control is also possible with this drive. The drive parameters can be programmed from a convenient digital keypad with an LCD screen. For most applications, the factory settings are good enough, though the acceleration and deceleration rates were readjusted for this work. Also, the drive had to be reprogrammed to enable reversal of speed.

For loading the induction motor, another induction motor was mounted on the same frame with the shafts of the two motors coupled together. A variable three-phase power supply was used for applying a voltage to the loading motor, with a phase sequence which was opposite to that of the main induction motor. This system is quite efficient and enables the application of step changes in load. The parameters of the induction motor used for speed estimation are given in Table 6.2. It should be noted here, that these parameters are not used anywhere for ANN training, since the ANN is trained to learn the motor characteristics by observing the machine inputs and outputs.

Table 6.2: Parameters of the induction motor used for experimental verification

Parameter	Symbol	Value
Power Rating		1.0 HP
Voltage		208V
Connection type		Y
Stator Resistance	R_s	2.75 Ω
Rotor Resistance	R_r	2.45 Ω
Stator Inductance	L_s	169 mH
Rotor Inductance	L_r	169 mH
Magnetizing Inductance	L_m	160 mH
Moment of Inertia	J	0.0630 $N-m^2$
Damping Coefficient	B	0.0030 $N-m^2/s$
Pole Pairs	P	2

6.3.2 Software components

For the purpose of carrying out the experimental work, the main addition to the suite of software described in chapter 3 was a class called *RTI_board* for handling the input and output from the RTI board described earlier in this section. It has the following functions which greatly simplify the task of the user.

- *in_voltage*: This function reads an analog voltage from the channel number which is passed as a parameter to the function. It returns the actual value of the voltage read, which has been calibrated against an oscilloscope.
- *out_voltage*: This function outputs a specified voltage to a specified channel, both of which are passed as parameters to the function.
- *digital_in*: This function reads a 1-bit digital value from specified bit number in the 8-bit digital I/O port on the RTI board.
- *digital_out*: This function writes an 8-bit digital value to the digital I/O port on the RTI board.

As has been mentioned in chapter 3, the ANN simulator has been written to provide a great deal of flexibility and reliability. However, because of the above features, it has not been optimized for speed. Also, the simulator is too large to run on a DOS platform. To circumvent these problems, it was decided to implement another ANN simulator which would simulate only fully-connected multi-layer feedforward networks without any training algorithm. This simulator is much smaller in size, with a lot less flexibility, but has been optimized for speed. The computation of the exponential function, which is required for the neuron activation function, has been implemented using a lookup table with 600 elements, since this saves a lot of time. Training of the ANN has to be

done off-line using the larger and more flexible UNIX based simulator. To facilitate the transfer of the weight set between the two programs, both of them follow exactly the same format for reading the weight file from the disk.

Another important modification in the real-time simulator was the "DQ-MF" block described in chapter 4. The need for modification resulted from the fact that computation of $\Delta\theta$ in particular, assumes sampling at a constant and fast rate, which would not be possible in real-time implementation, since the ANN also has to be run along with the "DQ-MF" block. Thus, it was felt necessary to take three equally spaced voltage and current feedbacks and use them for computing the present and delayed values of magnitude and $\Delta\theta$. Also, this block directly converts the 'a' and 'c' phase values to magnitude and $\Delta\theta$ to save time.

As pointed out in chapter 4, after conversion from DQ to MF, a second stage of filtration is required. In the real-time implementation, this has to be done with a digital filter which does not depend on critical timing and uses as few previous values as possible. Thus, it was decided to use a modified averaging filter, which uses just three previous values and computes the slope of the input to reject higher frequency components. This filter has been found to be very useful for such an application. The performance of this filter will be demonstrated in section 6.4.

The main program was written using the components described above and some others described in chapter 3. This program operates in two modes. In the *data collection* mode, it just collects the ANN training data using the RTI board and stores it onto the disk after applying the "DQ-MF" transformation and filtration. In the *feedforward ANN* mode, the data is handed over to the ANN, instead of being stored on the disk. The ANN output, which is an estimate of the induction motor speed, is converted to an analog voltage by the RTI board and can be seen on the oscilloscope, alongside the actual speed signal obtained from the f/v circuit shown in Figure 6.2.

6.4 Experimental results and discussion

The experimental verification was conducted by first collecting different training data sets, by running the V/f drive at various frequencies, under both forward and reverse modes of operation and applying step changes in the load. These different sets of data were combined together to generate a larger data set for off-line training. After the training was complete, and various networks and learning rates were tried out to get an optimum output, the ANN weights were stored in a weight file, which could be directly read by the real-time ANN program.

The 'a' phase voltage feedback signal before and after filtration is shown is Figure 6.6. As has been mentioned earlier, the THD for this waveform is only 3.5%, which should be clean enough for conversion from DQ to MF. The 'a' and 'c' phase filtered voltage waveforms, separated by 120°, are shown in Figure 6.7. The 'a' phase current feedback signal before and after filtration is shown is Figure 6.8. It can be seen that the current feedback signal from the LEM sensors has spikes due to the inverter operation. The 'a' and 'c' phase filtered current waveforms, separated by 120°, are shown in Figure 6.9.

It was seen earlier in chapter 4 that the magnitude and $\Delta\theta$ for stator currents and voltages have ripples due to imperfect filtering of these waveforms. These ripples would impede ANN training, and thus a second stage of digital filtering is required, this time for the magnitude and $\Delta\theta$ of the sinusoidal quantities. In the experimental setup, there is one more reason for noise in the magnitude and $\Delta\theta$ of the stator voltage and current, in spite of the improved filtering using the second order Sallen-Key filter. The reason is that high-end PCs have a lot of dynamic features like superscalar architecture, branch prediction and so on, which improve the overall speed, but make the system less suitable for real-time applications, since the execution time of the code might vary from one run

