NEURAL NETWORK BASED INCIPIENT FAULT DETECTION OF INDUCTION MOTORS

MOHD. ROKONUZZAMAN

Neural Network Based Incipient Fault Detection of Induction Motors

By
©Mohd. Rokonuzzaman, B.Sc. Eng.

A THESIS SUMBMITTED TO THE SCHOOL OF GRADUATE STUDIES IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING.

FACULTY OF ENGINEERING AND APPLIED SCIENCE.
MEMORIAL UNIVERSITY OF NEWFOUNDLAND.
MARCH, 1995

ST. JOHN'S NEWFOUNDLAND CANADA.

ISBN 0-612-01914-4

Abstract

An incipient fault detection scheme of induction motors through the recognition of frequency spectra of the stator current has been developed in this thesis. It is based on the adaptive resonance theory of neural networks. This fault diagnosis scheme is not only capable of detecting a fault but also can report if it cannot identify a particular fault so that necessary preventive steps can be taken to update the underlying neural network to adapt to this undetected fault. Moreover, it can update itself to cope with this dynamic situation retaining already acquired knowledge without the need of retraining with the old patterns.

A laboratory experimental set-up using a digital signal processing(DSP) technique has been employed to collect the frequency spectra of the stator current at different fault conditions. A wound-rotor induction motor has been used as the test motor to create different types of faults making unbalance in the stator and rotor circuits. A 24-bit high speed DSP board has been used with a personal computer to develop a real-time interactive software to collect the spectra. A driver for the HP-plotter has also been developed to directly plot the frequency spectra of the stator current .

Adaptive resonance theory(ART) based network is a recent addition to the neural network family. A new software has been successfully developed and implemented in the laboratory experiment using ART neural network. Its performances in training, recalling and dynamic updating have been studied with a set of example patterns. The incipient faults of a 3-phase wound rotor induction motor have been successfully diagonized by this neural network.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Professor M.A. Rahman for his important contribution to this work. It is through his patience, understanding and advice that this work has been done.

Special thanks are due to technical staff at the faculty of Engineering and Applied Science for their assistance in the experimental part of the thesis. I would like to extend my thanks to other researchers working in the Power Research Laboratory at the Memorial University of Newfoundland for their kind co-operation.

I am indebted to the Government of Canada as well as the Government of Bangladesh to provide financial and other technical assistances to carry out my graduate study. I wish to thank the officials of the Canadian International Development Agency(CIDA), Memorial University of Newfoundland(MUN) and Bangladesh Institute of Technology(BIT), Rajshahi for their respective assistances. Particular thanks to Professor M.A. Rahman, Director, CIDA/MUN/BIT project.

Finally, I owe it to my dear family for their patience, encouragement and blessings when I am thousands of miles away from them.

Contents

1 Introduction **1**

1.1 Incipient Fault Detection of Induction Motors 2

1.1.1 A Brief Description 2

1.1.2 Mathematical Analysis of Induction Motor 4

1.2 Artificial Neural Network for Fault Detection 7

1.2.1 Expert Approach for Fault Detection 7

1.2.2 Learning Skills of Artificial Neural Networks 8

1.3 Literature Review 8

1.4 Objective of the Present Work 12

1.5 Overview of Thesis 12

2 An overview of Artificial Neural Networks **14**

2.1 Introduction 14

2.1.1 Models of a Neuron 15

2.1.2 Network Architecture 17

2.1.3 Artificial Intelligence and Neural Networks 18

2.2 Backpropagation 19

2.3 The Binary Associative Memory(BAM) and the Hopfield Memory 21
2.3.1 The BAM 22
2.3.2 The Hopfield Memory 24
2.4 Simulated Annealing 25
2.4.1 The Boltzman Machine 25
2.5 The Counter Propagation Network 28
2.5.1 CPN Building Blocks 29
2.5.2 Training the CPN 32
2.5.3 Forward Mapping 33
2.6 Self-Organizing Maps 34
2.6.1 Unit Activations 35
2.6.2 The SOM Learning Algorithm 35
2.7 Spatiotemporal Pattern Classification 35
2.7.1 The Formal Avalanche 36
2.7.2 Architectures of Spatiotemporal Networks(STNS) . . . 36
2.8 The Neocognitron 37
2.8.1 Neocognitron Architecture 37
2.8.2 Neocognition Data Processing 38
2.9 Adaptive Resonance Theory(ART) 39
2.9.1 ART Network Description 39
2.9.2 ART1 41
2.9.3 ART2 43
2.10 Comparative analysis and selection of suitable Network 43