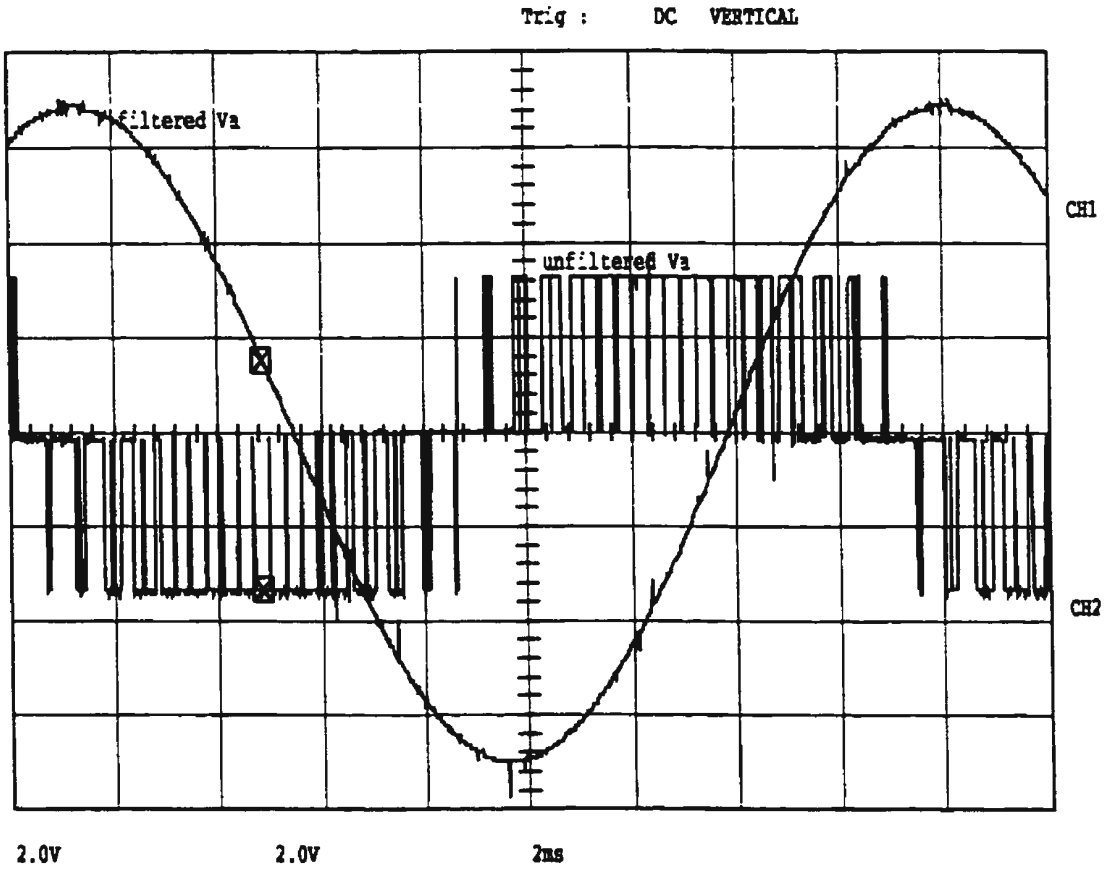


Figure 6.6: Inverter voltage before and after filtration

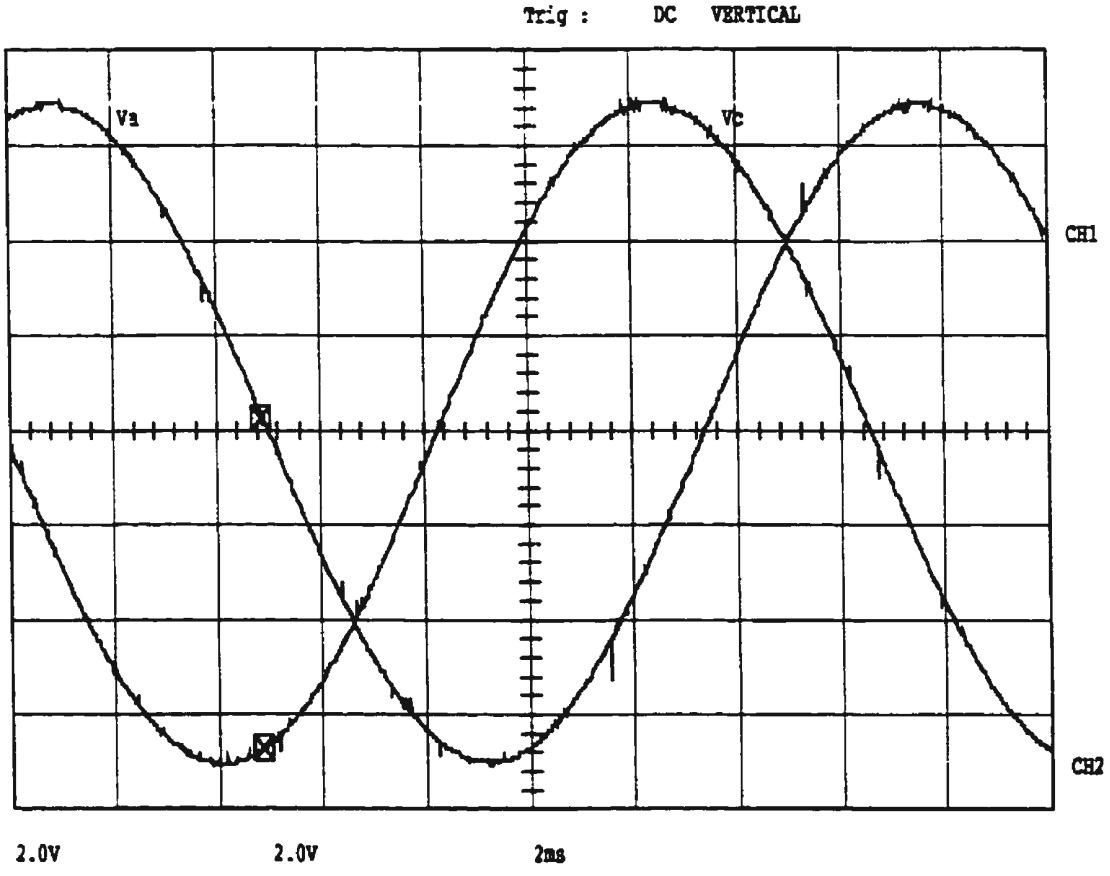


Figure 6.7: Filtered inverter voltages for phases 'a' and 'c'

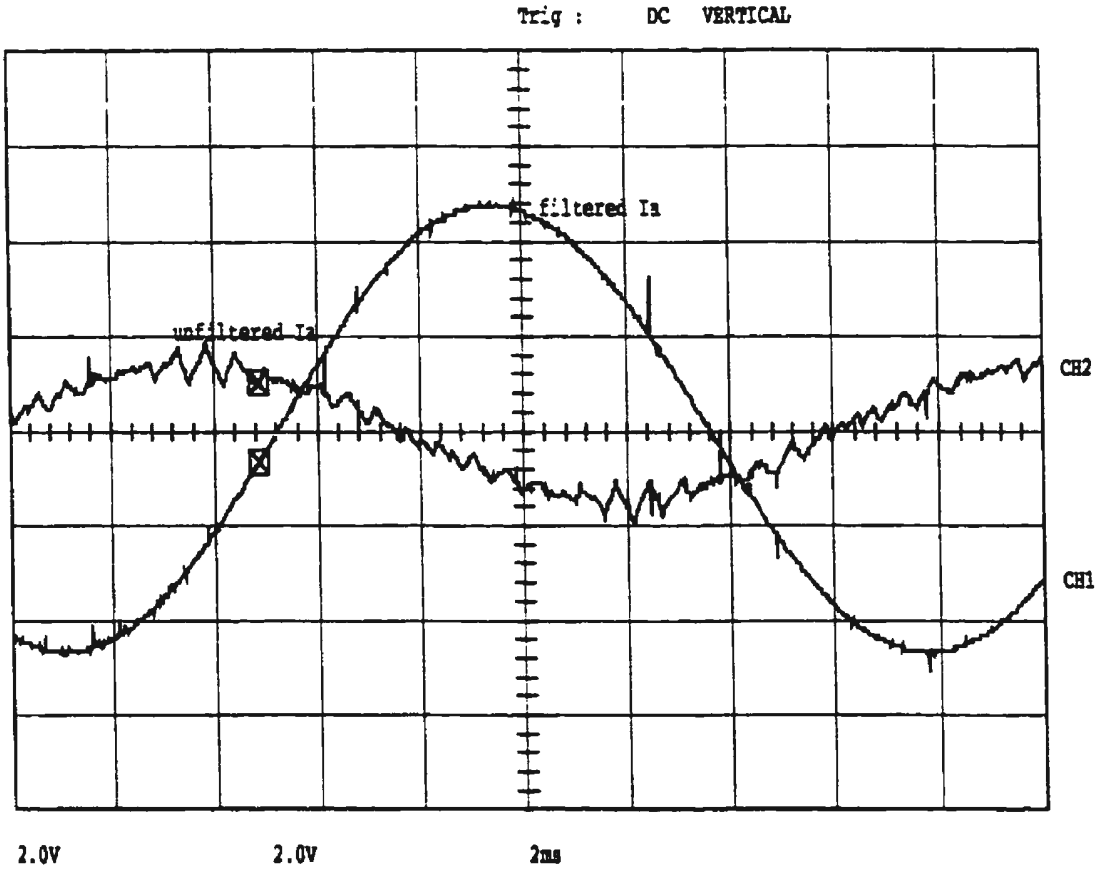


Figure 6.8: Inverter current before and after filtration

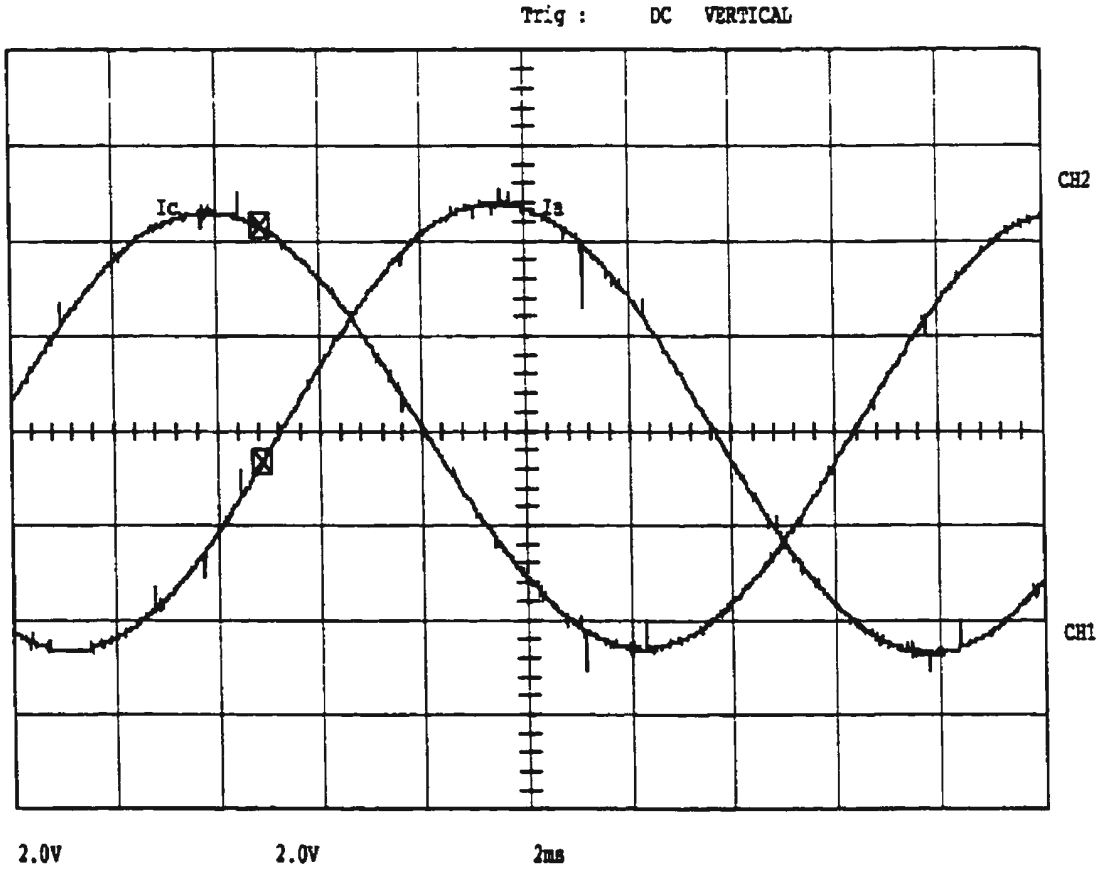


Figure 6.9: Filtered inverter currents for phases 'a' and 'c'

to another [69]. This implies that the computation of $\Delta\theta$, which assumes a constant sampling frequency, would have increased ripples. This problem can be rectified with a clock-based interrupt driven system, which would ensure that the loop time remains reasonably constant. However, for this work, a high performance digital filter was used, which has to operate under the constraints of a variable sampling time period and fewer available previous values. The average filter discussed in subsection 6.3.2 was designed with these objectives in mind and uses just 2 previous values to achieve a good output. The performance of the filter, on a 1000 point sample of the $\Delta\theta$ for the stator current, is shown in Figure 6.10.