3 ART2 Neural Network **45**

3.1 Introduction 45

3.2 ART2 Architecture 45

3.2.1 The Attentional Subsystem 46

3.2.2 The Orienting Subsystem 49

3.2.3 Gain Control in ART2 50

3.2.4 Least-mean-square Equations 51

3.2.5 Bottom-Up Least-mean-square Initialization 52

3.2.6 ART2 Processing Summary 52

3.3 ART2 Simulator 55

3.3.1 Model of ART2 as an Object 55

3.3.2 Modified Structure of Training and Recalling Pattern in the Network 55

3.3.3 Dynamic Updating 56

3.4 Experimental Varification of Performance 56

3.4.1 Training of the Neural Network with test Patttern . . . 58

3.4.2 Dynamic Neuron Addition 60

3.4.3 Pattern Recall from the Network 61

4 Fault Related Information Collection **63**

4.1 Introduction 63

4.2 Model of Spectra Collection 63

4.3 Discrete Time Signals and Systems 64

4.3.1 Discrete-Time Signals: Sequences 65

4.3.2 Discrete-Time Systems 65

4.3.3 Sampling of Continuous-Time Signals 66
4.3.4 The Discrete Fourier Transform(DFT) 67
4.3.5 Computation of the Discrete Fourier Transform 67
4.3.6 Fourier Analysis of Signals Using the Discrete Fourier Transform 68
4.4 Motorola's DSP56000 DSP Family 70
4.5 Ariel's Interface and DSP Library 74
4.5.1 Interface Library 74
4.5.2 DSP-Library 75
4.6 Application Software for Spectra Collection 76
4.7 Experimental Setup 76
4.8 Spectra for Different Fault Conditions 78
4.8.1 No Fault condition: 80
4.8.2 Stator phase 1 is open: 81
4.8.3 Stator phase 2 is open: 82
4.8.4 Stator phase 3 is open: 82
4.8.5 Short circuit fault through resistance in Stator Phase 1: 83
4.8.6 Short circuit fault through resistance in Stator Phase 2: 84
4.8.7 Short circuit fault through resistance in Stator Phase 3: 85
4.8.8 Short circuit fault in Stator Phase 1: 86
4.8.9 Short circuit fault in Stator Phase 2: 87
4.8.10 Short circuit fault in Stator Phase 3: 88
4.8.11 Rotor open circuit fault in phase 1: 89
4.8.12 Rotor open circuit fault in phase 2: 90
4.8.13 Rotor open circuit fault in phase 3: 91

4.8.14 Rotor phase 1 is shorted to neural: 92
4.8.15 Rotor phase 2 is shorted to neural: 93
4.8.16 Rotor phase 3 is shorted to neural: 94
4.8.17 Rotor short circuit fault through resistance in phase 1 : 95
4.8.18 Rotor short circuit fault through resistance in phase 2 : 96
4.8.19 Rotor short circuit fault through resistance in phase 3 : 97
4.8.20 Rotor phase 1 is unbalanced through an external resistor: 98
4.8.21 Rotor phase 2 is unbalanced through an external resistor: 99
4.8.22 Rotor phase 3 is unbalanced through an external resistor:100
4.8.23 Simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 1: 101
4.8.24 Simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 2: 102
4.8.25 Simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 3: 103
4.8.26 Remarks on Fault Related Frequenency spectra of the Stator Current 103

5 Fault Recognition by ART2 Neural Network 104

5.1 Introduction 104
5.2 Training Data Set 105
5.2.1 Data Reduction: 106
5.3 Structure of the Network 108
5.4 Training of the Network 109
5.5 Fault Recognition by the trained network. 111

5.6 Model of ART2 neural network based Incipient Fault Detection System 112

6 Conclusions and Recommendations for Future Work 116

6.1 Conclusions 116

6.2 Recommendations for Future Work 118

Appendix-A **Program listing for ART2 Neural Network.** 124
Appendix-B **Program listing for real-time frequency spectra acquisition.** 141
Appendix-C **Program listing of HP-plotter driver.** 150

List of Figures

1.1 (a) Frequency spectra of input current of a healthy machine.(b) Frequency spectra of input current of a faulty machine. (c). Model of Neural Network based fault related spectra identification system. 9

2.1 Model of a neuron 15

2.2 Activation Functions: (a) Threshold Function. (b) Piecewise-Linear Function (c) Sigmoidal Function 18

2.3 The three layer BPN architecture 20

2.4 BAM architecture 22

2.5 Discrete Hopfield Memory 25

2.6 A simple energy landscape with two minimal, a local minimum and a global minimum 26

2.7 The Botzman Completion Architecture 26

2.8 Forward-mapping CPN 29

2.9 Layer of input units of a CPN 30

2.10 This figure shows (a) the general form of the processing elements (b) the instar form of processing elements 31

2.11 A layer of instars arranged in c CPN 32

2.12 Outstar and its relationship to the CPN architecture. (a). Outstar structures in CPN network (b). A single outstar unit is shown 33

2.13 Grossber's formal avalanche structure 36

2.14 Power Spectra generated from speech 37

2.15 Model of ART system 40

2.16 A pattern matching cycle in an ART. (a) pattern-matching attempt (b) reset in (c) final recognition (d) end of matching cycle 41