The chosen network had a 6-20-1 architecture and was trained for 26 epochs on the data set, and the learning rate was 0.08 for the output neuron and 0.80 for all other neurons. The Pentium-II was able to run the ANN, along with the sampling and "DQ-MF" conversion and the six input filters, in about 400μs. This is a very impressive speed and demonstrates the feasibility of this approach. A comparison of the ANN estimated speed and the actual speed is shown in Figure 6.11. The data in this figure has been collected while the ANN was running on-line. The percentage error in the ANN speed estimate is shown in Figure 6.12. The sum squared error for this ANN during training is shown in Figure 6.13. Figure 6.14 shows the ANN speed estimate (CH1) along with the actual speed signal from the f/v circuit (CH2) in the forward mode of operation. The ANN speed estimate has a steady state error of less than 1.0%. Figure 6.15 shows the ANN speed estimate (CH1) along with the actual speed signal from the f/v circuit (CH2) in the reverse mode of operation. The ANN speed estimate has a negligible steady state error for this case. Figures 6.16 and 6.17 show the ANN performance for 30 Hz. operation of the drive. Here again, the error is about 1% for the forward mode of operation and almost negligible for the reverse mode.

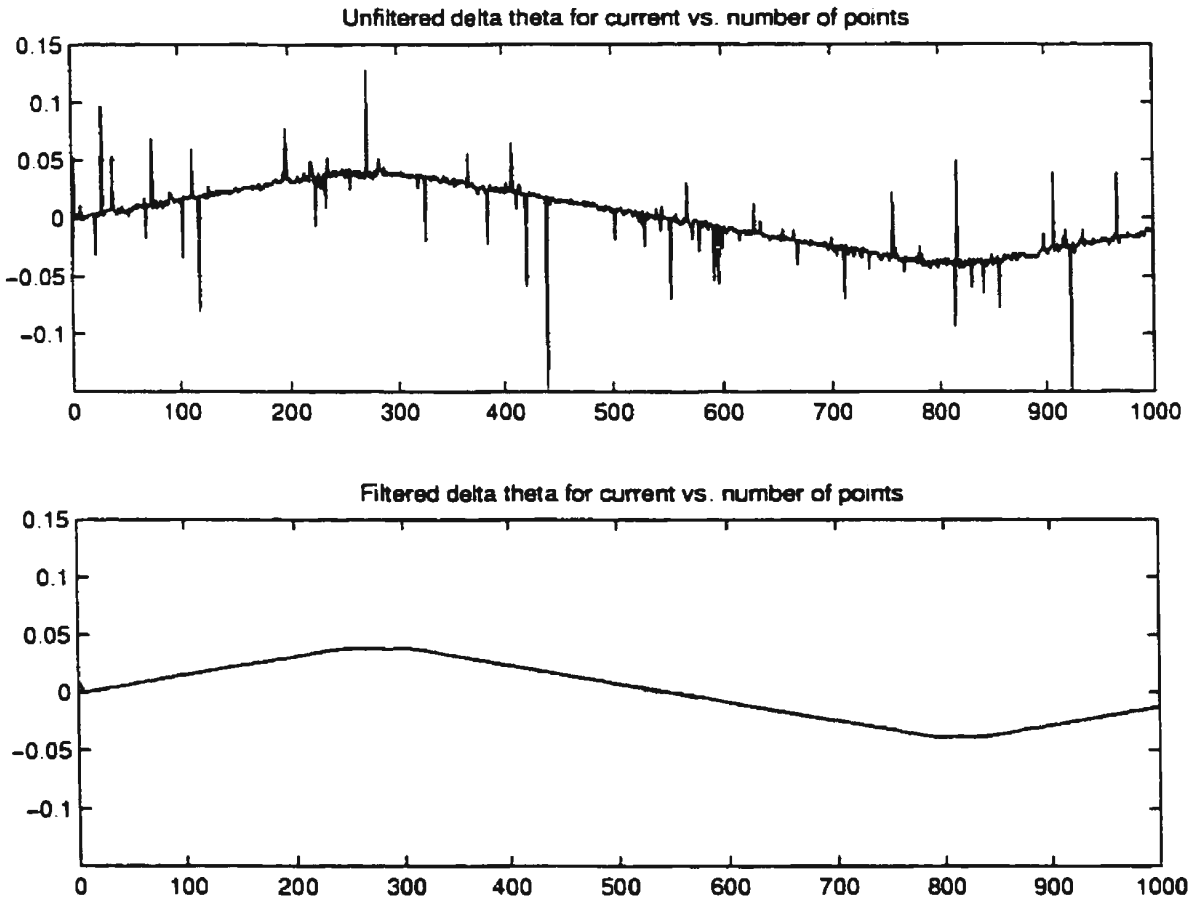


Figure 6.10: Performance of the modified averaging filter

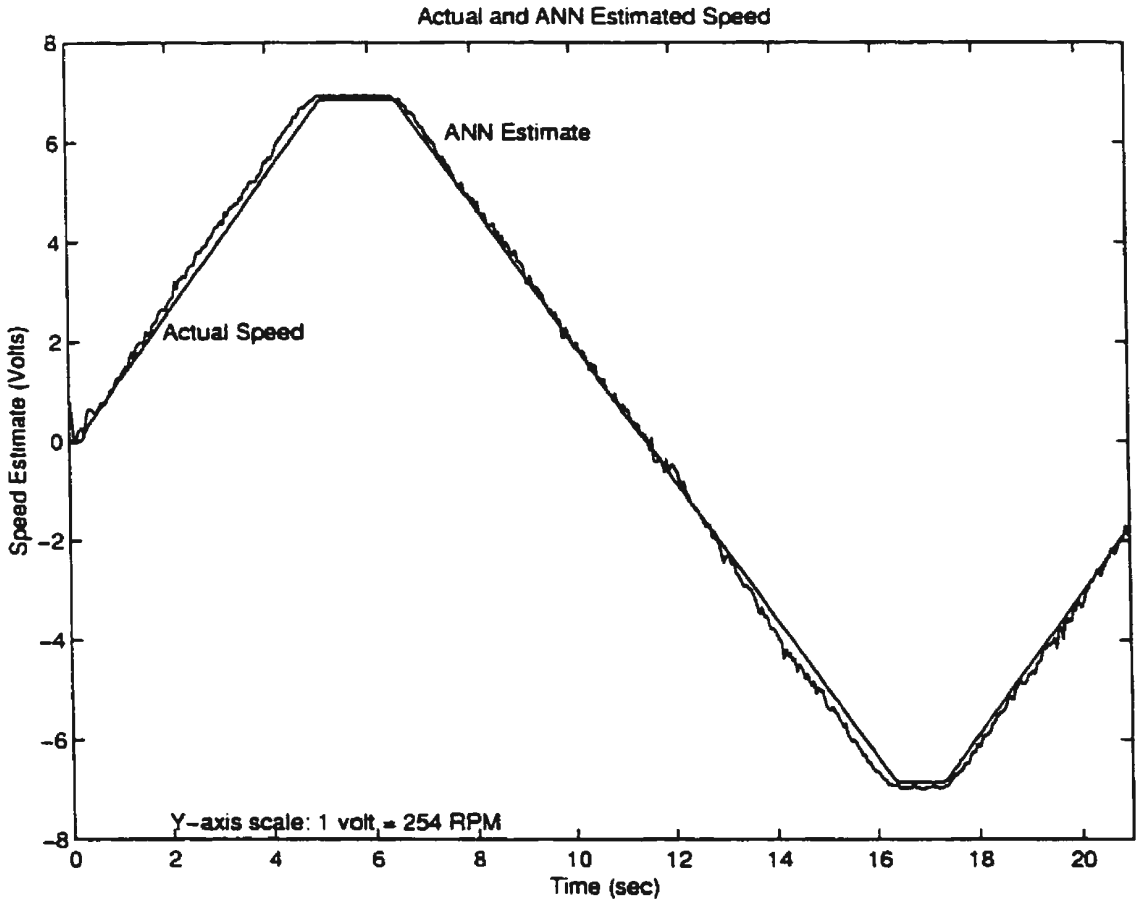


Figure 6.11: Actual vs. ANN estimated speed (real-time implementation)

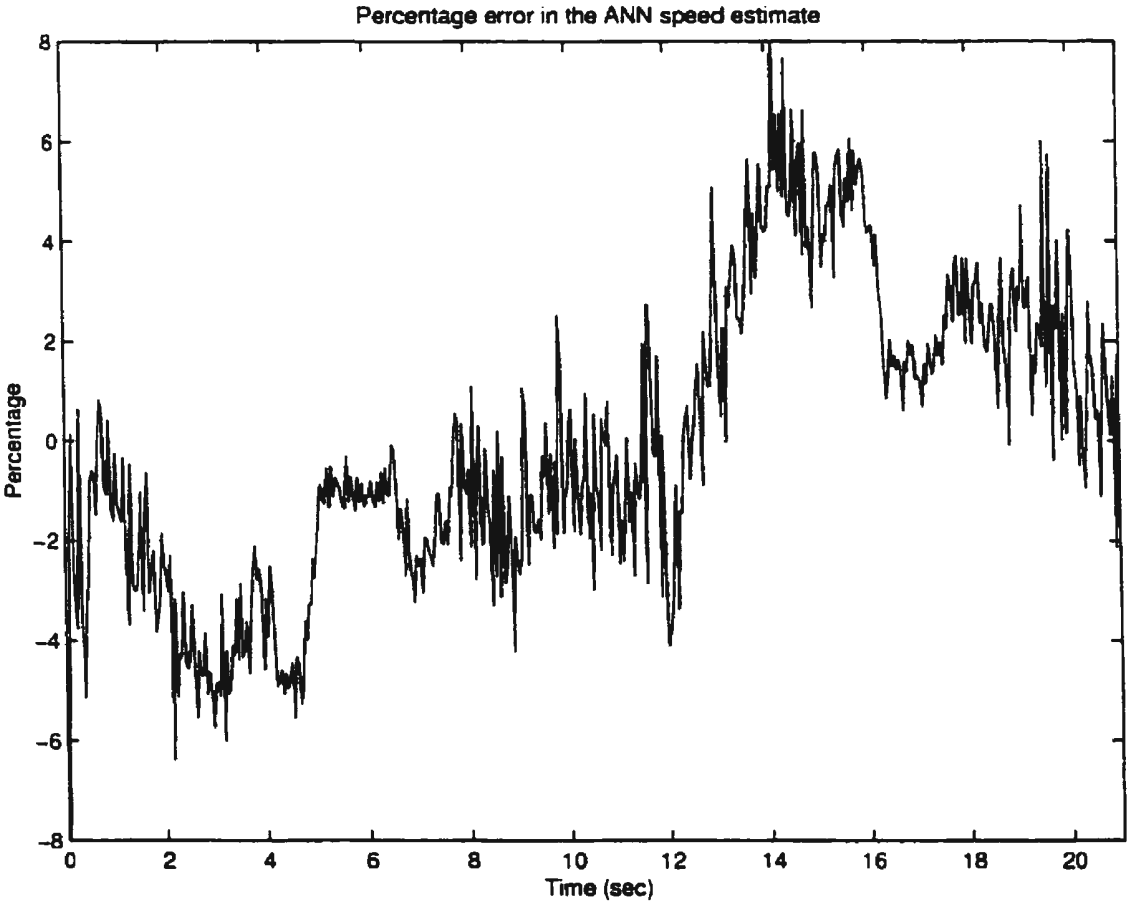


Figure 6.12: Percentage error in the ANN speed estimate (real-time implementation)