3.1 The overall structure of ART2 46

3.2 Structure of processing element on F_1 layer 47

3.3 Structure of processing element on F_2 layer 50

3.4 Model of ART2 as an object 56

3.5 Flow chart of training algorithm of ART2 57

3.6 Modified Training and Recall Algorithm of ART2 58

4.1 Model of Fault Related Spectra Collection 64

4.2 Representation of discrete-time system 65

4.3 Processing steps in the discrete-time Fourier analysis of a continuous-time signal 68

4.4 Illustration of the Fourier transforms in the system : (a) Fourier transform of continuous-time input signal. (b) Frequency response of anti aliasing filter. (c) Fourier transform output of anti aliasing filter. (d) Fourier transform of sampled signal (e) Fourier transform of window sequence (f) Fourier transform of windowed signal segment and frequency samples obtained using DFT samples 69
4.5 DSP56000 Block Diagram 70
4.6 PC-56 Block Diagram 74
4.7 Flowchart of the application program 77
4.8 Block diagram of the experimental setup. 78
4.9 Frequency spectra at no fault condition 80
4.10 Frequency spectra when stator phase 1 is open 81
4.11 Frequency spectra when stator phase 2 is open 82
4.12 Frequency spectra when stator one coil of phase 1 has been externally replaced by a resistor 83
4.13 Frequency spectra when stator one coil of phase 2 has been externally replaced by a resistor 84
4.14 Frequency spectra when stator one coil of phase 3 has been externally replaced by a resistor 85
4.15 Frequency spectra when stator one coil of phase 1 was short . 86

4.16 Frequency spectra when stator one coil of phase 2 was short . 87
4.17 Frequency spectra when stator one coil of phase 3 was short . 88
4.18 Frequency spectra of the stator current when rotor one coil M1 was open circuit 89
4.19 Frequency spectra of the stator current when rotor one coil M2 was open circui 90
4.20 Frequency spectra of the stator current when rotor one coil M3 was open circui 91
4.21 Frequency spectra of the stator current when rotor phase M1 is shorted to neutral 92
4.22 Frequency spectra of the stator current when rotor phase M2 is shorted to neutral 93
4.23 Frequency spectra of the stator current when rotor phase M3 is shorted to neutral 94
4.24 Frequency spectra of the stator current at rotor short circuit fault through resistance in phase M1 95
4.25 Frequency spectra of the stator current at rotor short circuit fault through resistance in phase M2 96
4.26 Frequency spectra of the stator current at rotor short circuit fault through resistance in phase M3 97
4.27 Rotor phase 1 is unbalanched through an external resistance . 98
4.28 Rotor phase 2 is unbalanched through an external resistance . 99
4.29 Rotor phase 3 is unbalanched through an external resistance . 100
4.30 Simultaneous open circuit fault in rotor phase 2 as well as stator phase 1 101

4.31 Simultaneous open circuit fault in rotor phase 2 as well as stator phase2 102
5.1 Model of ART2 neural network based on-line incipient fault diagnosis system for induction motors. 115

List of Tables

2.1 Comparative Performance of Different ANN Techniques 44

3.1 Values of parameters on F-1 layer 48
3.2 Set of training examples used to train ART2 59
3.3 Bottom-up weight matrix after training 59
3.4 Top-down weight matrix after training 59
3.5 New training vector to train after neuron addition 60
3.6 New Bottom-up weight matrix after training 60
3.7 New Top-down weight matrix after training 61
3.8 Pattern matching result of the trained network 62

5.1 Table of Faults with unique number. 105
5.2 Fault related spectral components in matrix form 106
5.3 Training Matrix of fault related current spectra. 107
5.4 Faults mapping of the trained network in high precision domain 108
5.5 Top-down weight matrix of the trained network. 110
5.6 Bottom-up weight matrix of the trained network. 110
5.7 Fault related noisy current spectra. 113
5.8 Diagnostic test result of the trained network in noisy situation. 113
5.9 Faults mapping in training phase in low precision domain. . . 114
5.10 Fault diagnostic performance of the network trained in low precision domain in noisy situation 114

Abbreviations and Symbols:

The following lists of terms and symbols appear throughout the body of this document. They are defined here approximately in the order in which they appear in the text:

Abbreviations:

Terms	Definition
ANN	Artificial neural network
FNN	Feedforward neural network
DSP	Digital signal processing
BPN	Backpropagation neural network
BAM	Binary associative memory
HM	Hpfield memory
VLSI	Very large scale integration
CPN	Counter propagation network
SOM	Self-organizing maps
STNS	Spatiotemporal networks
ART	Adaptive resonance theory
STM	Short term memory
LTM	Long term memory
DFT	Discrete Fourier transform
FFT	Fast Fourier transform
RAM	Random access memory
ROM	Read only memory
ALU	Arithmetic logic unit
AGU	Address generation unit
MCU	Microcontroller unit
MPU	Microprocessor unit
MIP	Million instructions per second
IIR	Infinite impulse response

Abbreviations:

Terms	Definition
SCI	Serial communication interface
SSI	Synchronous serial interface
PC	Personal computer
ARIELMON	Ariel monitor
DEGMON	Debugg monitor
IBM	Internation Buisiness Machine
HP	Hewlet & Packard
AS	Attentional subsystem
PE	Processing element

Symbols:

Symbols	Definition
i_s	Stator current
ϕ_r	Rotor flux
v_s	Stator voltage per phase
R_s	Stator resistance per phase
R_r	Rotor resistance per phase
L_s	Stator self inductance per phase
L_r	Rotor self inductance per phase
M	Mutual inductance
δ	Leakage coefficient
ω_r	Angular velocity
u_k	Linear combiner output
w_{kj}	Synaptic weight of synapse j belonging to neuron k

Symbols:

Symbols	Definition
x_j	Input signal j.
y_k	the output signal of neuron k
φ	The activation function
θ_k	The threshold
v_k	The activity level
$\mathbf{x}$	Input vector
net_{pj}^{h}	Net input values to the hidden layer units
O_{pk}	Calculated output
E_p	Measure of learning
w	Weight vector
$\mathbf{E}$	System energy
ΔE_k	The energy difference
Θ_i	The reflectance pattern
$x[n]$	Discrete input sequence
$T.$	Discrete operator
$y[n]$	Discrete output sequence
$S_c(t)$	Band-unlimited continuous-time signal
$x_c(t)$	Bandlimited continuous-time signal
$X[k]$	Discrete fourier transform
Ω_s	Nyquist sampling frequency
J_k^+	Excitory input
J_k^-	Inhibitory input
$\mathbf{z}$	Weight vector in ART network
M	Number of units in input of ART network
N	Number of output units of ART network

Chapter 1

Introduction

Fault diagnosis has been an active area of research in both the engineering and computer science communities. In spite of the many advances in this area, fault diagnosis still remains challenging with many questions unanswered. With advances made in technology, complexity is a major factor that we have to deal with. Complexity confronts us, perhaps, when something breaks down. We are forced to come up with more effective techniques and analysis for detecting and diagnosing such anomalies. Diagnostic analysis is not always quantitative. In fact much detection and diagnostic analysis is heuristic, and results from repeated and long time cognitive experiences. In the last decades, ac drive installation has shown tremendous growth both in size and complexity. The interruption of service due to faults in a motor drive installation is often costly and could interfere with public safety in some installations.

These motors are exposed to a wide variety of environments and conditions which make the motor subject to incipient faults. These incipient faults, if left undetected, contribute to the degradation and eventual failure of the motors. With proper monitoring and fault detection schemes, the incipient faults can be detected in their early stages, and maintenance and down time expenses can be reduced while also improving safety. Thus, the motor incipient fault detection can be used in the motor preventive maintenance programs also.

1.1 Incipient Fault Detection of Induction Motors

1.1.1 A Brief Description

Although rotating machines are usually well constructed and robust, the possibility of incipient faults is inherent in the machines due to the stresses involved in the conversion of electrical energy to mechanical energy and vice versa [1, 2]. Incipient faults within a machine will affect the performance of the machine before major failures occur. With proper system monitoring and fault detection schemes, maintenance costs can be reduced and reliability of the machines can be improved significantly.

An experienced engineer may detect and diagnose the motor faults by observing the motor's operating performances. However, experienced engineers are expensive and difficult to train. It is, therefore desirable to automate

the system monitoring and fault detection schemes rather than to rely on an expert to perform continuous on-line monitoring. Several fault detection methods have been developed, each with their own prospects and constraints. Some techniques require expensive diagnostic equipment and/or off-line fault analysis to determine the motor condition. For instance, the radio frequency monitoring scheme injects radio frequency signals to the stator winding of a machine and measures the changes of the signal waveform to determine whether the winding insulation contains faults [2]. This technique requires expensive equipment and is justified only for use with large and expensive machines. Other popular techniques, such as particle analysis which requires bringing the motor oil samples to a laboratory for analysis [2] to determine the motor condition, are more suitable for overhaul or routine check-up rather than on-line monitoring and fault detection.

The parameter estimation approach [3] is a non-invasive fault detection scheme. Non-invasive fault detection schemes are based on easily accessible and inexpensive measurements to predict the motor condition without disintegrating the motor structure. These schemes are suitable for on-line monitoring and fault detection purposes. Due to their economical and non-destructive features, non-invasive techniques are often preferred by many engineers. However, the parameter estimation approach requires an accurate mathematical model and an elaborate understanding of the system dynamics based on a set of system parameters. The parameters are usually chosen to reflect the motor conditions. For example, the bearing condition will affect the damping coefficient of the motor's mechanical equation. As the

bearing wears out, the damping coefficient increases. Thus, the parameter estimation approach can be based on the motor's mechanical equation and measurements to estimate the value of the damping coefficients. After estimating the numerical value of the chosen parameter, a means to translate the estimated numerical values to qualitative description is required. The major difficulty with the parameter estimation approach is that an accurate mathematical model is required, and is usually difficult to obtain. Other techniques like non-parametric surface fitting method also require case by case specific mathematical analysis. In addition, the interpretation of the fault conditions, which is a fuzzy concept using rigorous mathematical formulations, is generally impractical and inaccurate.