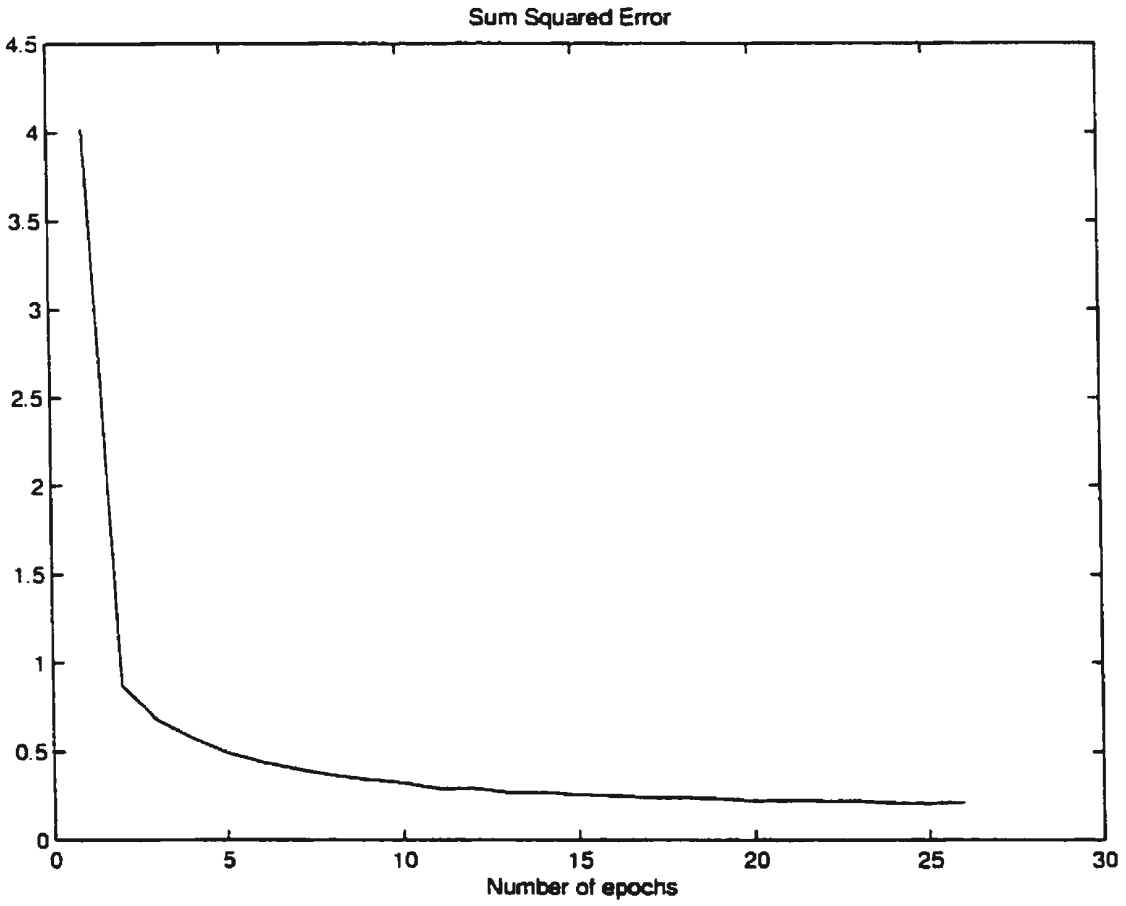


Figure 6.13: Sum squared error for the experimental ANN during training

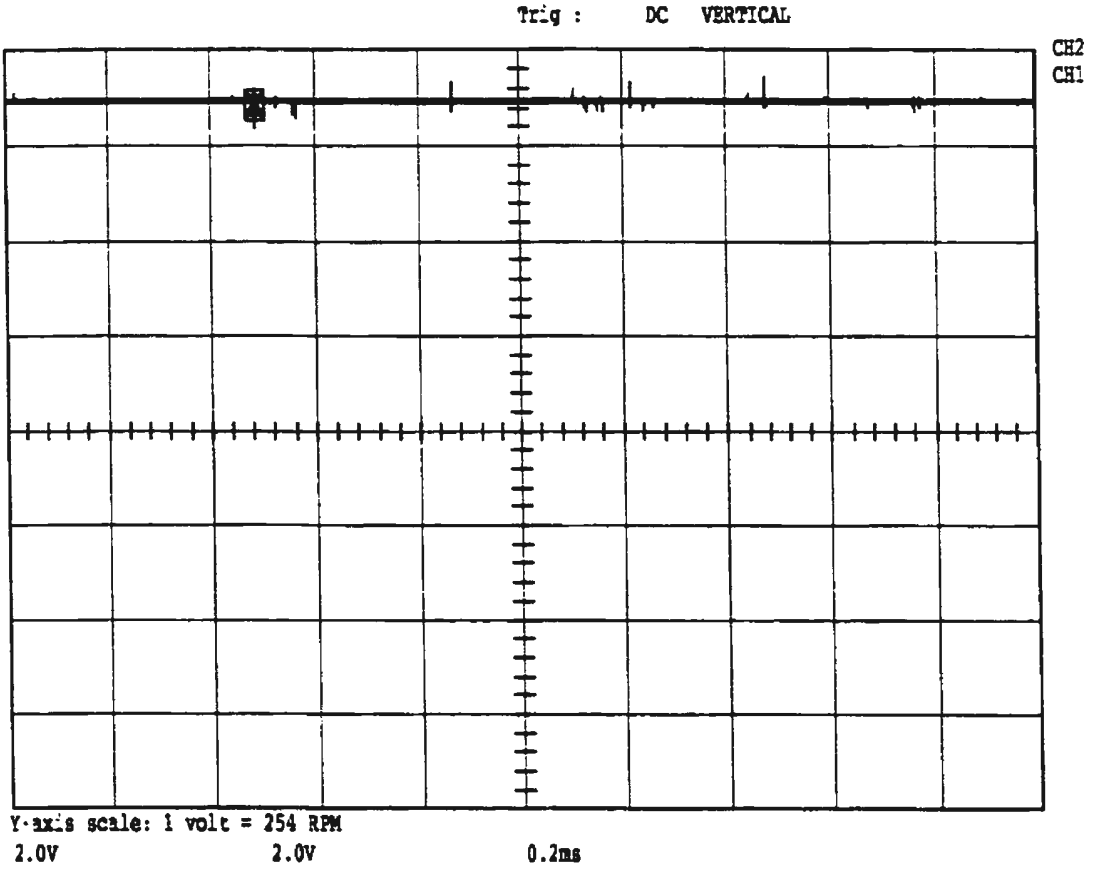


Figure 6.14: Actual and ANN estimated speed for 60Hz operation in the forward mode of drive operation

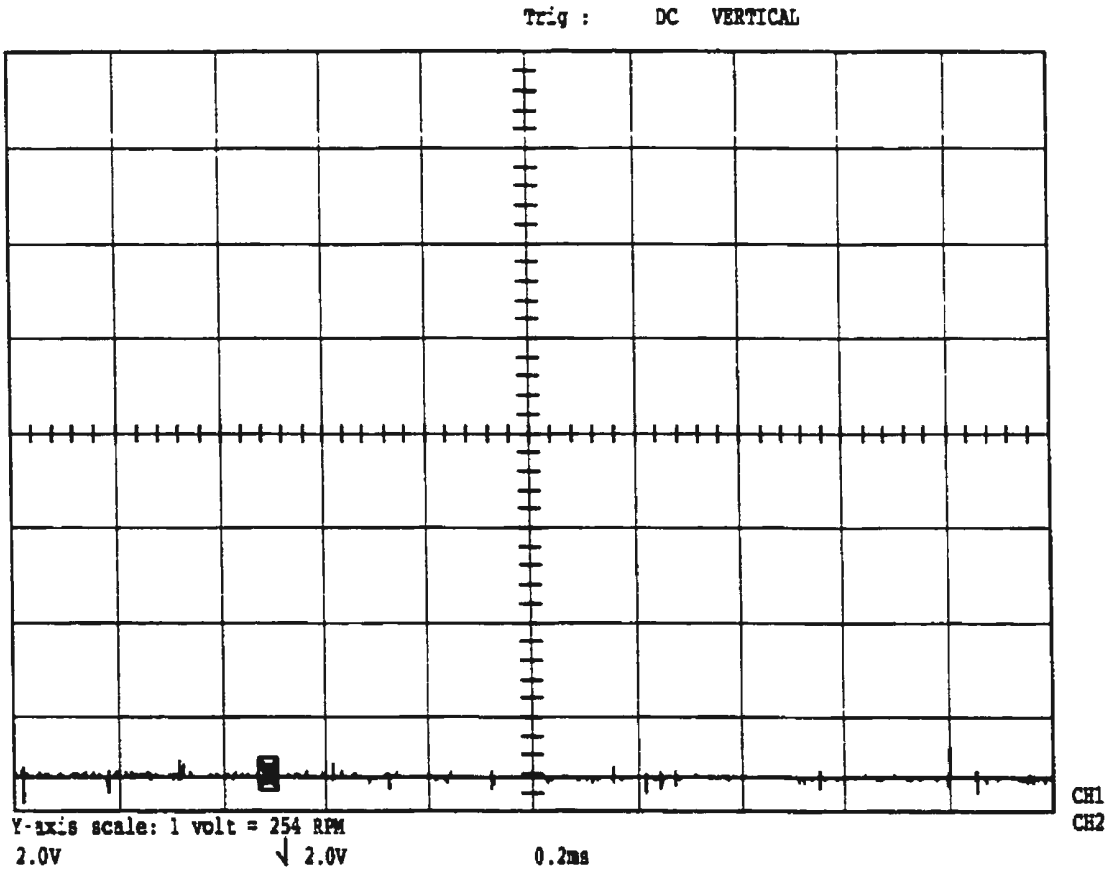


Figure 6.15: Actual and ANN estimated speed for 60Hz operation in the reverse mode of drive operation

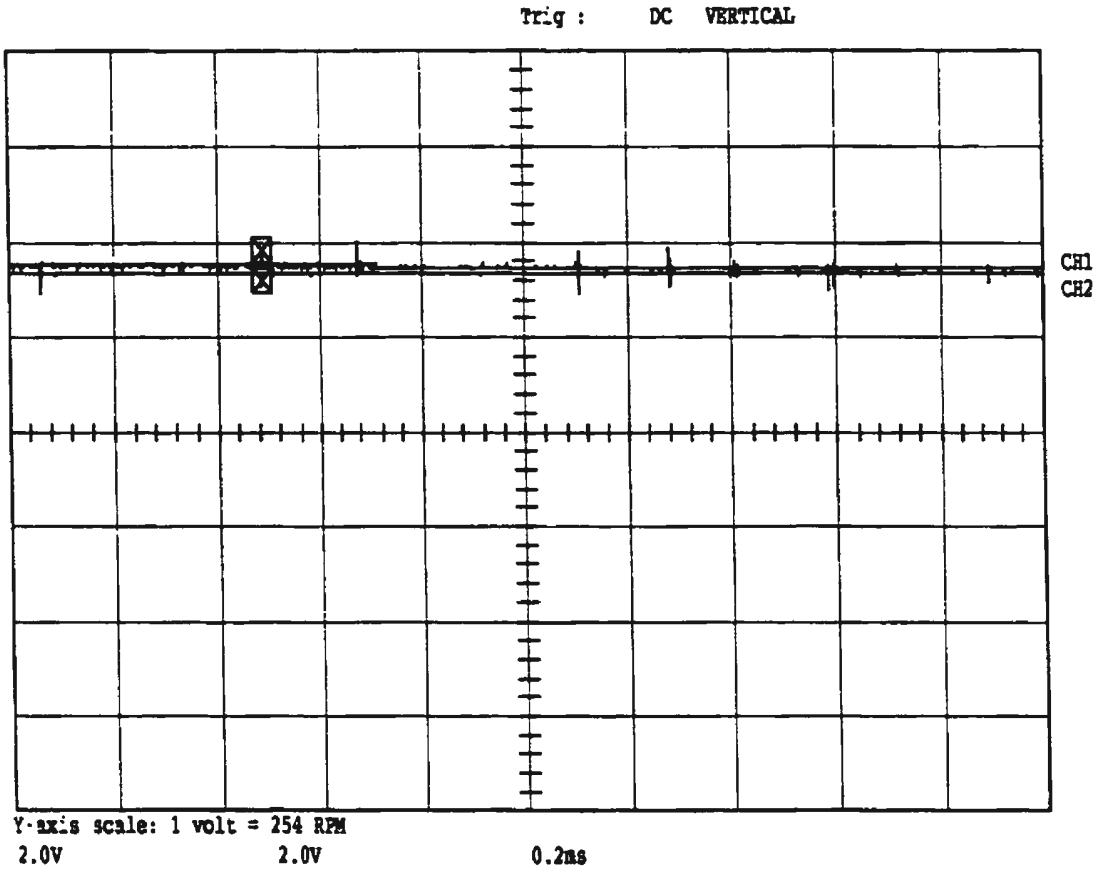


Figure 6.16: Actual and ANN estimated speed for 30Hz operation in the forward mode of drive operation

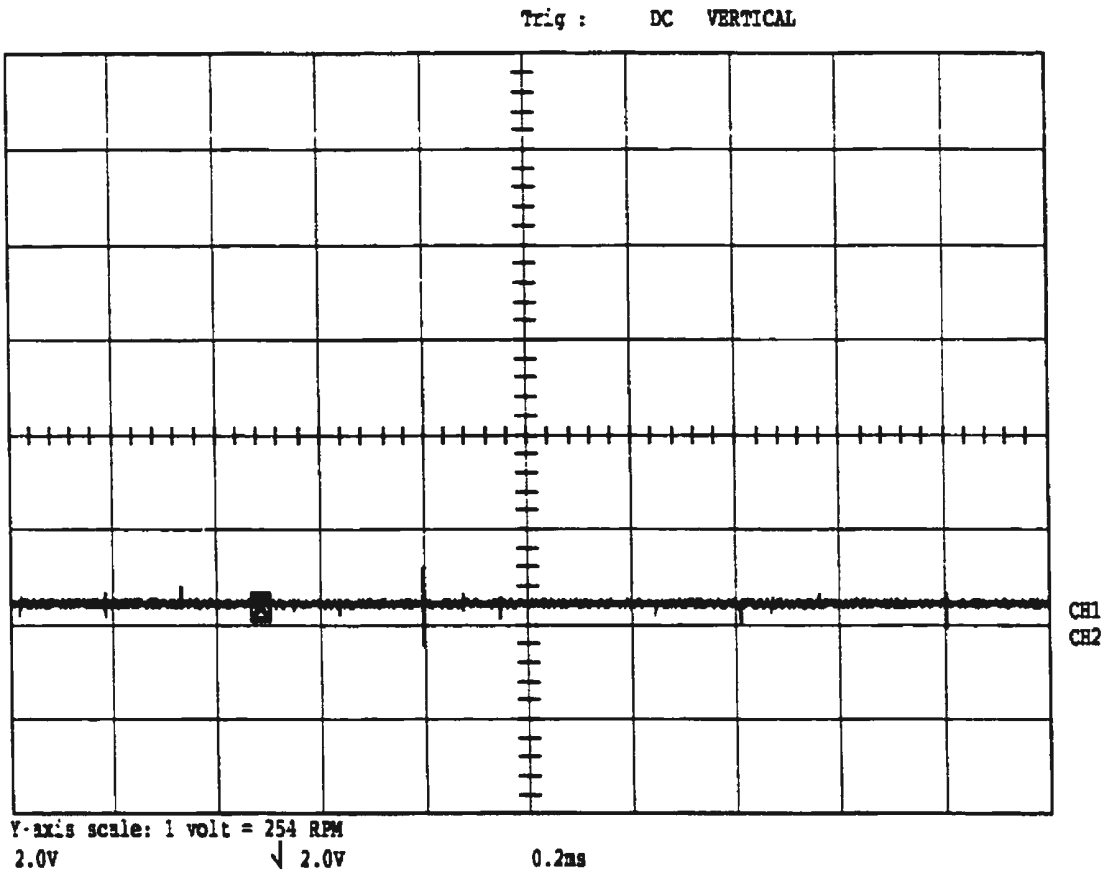


Figure 6.17: Actual and ANN estimated speed for 30Hz operation in the reverse mode of drive operation

The results presented in this section clearly indicate the efficacy of the ANN speed estimator for real-time applications. The ANN performs very well, and the results are comparable to a similar network demonstrated in chapter 4, even though the latter was trained with a simulated vector controlled drive. In fact, this proves the generality of the scheme, and the same method can be used with any other drive, as long as the ANN is suitably trained. This ANN architecture is by no means optimum, and a higher performance ANN can be obtained, if more time is spent on trying out different network sizes and learning rates. However, the primary scope of this thesis was to establish the theoretical basis and feasibility for the proposed scheme, rather than attempt to obtain the most efficient and commercially viable estimator.

This experimental verification also helps to validate the theory and simulation studies presented for ANN control of induction motor in chapter 5. If two control outputs from the control circuit of the V/f drive to the inverter can be obtained, for example the magnitude and frequency of the desired voltage, or i_{ds}^{*}, i_{qs}^{*}, then the same setup can be used for induction motor control. The only thing that would change would be the training data collected and the outputs of the ANN. After off-line training this ANN could be used for mimicing the V/f controller. If a vector controller was not available, it would not make any difference to the scheme outlined, since the ANN can mimic any kind of controller. Presently, the size of the ANN used in the simulation studies for control is too large to be run in a convenient time frame even on a high performance PC like the Pentium-II 266MHz machine used in this setup. Also, a split-ANN cannot be implemented using the real-time ANN simulator, which does not have the same flexibility as the one used for simulation studies. Even if the speed requirements were satisfied, it would be desirable to have a clock-based interrupt driven system as mentioned earlier. This is more crucial for the ANN controller, because unlike a speed estimator, the controller outputs affect its own inputs, thereby making it a vastly more challenging problem.