On the other hand, use of an artificial neural network for fault detection is also a non-invasive technique [3, 4] . But, unlike the parameter estimation approach, neural networks can perform fault detection based on measurements and training without the need of complex and rigorous mathematical models. In addition, heuristic interpretation of the motor conditions, which sometimes only humans are capable of doing, can be easily implemented in the neural network through supervised training.

1.1.2 Mathematical Analysis of Induction Motor

In order to successfully perform fault detection, different sets of criteria are needed to define a motor's status at different operating conditions. The fault detection of a 3-phase induction motor has been used for illustration

purposes. It is worthwhile to describe the fault detection problem in mathematical terms to facilitate future discussions on the subject.

Mathematical Description of Motor Dynamics

An induction motor can be described by the following state equations in the stationary reference frame:

$$\frac{d}{dt}\begin{bmatrix} i_s \\ \phi_r \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}\begin{bmatrix} i_s \\ \phi_r \end{bmatrix} + \begin{bmatrix} B_1 \\ 0 \end{bmatrix} v_s$$

$$= Ax + Bv_s \tag{1.1}$$

$$i_s = Cx \tag{1.2}$$

Where

- $$Stator\ Current\ i_s = \begin{bmatrix} i_{ds} & i_{qs} \end{bmatrix}^T$$
- $$Rotor Flux\ \phi_r = \begin{bmatrix} \phi_{dr} & \phi_{qr} \end{bmatrix}^T$$
- $$Stator\ Voltage\ v_s = \begin{bmatrix} v_{ds} & v_{qs} \end{bmatrix}^T$$
- $A_{11} = -(R_s/(\delta L_s) + (1-\delta)/(\delta\tau_r))\ \mathrm{I} = a_{r11}\mathrm{I}$
- $A_{12} = M/(\delta L_s L_r)((1/\tau_r)\ \mathrm{I}\omega_r\mathrm{J}\ ) = a_{r12}\mathrm{I} + a_{i12}\mathrm{J}$

- $A_{21} = (M/\tau_r)\ \mathrm{I} = a_{r21}\mathrm{I}$
- $A_{22} = -(1/\tau_r)\ \mathrm{I} = \omega_r \mathrm{J} = a_{r22}\mathrm{I} + a_{i22}\mathrm{J}$
- $B_1 = 1/(\delta L_s)\mathrm{I} = b_1 \mathrm{I}$

$$C = \begin{bmatrix} I & 0 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

R_s and R_r are stator and rotor resistances, respectively,
L_s and L_r are stator and rotor self inductances, respectively,
M is mutual inductance,
Leakage coefficient $\delta = 1 - M^2/(L_s L - r)$,
Rotor time constant $\tau_r = L - r/R_r$ and
ω_r is motor angular velocity in radian/seconds.

Input current depends on the motor parameters R_s, R_r, L_s, L_r, M and δ. An internal fault in the machine will be reflected in the stator current of the machine. It is possible therefore, to detect the fault from the analysis of stator current.

1.2 Artificial Neural Network for Fault Detection

1.2.1 Expert Approach for Fault Detection

As stated previously, the interpretation of a motor's condition based on numerical value is usually a difficult task because fault detection is a fuzzy concept and usually requires experience [5]. Therefore, in many cases, heuristic interpretation of the results, which only humans are capable of doing becomes necessary. An experienced engineer can diagonose the motor's condition based on its operating conditions and measurements without knowing exact mathematical model of the motor. The approach is simple and reliable, and the complicated mathematical relation is embedded in the engineer's knowledge about the motor. However, an experienced engineer may not be able to give detailed explanations regarding his/her reasoning and logic used to make the decisions, simply because experience belongs to the fuzzy logic realm and is difficult to describe accurately in exact mathematical terms.

As it turns out, this human expertise approach has many advantages over the parameter estimation approach. However, the major drawback of the human expertise approach is that experience is difficult to transfer and automate. Both researchers and engineers usually transfer experience and knowledge through languages and mathematics, which are sometimes time consuming and inaccurate. In practice, the experience and the knowledge used by expert engineers to perform motor fault detection and or diagnosis

historical fault detection data gathered by the experts.

1.2.2 Learning Skills of Artificial Neural Networks

With the emerging technology of artificial neural networks, the human expertise approach can be mimicked and automated [6, 7, 8]. Artificial neural networks(ANN) can be trained to perform motor fault detection by learning expert's knowledge using a representative set of motor data [9]. In the case of an induction motor, incipient faults can be detected by analyzing the frequency spectrum of the stator current as shown in Fig. 1.1(a)-(b). Now it is clear that the stator current spectrum carries the signature of an internal fault within the machine. So by training an ANN with the values of spectral component related to particular faults without the need of mathematical models, the complexity of the parameter estimation approach can be avoided. Once the ANN is trained appropriately, the network weights contain the knowledge needed to perform fault detection, which is equivalent to the expertise gained by an engineer over the years in machine fault diagnosis.