6.5 Summary

The aim of the work presented in this chapter was to demonstrate the feasibility of the proposed schemes and to verify some of the theory and simulations outlined in chapter 4. A review of the ANN hardware shows that procuring dedicated hardware for ANN experiments is still not a very attractive option. Thus, the approach taken for this experiment was to build a setup based on a high-end PC, which should be able to handle a large number of computations in real-time. The experimental results prove the efficacy of the scheme, and show that an off-line trained ANN can be used for high performance speed estimation of induction motors.

Chapter 7

Conclusion

The work done for this thesis has led to significant contributions to the area of ANN based induction motor drives. The objective of obtaining an accurate ANN based speed motor estimator has been achieved, as demonstrated in chapter 4. Four methods of induction motor speed estimation using ANNs are discussed in this chapter. Speed expressions derived from the d-q axis dynamic equation of the induction motor form a basis for the first three methods. These methods function well if the motor is operated with a power supply, but do not produce satisfactory output if the motor is operated with a vector-controlled drive. A fourth method is proposed in which the ANN is able to function quite well in the presence of a vector-controlled drive, and is also small enough to be implemented in real-time.

The objective of implementing induction motor control using only an ANN has been achieved as shown in chapter 5. In this chapter, a scheme has been presented, which implements induction motor control with an ANN which has been off-line trained to mimic a vector controller. The performance of this ANN is quite good and it is able to generalize effectively. Having a controller which can function with only off-line

training is a big advantage, since it saves a lot of computational time for real-time implementation. However, this ANN controller has a small steady state error and another scheme is also presented, whereby the off-line trained ANN can be on-line trained to minimize this steady state error. This on-line scheme has another benefit in that it is computationally less demanding than other on-line training schemes, because only the signs of the quantities in the Jacobian vector are used. Both of these methods were previously unreported in the literature.

The experimental verification objective has been partly achieved as outlined in chapter 6. Here, the ANN based speed estimator using method 4 in chapter 4 was implemented in real-time using a Pentium-II PC based hardware. The performance of this real-time estimator is very good, and clearly demonstrates the practicability of the scheme. The ANN based control strategy could not be verified in real-time because of hardware limitations. It has been pointed out in chapter 6 that dedicated ANN hardware is not yet standardized, and the network size for the ANN controller is too large for proper real-time implementation using the available Pentium-II based setup. Also, ANN control would require that two control outputs be obtained from the motor drive for the purpose of collecting testing data. These outputs are not readily accessible in the motor drive used in the setup. However, as mentioned in chapter 5, verification of training with the "DQ-MF" block has been done for the speed estimator, and this lends credibility to the ANN based control scheme which uses the same block for training.

7.1 Contributions of this work

The work outlined in this thesis has made some important contributions to the area of induction motor control using ANNs. These contributions are discussed below:

- *Speed estimation of induction motor*: Prior to the publication of some of the work outlined in chapter 4, almost no work was done in the area of induction motor speed estimation using ANNs, even though many alternate techniques were used for speed estimation, as was seen in the literature review. This work presented an off-line trained ANN speed estimator for the first time. The speed estimation scheme outlined in chapter 4, which has been experimentally verified, is also a first. To date, the author has not come across any other ANN based induction motor speed estimator, which has been verified in experimentation.

- *Induction motor control using off-line trained ANN*: This is probably the biggest contribution of this work. To date, the author has not come across any work reporting satisfactory and complete induction motor control, using just one or more ANNs, and without using any other conventional controller. Wishart and Harley's work [45] is probably the best work on ANN control of induction motor presented so far, and it uses one conventional PI controller along with the ANN. Also, the strategy cannot handle step changes in load torque, thereby severely limiting its utility. Lastly, it uses on-line training, which requires extremely high computing power to implement in real-time. The work done for this thesis has demonstrated induction motor control using a single ANN, which is off-line trained to mimic an existing high performance controller and which can handle step changes in load. For better performance, the ANN can also be trained on-line and this leads to improved steady state response and robustness in the presence of motor parameter variations.

- *Object-oriented simulator*: The suite of software developed as part of this research, and discussed in chapter 3, has proved to be invaluable for this work. Also, the simulation programs have been written in a way such that it becomes very easy for anyone to use them, with only a minimal knowledge of C++ programming. The ANN control schemes presented in chapter 5 would have been

impossible to implement, if the only available option was a conventional ANN simulator like the MATLAB neural network toolbox.

7.2 Suggestions for future work

The following are some suggestions for future work in this area.

1. *Experimental verification of the off-line trained ANN controller*: This should be possible with a high performance drive in which the two control outputs to the inverter can be tapped. The rest of the experimental setup would not change much. The basic idea would be to collect data, which would comprise a set of inputs (reference speed, actual speed, and magnitude and $\Delta\theta$ of the stator voltage, along with the previous values of all these quantities) and a set of desired outputs (magnitude and $\Delta\theta$ of the reference voltage or current). A split ANN could then be trained to mimic the controller as outlined in chapter 5. A faster PC might be required, with which a larger network could be run in real-time (in about 800 μs), and which could handle more stringent timing requirements.

   Significant improvement in performance could be obtained by using an integrated circuit to perform the "DQ-MF" and "MF-DQ" transformations. This would free up the PC from these time critical conversions, and would enable second stage filtration in hardware itself. An application specific integrated circuit (ASIC) could be used to realize this circuit in hardware.

2. *ANN based sensorless induction motor drive*: Based on this work, combining the ANN control scheme with the ANN speed estimator would result in a complete ANN based sensorless drive. In this scheme, the control ANN could be trained as before. However, instead of the actual speed, it would receive the speed estimate

from the ANN speed estimator, which would have to be off-line trained to a high degree of accuracy. Of course, restricting to only off-line training would imply foregoing robustness in the control scheme. This scheme would be a lot more challenging than just putting these individual components together, because any inaccuracy in the speed estimate would propagate back into the control scheme, acting like a positive feedback, and further worsening the speed estimate.

As a first step towards obtaining the sensorless drive, a very high performance off-line trained ANN speed estimator should be developed. The vector-controller should be able to function properly with feedback from the ANN speed estimator. The training data for the ANN controller should be collected while running the drive with speed feedback obtained from the ANN speed estimator instead of the shaft encoder. This data should be used for training the control ANN, and after successful training, this ANN should be able to function in real-time, resulting in an ANN based sensorless induction motor drive.

3. *ANN based indirect adaptive control of induction motor*: In this scheme two ANNs would be required, one for plant identification and the other one for control. The ANN speed estimator would be the plant identifier in this case, and would be on-line trained with the help of the actual speed obtained from a speed sensor. The control ANN would also be on-line trained, and its cost function would be derived from the error between the reference speed and the actual speed. The computation of the Jacobian, which is required for adjusting the weights on-line using backpropagation, would be done through the speed estimator ANN. This scheme looks promising, and it should be possible to get performance comparable to the schemes mentioned in this work.

References

[1] P. Vas. *Vector Control of AC Machines.* Clarendon Press - Oxford, 1990.

[2] F. Harashima. "Power Electronics and Motion Control — A Future Perspective". *Proceedings of the IEEE*, vol. 82, no. 8, pp. 1107–1111, Aug 1994.

[3] S. uk Kim, I. woo Yang, Y. jo Kim, and Y. seok Kim. "Robust TDOF Controller of Induction Motor Without Speed Sensors for Variations of Rotor Resistance". In *Power Electronics Specialists Conf.*, vol. 2, pp. 1043–1049, 1997.

[4] C. Won, D. Kim, S. Kim, and D. Yoo. "Position Control of Induction Motor With a New Fuzzy-Sliding Mode Controller". In *Proceedings of Power Conversion Conf. - Yokohama*, pp. 421–427, 1993.

[5] General Motors Corporation. "EV1 Electric Vehicle Specifications". In *www site: http://www.gmev.com/specs/specs.htm*, 1998.

[6] N. B. Karayiannis and A. N. Venetsanopoulos. *Artificial Neural Networks — Learning Algorithms, Performance Evaluation, and Applications.* Kluwer Academic Publishers, 1993.

[7] M. H. Hassoun. *Fundamentals of Artificial Neural Networks.* The MIT Press, Cambridge, Massachusetts, 1995.

[8] S. Haykin. *Neural Networks — A Comprehensive Foundation.* Macmillan College Publishing Company, Inc., 1994.

[9] C.-T. Lin and G. Lee. *Neural Fuzzy Systems — A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentice Hall PTR, Upper Saddle River, NJ 07458, 1996.

[10] B. K. Bose. "Power Electronics and Motion Control — Technology Status and Recent Trends". *IEEE Trans. Ind. Applicat.*, vol. 29, no. 5, pp. 902–909, Sep/Oct 1993.

[11] K. Koga, R. Ueda, and T. Sonoda. "Constitution of *V/f* Control for Reducing the Steady-State Speed Error to Zero in Induction Motor Drive System". *IEEE Trans. Ind. Applicat.*, vol. 28, no. 2, pp. 463–471, Mar/Apr 1992.

[12] P. Famouri and J. J. Cathey. "Loss Minimization Control of an Induction Motor Drive". *IEEE Trans. Ind. Applicat.*, vol. 27, no. 1, pp. 32–37, Jan/Feb 1991.

[13] M. Iwasaki and N. Matsui. "Robust Speed Control of IM with Torque Feedforward Control". *IEEE Trans. Ind. Elect.*, vol. 40, no. 6, pp. 553–560, Dec 1993.

[14] F. Alonge. "MRAC and Sliding Motion Control Techniques to Design a new Robust Controller for Induction Motor Drives.". In *Proceedings of Power Conversion Conf. - Yokohama*, pp. 290–296, 1993.

[15] P. T. Krein, F. Disilvestro, I. Kanellakopoulos, and J. Locker. "Comparative Analysis of Scalar and Vector Control Methods for Induction Motors". In *Power Electronics Specialists Conf.*, pp. 1139–1145, 1993.

[16] F. Blaschke. "The Principle of Field Orientation as Applied to the new TRANSVECTOR Closed Loop Control System for Rotating-Field Machines". *Siemens Review*, vol. 34, pp. 217–220, May 1972.

[17] R. D. Lorenz, T. A. Lipo, and D. W. Novotny. "Motion Control with Induction Motors". *Proceedings of the IEEE*, vol. 82, no. 8, pp. 1215–1240, Aug 1994.

[18] D. Fodor, Z. Katona, and E. Szesztay. "Digitized Vector Control of Induction Motor with DSP". In *Proceedings of the IEEE IECON*, vol. 3, pp. 2057–2062, 1994.

[19] L. Zhen and L. Xu. "A Mutual MRAS Identification Scheme for Position Sensorless Field Oriented Control of Induction Machines". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 1, pp. 159–165, 1995.

[20] K. Rajashekara, A. Kawamura, and K. Matsuse. *Sensorless Control of AC Motor Drives: Speed and Position Sensorless Operation.* IEEE Press, 1996.

[21] T. Ohtani, N. Takada, and K. Tanaka. "Vector Control of Induction Motor without Shaft Encoder". *IEEE Trans. Ind. Applicat.*, vol. 28, no. 1, pp. 157–164, Jan/Feb 1992.

[22] T.-H. Chin. "Approaches for Vector Control of Induction Motor without Speed Sensor". In *Proceedings of the IEEE IECON*, vol. 3, pp. 1616–1620, 1994.

[23] C. Ilas, A. Bettini, L. Ferraris, G. Griva, and F. Profumo. "Comparison of Different Schemes without Shaft Sensors for Field Oriented Control Drives". In *Proceedings of the IEEE IECON*, vol. 3, pp. 1579–1588, 1994.

[24] T. Kanmachi and I. Takahashi. "Sensor-Less Speed Control of an Induction Motor". *IEEE Ind. Applicat. Magazine*, pp. 22–27, Jan/Feb 1995.

[25] K. Ohnishi, N. Matsui, and Y. Hori. "Estimation, Identification, and Sensorless Control in Motion Control System". *Proceedings of the IEEE*, vol. 82, no. 8, pp. 1253–1265, Aug 1994.

[26] G. C. Verghese and S. R. Sanders. "Observers for Flux Estimation in Induction Machines". *IEEE Trans. Ind. Elect.*, vol. 35, no. 1, pp. 85–94, Feb 1988.

[27] H. Kubota, K. Matsuse, and T. Nakano. "DSP-Based Speed Adaptive Flux Observer of Induction Motor". *IEEE Trans. Ind. Applicat.*, vol. 29, no. 2, pp. 344–348, Mar/Apr 1993.

[28] C. Schauder. "Adaptive Speed Identification for Vector Control of Induction Motors without Rotational Transducers". *IEEE Trans. Ind. Applicat.*, vol. 28, no. 5, pp. 1054–1061, Sep/Oct 1992.

[29] F.-Z. Peng and T. Fukao. "Robust Speed Identification for Speed Sensorless Vector Control of Induction Motors". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 1, pp. 419–426, 1993.

[30] R. T. Stefani, Clement J. Savant, Jr., B. Shahian, and G. H. Hostetter. *Design of Feedback Control Systems.* Saunders College Publishing, 1994.

[31] Y.-R. Kim, S.-K. Sul, and M.-H. Park. "Speed Sensorless Vector Control of an Induction Motor Using an Extended Kalman Filter". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 1, pp. 594–599, 1992.

[32] K. Hurst, T. Habetler, G. Griva, and F. Profumo. "Speed Sensorless Field-Oriented Control of Induction Machines Using Current Harmonic Spectral Estimation". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 1, pp. 601–607, 1994.

[33] H. Kubota and K. Matsuse. "Simultaneous Estimation of Speed and Rotor Resistance of Field Oriented Induction Motor without Rotational Transducers". In *Proceedings of Power Conversion Conf. – Yokohama*, pp. 473–477, 1993.

[34] K. Gopakumar, V. Ranganathan, and S. Bhat. "Vector Control of Induction Motor With Split Phase Stator Windings". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 1, pp. 569–574, 1994.

[35] U. Baader, M. Depenbrock, and G. Gierse. "Direct Self Control (DSC) of Inverter-Fed Induction Machine: A Basis for Speed Control Without Speed Measurement". *IEEE Trans. Ind. Applicat.*, vol. 28, no. 3, pp. 581–588, May/Jun 1992.

[36] Cybenko. "Approximations by Superpositions of a Sigmoidal Function". *Mathematics of Contr., Signals and Syst.*, vol. 2, pp. 303–314, 1989.

[37] A. K. Toh, E. P. Nowicki, and F. Ashrafzadeh. "A Flux Estimator for Field Oriented Control of an Induction Motor using an Artificial Neural Network". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 1, pp. 585–592, 1994.

[38] M. Mohamadian, E. Nowicki, and J. Salmon. "A Neural Network Controller for Indirect Field Orientation Control". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 2, pp. 1770–1774, 1995.

[39] M. G. Simões and B. K. Bose. "Neural Network Based Estimation of Feedback Signals for a Vector Controlled Induction Motor Drive". *IEEE Trans. Ind. Applicat.*, vol. 31, no. 3, pp. 620–629, May/Jun 1995.

[40] P. Marino, M. Milano, and F. Vasca. "Robust Neural Network Observer for Induction Motor Control". In *Power Electronics Specialists Conf.*, vol. 1, pp. 699–705, 1997.

[41] A. Ba-Razzouk, A. Ch'eriti, G. Olivier, and P. Sicard. "Field-Oriented Control of Induction Motors Using Neural-Network Decouplers". *IEEE Trans. Power Electronics*, vol. 12, no. 4, pp. 752–763, Jul 1997.

[42] L. Ben-Brahim. "Motor Speed Identification via Neural Networks". *IEEE Ind. Applicat. Magazine*, pp. 28–32, Jan/Feb 1995.

[43] P. Mehrotra, J. E. Quaicoe, and R. Venkatesan. "Speed Estimation of Induction Motor Using Artificial Neural Networks". In *Proceedings of the IEEE IECON*, vol. 2, pp. 881–886, Aug 5-10, 1996.

[44] B. Burton, F. Kamran, R. G. Harley, T. G. Habetler, M. Brooke, and R. Poddar. "Identification and Control of Induction Motor Stator Currents Using Fast On-Line Random Training of a Neural Network". In *Conf. Rec. IEEE Ind. Applicat. Soc. Ann. Meeting*, vol. 2, pp. 1781–1787, 1995.

[45] M. T. Wishart and R. G. Harley. "Identification and Control of Induction Machines using Neural Networks". *IEEE Trans. Ind. Applicat.*, vol. 31, no. 3, pp. 612–619, May/Jun 1995.

[46] L. A. Cabrera, M. E. Elbuluk, and D. S. Zinger. "Learning Techniques to Train Neural Networks as a State Selector for Inverter-Fed Induction Machines Using Direct Torque Control". In *Power Electronics Specialists Conf.*, vol. 1, pp. 233–242, 1994.

[47] L. A. Cabrera, M. E. Elbuluk, and I. Husain. "Tuning the Stator Resistance of Induction Motors Using Artificial Neural Network". *IEEE Trans. Power Electronics*, vol. 12, no. 5, pp. 779–787, Sep 1997.

[48] Y. Kung, C. Liaw, and M. Ouyang. "Adaptive Speed Control for Induction Motor Drives Using Neural Networks". *IEEE Trans. Ind. Elect.*, vol. 42, no. 1, pp. 25–32, Feb 1995.

[49] S. Tadakuma, S. Tanaka, H. Naitoh, and K. Shimane. "Improvement of Robustness of Vector Controlled Induction Motors Using Feedforward and Feedback Control". *IEEE Trans. Power Electronics*, vol. 12, no. 2, pp. 221–227, Mar 1997.

[50] K. S. Narendra and K. Parthasarthy. "Identification and Control of Dynamical Systems using Neural Networks". *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, Mar 1990.

[51] K.-K. Shyu, H.-J. Shieh, and S.-S. Fu. "Model Reference Adaptive Speed Control for Induction Motor Drive Using Neural Networks". *IEEE Trans. Ind. Elect.*, vol. 45, no. 1, pp. 180–182, Feb 1998.

[52] Howard Demuth and Mark Beale. *Neural Network Toolbox: For Use with MATLAB, version 3.* The MathWorks Inc., Natick, MA 01760-1500, 1998.

[53] *SIMULINK: Dynamic System Simulation for MATLAB, version 2*, 1997.

[54] *Solutions for Control — dSPACE Catalogue*, 1997.

[55] EMTP Development Coordination Group. *Electromagnetic Transients Program (EMTP) Revised Rule Book, Version 2.0*, 1989.

[56] P. Mehrotra, R. Venkatesan, and J. E. Quaicoe. "Development of a Flexible Object-Oriented Artificial Neural Network Simulator". In *Canadian Conf. on Electrical and Computer Engineering*, vol. 1, pp. 318–321, May 25-28, 1997.

[57] B. Eckel. *C++ Inside & Out.* Osborne McGraw-Hill, 1993.

[58] R. H. Nielsen. "Theory of Backpropogation Neural Network". In *International Joint Conf. on Neural Networks*, pp. I585–I592, 1989.

[59] P. Mehrotra, J. E. Quaicoe, and R. Venkatesan. "Induction Motor Speed Estimation Using Artificial Neural Networks". In *Canadian Conf. on Electrical and Computer Engineering*, vol. 2, pp. 607–610, May 26-29, 1996.