1.3 Literature Review

ANNs have been proven to be capable of successfully performing motor fault detections [9, 10]. One of the advantages of this type of pattern recognition techniques is that it can save time in information processing in run time, where all the computational complexities are done off-line in the training

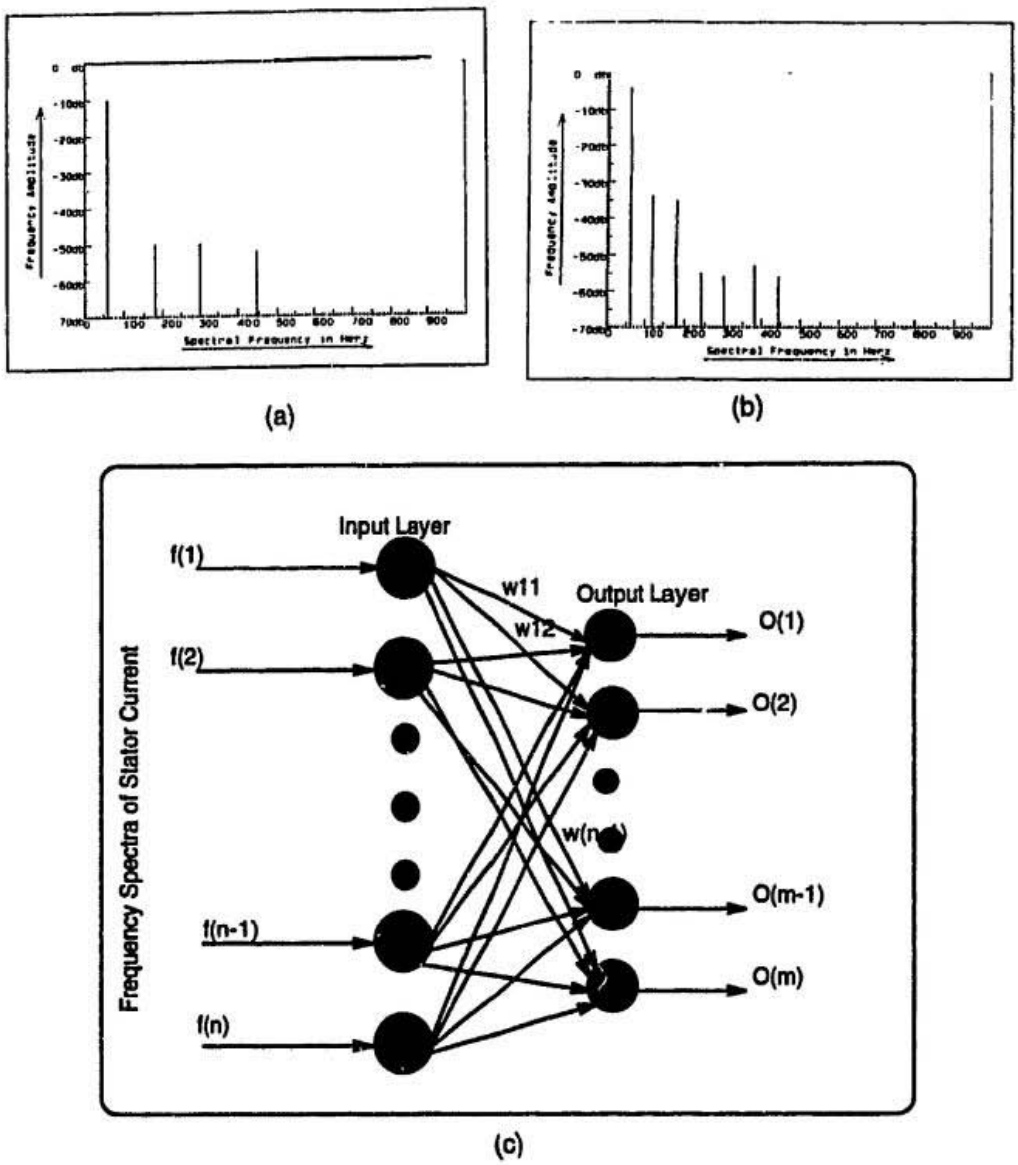

Figure 1.1: (a) Frequency spectra of input current of a healthy machine.(b) Frequency spectra of input current of a faulty machine. (c). Model of Neural Network based fault related spectra identification system.

period of the network.

When developing an ANN based system to perform a particular pattern classification problem, typically the process is to gather a set of examples or training patterns, then using these examples to train the underlying ANN. During the training, the information is coded in the system by the adjustments of the weight values. Once the training is deemed to be adequate, the system is ready to be used in the real-time situations, and usually no additional weight modification is required.

This operational scenario is acceptable provided the problem domain has well-defined boundaries and is stable. Under such conditions it is possible to define an adequate set of training inputs for whatever problem being solved. Unfortunately, like many realistic situations involving incipient fault detection of induction motors, the environment is neither bounded nor stable. To solve this dynamic behaviour conventional Feed-forward Neural Network(FNN) suffers a major set back.