[60] P. Mehrotra, J. E. Quaicoe, and R. Venkatesan. "Development of an Artificial Neural Network Based Induction Motor Speed Estimator". In *Power Electronics Specialists Conf.*, vol. 1, pp. 682–688, June 24-27, 1996.

[61] K. S. Narendra. "Neural Networks for Control: Theory and Practice". *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1385–1406, Oct 1996.

[62] J. Tanomaru and S. Omatu. "Process Control by On-Line Trained Neural Network Controllers". *IEEE Trans. Ind. Elect.*, vol. 39, no. 6, pp. 511–521, Dec 1992.

[63] P. Mehrotra, R. Venkatesan, and J. E. Quaicoe. "Induction Motor Control Using Artificial Neural Networks With Vector Controller Mimicing". *paper submitted to the IEEE Trans. Ind. Elect.*, 1999.

[64] P. Mehrotra, J. E. Quaicoe, and R. Venkatesan. "A Real-Time Offline Trained High Performance Artificial Neural Network Induction Motor Speed Estimator". *paper submitted to the IEEE Trans. Power Elect.*, 1999.

[65] P. Ienne, T. Cornu, and G. Kuhn. "Special-Purpose Digital Hardware for Neural Networks: An Architectural Survey". *Journal of VLSI Signal Processing Systems*, vol. 13, no. 1, pp. 5–25, 1996.

[66] A. König. "Survey and Current Status of Neural Network Hardware". In *Proceedings of the International Conference on Artificial Neural Networks*, pp. 391–410, 1995.

[67] C. S. Lindsey and T. Lindblad. "Review of Hardware Neural Networks". In *www site: http://msia02.msi.se/ ~lindsey/elba2html/elbaFramed.html*, Last modified: June, 1998.

[68] Analog Devices. *RTI-800/815 User's Manual*, 1987.

[69] L. Geppert. "High-flying DSP Architectures". *IEEE Spectrum*, vol. 35, no. 11, pp. 53–56, Nov 1998.

Appendix A

Simulation Input Files

This appendix contains sample parameter files, which serve as inputs to the various components of the simulation. All blank lines in the parameter files, as well as those that begin with a '%' sign are ignored. There must be an '=' sign (without any spaces), between the parameter name and its value (e.g. P=2). The parameters can be in any order within a parameter file, as long as they are identified with the correct parameter name.

A.1 Induction machine parameters (*mach.par*)

```
% 2-kW, Vnom = 120/208 V - 60 Hz., Inom = 15.2/8.8 A,
% rated speed = 1770rpm, Tem,nom = 10.8 Nm
% Number of pole pairs
P=2
% Connection type (WYE or DELTA)
```

```
YD=WYE
% Stator resistance
Rs=0.60
% Rotor resistance
Rr=0.40
% Stator inductance
Ls=0.0727
% Rotor inductance
Lr=0.0727
% Magnetizing inductance
Lm=0.0698
% Damping coefficient
B=0.0030
% Moment of inertia
J=0.0357
```

A.2 Vector controller parameters (*vector.par*)

```
% Induction motor parameters needed in the vector controller
Rr=0.40
Lr=0.0727
Lm=0.0698
P=2
% Speed controller PI constants
Kp_sc=2.0
Ki_sc=8.0
```

```
% Torque controller PI constants
Kp_tc=0.3
Ki_tc=15.0
% Flux controller PI constants
Kp_fc=5.0
Ki_fc=25.0
% Ramp rate for speed reference
RAMP_RATE=400
% Current limit is used to compute the limit for the torque controller
CURRENT_LIMIT=20.0
% imr reference (proportional to the rotor flux)
IMR_SETTING=6.0
```

A.3 Inverter parameters (*inv.par*)

```
% Transistor current rating
CURRENT_RATING=150
% Transistor voltage rating
VOLTAGE_RATING=600
```

A.4 PWM current controller parameters (*pwm.par*)

```
% Type of carrier signal (TRIANGLE, SINE, SQUARE or SAWTOOTH)
CARRIER_TYPE=TRIANGLE
% Amplitude of the carrier signal
```

```
CARRIER_AMPL=2.0
% Frequency of the carrier signal
CARRIER_FREQ=4000
% DC offset for the carrier signal
DC_OFFSET=0.0
```

A.5 Backpropagation learning parameters (*bpn.par*)

The information about the number of layers and number of neurons in each layer is ignored by the backpropagation program. This information is used only by the *layerann* program, which generates the ANN parameter file.

```
% Number of layers
3
% Number of neurons in each layer, their types, learning rates
% and optional parameters
6
20 TANSIG LR=0.20
 1 TANSIG LR=0.05
% The information below is used by the backpropagation program
% Scale factors for ANN inputs
I0=23.0
I1=0.036
I2=23.0
I3=0.036
I4=84.0
```

```
I5=84.0
% Scale factors for ANN outputs
O0=178.0
% Momentum parameter
MOMENTUM=0.5
% Weight storage file
WTFILE=speedest.net
% Saving frequency
SAVE_FREQ=1000
% Training frequency
TRAIN_FREQ=1
% Network mode (LOAD_NET: load trained network from file,
% NEW_NET = create new network)
NET_MODE=NEW_NET
% TRAIN (train the network) OR COMPUTE (feedforward mode)
RUN_MODE=TRAIN
```

A.6 ANN architectural parameters (*ann.par*)

This file has been generated by running the *layerann* program, which reads the first few lines on the *bpn.par* parameter file and creates the *ann.par* file shown below. This file contains all the details about the ANN architecture, and can be manually reconfigured to add or remove connections, change the neuron types and so on.

```
% Part 1 of parameter file
% Number of neurons
```

```
21
% Number of ann inputs
6
% Number of ann outputs
1

% Part 2 of parameter file (neuron description)
0 TANSIG 6 LR=0.20
1 TANSIG 6 LR=0.20
2 TANSIG 6 LR=0.20
3 TANSIG 6 LR=0.20
4 TANSIG 6 LR=0.20
5 TANSIG 6 LR=0.20
6 TANSIG 6 LR=0.20
7 TANSIG 6 LR=0.20
8 TANSIG 6 LR=0.20
9 TANSIG 6 LR=0.20
10 TANSIG 6 LR=0.20
11 TANSIG 6 LR=0.20
12 TANSIG 6 LR=0.20
13 TANSIG 6 LR=0.20
14 TANSIG 6 LR=0.20
15 TANSIG 6 LR=0.20
16 TANSIG 6 LR=0.20
17 TANSIG 6 LR=0.20
18 TANSIG 6 LR=0.20
19 TANSIG 6 LR=0.20
20 TANSIG 20 LR=0.05
```

```
% Part 3 of parameter file (connection description)
INPUT 0 0
INPUT 1 0
INPUT 2 0
INPUT 3 0
INPUT 4 0
INPUT 5 0
INPUT 0 1
INPUT 1 1
INPUT 2 1
INPUT 3 1
INPUT 4 1
INPUT 5 1
INPUT 0 2
INPUT 1 2
INPUT 2 2
INPUT 3 2
INPUT 4 2
INPUT 5 2
INPUT 0 3
INPUT 1 3
INPUT 2 3
INPUT 3 3
INPUT 4 3
INPUT 5 3
INPUT 0 4
INPUT 1 4
```

```
INPUT 2 4
INPUT 3 4
INPUT 4 4
INPUT 5 4
INPUT 0 5
INPUT 1 5
INPUT 2 5
INPUT 3 5
INPUT 4 5
INPUT 5 5
INPUT 0 6
INPUT 1 6
INPUT 2 6
INPUT 3 6
INPUT 4 6
INPUT 5 6
INPUT 0 7
INPUT 1 7
INPUT 2 7
INPUT 3 7
INPUT 4 7
INPUT 5 7
INPUT 0 8
INPUT 1 8
INPUT 2 8
INPUT 3 8
INPUT 4 8
INPUT 5 8
```

```
INPUT 0 9
INPUT 1 9
INPUT 2 9
INPUT 3 9
INPUT 4 9
INPUT 5 9
INPUT 0 10
INPUT 1 10
INPUT 2 10
INPUT 3 10
INPUT 4 10
INPUT 5 10
INPUT 0 11
INPUT 1 11
INPUT 2 11
INPUT 3 11
INPUT 4 11
INPUT 5 11
INPUT 0 12
INPUT 1 12
INPUT 2 12
INPUT 3 12
INPUT 4 12
INPUT 5 12
INPUT 0 13
INPUT 1 13
INPUT 2 13
INPUT 3 13
```

```
INPUT 4 13
INPUT 5 13
INPUT 0 14
INPUT 1 14
INPUT 2 14
INPUT 3 14
INPUT 4 14
INPUT 5 14
INPUT 0 15
INPUT 1 15
INPUT 2 15
INPUT 3 15
INPUT 4 15
INPUT 5 15
INPUT 0 16
INPUT 1 16
INPUT 2 16
INPUT 3 16
INPUT 4 16
INPUT 5 16
INPUT 0 17
INPUT 1 17
INPUT 2 17
INPUT 3 17
INPUT 4 17
INPUT 5 17
INPUT 0 18
INPUT 1 18
```

```
INPUT 2 18
INPUT 3 18
INPUT 4 18
INPUT 5 18
INPUT 0 19
INPUT 1 19
INPUT 2 19
INPUT 3 19
INPUT 4 19
INPUT 5 19
HIDDEN 0 20
HIDDEN 1 20
HIDDEN 2 20
HIDDEN 3 20
HIDDEN 4 20
HIDDEN 5 20
HIDDEN 6 20
HIDDEN 7 20
HIDDEN 8 20
HIDDEN 9 20
HIDDEN 10 20
HIDDEN 11 20
HIDDEN 12 20
HIDDEN 13 20
HIDDEN 14 20
HIDDEN 15 20
HIDDEN 16 20
HIDDEN 17 20
```

```
HIDDEN 18 20
HIDDEN 19 20
OUTPUT 20 0
```