Mo-Yuen Chow and others [3]-[5],[9]-[11] have done significant works in neural network based incipient fault diagnosis of induction motors. But they have used FNN as fault diagnosis tool and in their research work they have neglected this dynamic operational scenario which is an indispensable part in real-world environment. M.F. Abdel Mageed and his colleagues [12] have used Hierarchical neural network, but this neural network also suffers the same limitation as FNN. The same limitation also prevails in the research works of F.Filippetti [13, 14], Chin-Teng [6] and their colleagues. Moreover,

the reporting ability of the neural network, if it cannot diagnose a particular fault has not been considered by them. So, there is a need to carry out a research work to find suitable neural network which is not only capable to diagonose a fault but also can report if it cannot, so that preventive steps can be taken to update the neural network to adapt to this new fault, while retaining the already acquired knowledge without retraining of the already trained patterns.

Mathematical analysis as mentioned in section(1.1.2) makes sense that wave shape of the stator current carries the signature of internal condition of the machine. R. Natarajan [15] has used the stator current to diagonise the fault by only measuring its value, not though the spectral analysis, which is necessary for neural network based fault detection scheme to get better result. F.Filippetti and M.Martelli [13] have considered the frequency spectra of the stator current as the key fault related information carrier of the fault diagnosis scheme, but they have not reported a detailed study of frequency spectra of the stator current at different fault conditions. B.C. Papadias [16] and others have given an outline to develop an expert system for troubleshooting of electrical machines, but the collection of fault related information is not mentioned. While the focus of research work of Mo-Yuen Chow and others [3]-[5],[9]-[11] is towards the applicability of neural network in the incipient fault diagnosis of induction motors specially in bearing fault, they have not considered the spectral analysis of the stator current. At this present state, it is evident that it is important to pay attention to collect the frequency spectra of the stator current at different fault conditions.

1.4 Objective of the Present Work

The long-term objective of this work is to improve the state-of-the-art of incipient fault diagnosis of induction motors. The specific short-term objectives are summarized in the following three points:

- To find a neural network suitable for incipient faults detection of electric machines. This fault diagnosis scheme is not only capable of detecting a fault but also can report if it cannot identify a particular fault so that necessary preventive steps can be taken to update the underlying neural network to adapt to this undetected fault.

- To develop a laboratory set-up to collect frequency spectra of the stator current of induction motor in real-time using digital signal processing techniques, and to collect frequency spectra of the stator current at different fault conditions for the experimental induction motor.

- To train the selected neural network with fault related frequency spectra of the stator current and to study the performance of the trained network to diagonise faults in noise free as well as noisy conditions.

1.5 Overview of Thesis

The contents of the theses can be summarized in the following chapters:

Chapter 2 covers a survey of available neural networks. A comparative study has been done to select a neural network to satisfy one of the objectives.

Chapter 3 discusses the software implementation of the selected neural network. It's performance has been tested with a number of examples.

Chapter 4 explains the related algorithms and techniques to collect the fault related frequency spectra of the stator current of a three phase induction motor. This chapter also gives a brief outline of the digital signal processing (DSP) board as well as the DSP library. Salient features of the software developed as part of this work to acquire real time frequency spectra of the stator current has been also described in this chapter.

Chapter 5 explains the performances of the selected neural network to classify faults based on fault related frequency spectra of the stator current. A model based on the selected neural network for on-line incipient fault diagnosis system for the induction motor has also been reported in this chapter.

Chapter 6 contains the conclusions and recommendations for future work. It has been explained that there is a good prospect to do further work on the irrespective of design parameter, type of machine as well as operating conditions fault diagnosis system. Moreover, the scope of development of multi-machine fault diagnosis system to make it cost-effective and user-friendly has also been emphasized.

Chapter 2

An overview of Artificial Neural Networks

2.1 Introduction

A neural network is a massively parallel distributed processor that has natural propensity for storing experimental knowledge and making it available for use [17]. It mimics the brain in two respects:

a). Knowledge is acquired by the network through a learning process.

b). Inter-neuron connection strengths, usually known as synaptic weights, are used to store the knowledge.

The procedure used to perform the learning process is called a "learning algorithm", the function of which is to modify the synaptic weights of the network in an orderly fashion so as to attain a desired design objective. The

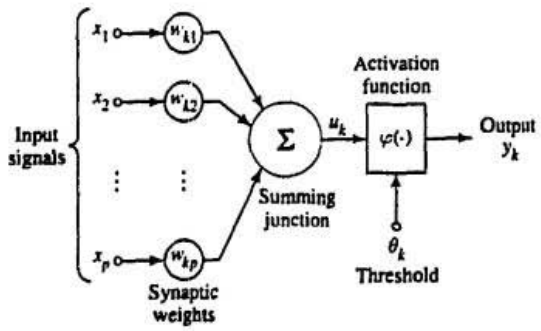

Figure 2.1: Model of a neuron

use of neural networks offers a number of benefits, among them nonlinearity, input-output mapping and adaptivity are most important.

2.1.1 Models of a Neuron

Fig. 2.1 shows the model of a neuron. Three basic elements of the neuron are explained as follows:

1. A set of synapses or connecting links, each of which is characterized by a weight or strength of its own.

2. An Adder for summing the input signals.

3. An activation function for limiting the amplitude of the output of the neuron.

A neuron K can be explained by the following equations:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \tag{2.1}$$

$$y_k = \varphi(u_k - \theta_k) \tag{2.2}$$

where $x_1, x_2, ..., x_p$ are input signals; $w_{k1}, w_{k2}, ..., w_{kp}$ are the synaptic weights of neuron k; u_k is the linear combined output; θ_k is the threshold; φ is the activation function; and y_k is the output signal of the neuron. The use of threshold θ_k has the effect of applying an affine transformation to the output u_k of the linear combiner in the model as shown by

$$v_k = u_k - \theta_k \tag{2.3}$$

The output can be represented by the following equation:

$$y_k = \varphi(v_k) \tag{2.4}$$

Types of activation function

Generally used three different types of activation functions are described below [17]:

1. **Threshold Function:**

For this type of function as shown in Fig. 2.2(a)

$$\varphi(v) = \begin{cases} 1 & if \quad v \geq 0 \\ 0 & if \quad v < 0 \end{cases} \tag{2.5}$$

$$y_k = \begin{cases} 1 & if \quad v_k \geq 0 \\ 0 & if \quad v_k < 0 \end{cases} \tag{2.6}$$

$$v_k = \sum_{j=1}^{p} w_{kj} x_j - \theta_k \tag{2.7}$$

2. **Piecewise-Linear Function:**

A Piecewise-linear function as shown in Fig. 2.2(b) can be explained by the following equation:

$$\varphi(v) = \begin{cases} 1, & if \quad v \geq \frac{1}{2} \\ v, & if \quad -\frac{1}{2} > v > -\frac{1}{2} \\ 0, & if \quad v \leq -\frac{1}{2} \end{cases} \tag{2.8}$$

3. **Sigmoidal Function:**

The Sigmoidal function as shown in Fig. 2.2(c) is by far the most common form of activation function used in the construction of artificial neural networks as explained by the following equation:

$$\varphi(v) = \frac{1}{1 + exp(-av)} \tag{2.9}$$

where a is the slope parameter.

2.1.2 Network Architecture

Learning algorithm to train a neural network depends on the way it is structured. In general, there are four different classes of network architectures:

1. Single-Layer Feedforward Networks.
2. Multilayer Feedforward Network.
3. Recurrent Networks.
4. Lattice Structures

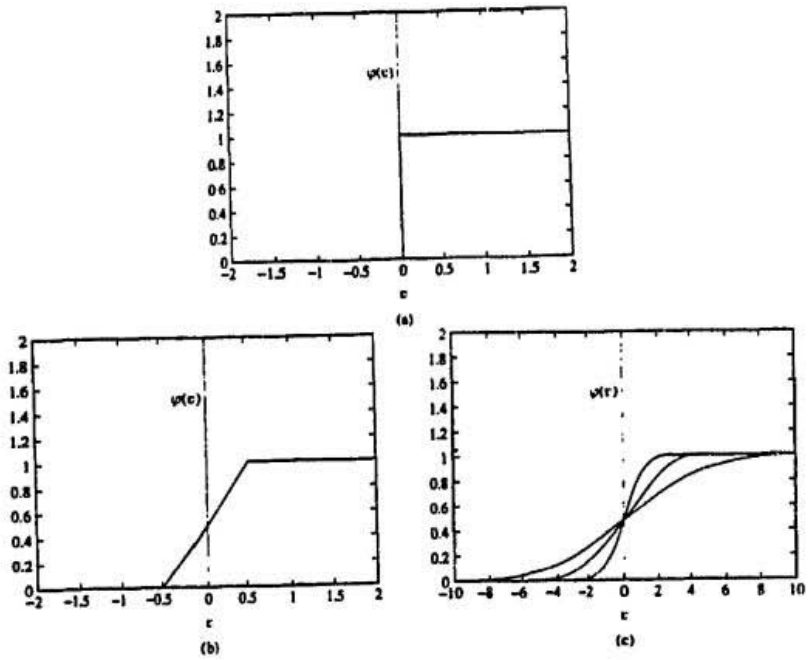

Figure 2.2: Activation Functions: (a) Threshold Function. (b) Piecewise-Linear Function (c) Sigmoidal Function

2.1.3 Artificial Intelligence and Neural Networks

The aim of artificial intelligence(AI) is the development of paradigms or algorithms that require machines to perform tasks that apparently require cognition when performed by humans. An AI system must be capable of:

1. storeing knowledge;
2. appling the knowledge stored to solve problems;
3. acquiring new knowledge through experience;

An AI system has three key components: representation, reasoning, and learning. AI can be described as the formal manipulation of a language of algorithms and data representations in a top-down fashion. On the other hand, neural network can be described as parallel distributed processors with a natural learning capability, and which usually operate in bottom-up fashion. For the implementation of cognitive tasks, it therefore appears that rather than seek solution based on AI or neural network alone, a more potentially useful approach would be to build structured connectionist models that incorporate both of them.

Some important neural networks have been explained briefly in the folowing sections ending with a comparative analysis for the selection of proper neural network for incipient fault detection of an induction motor.

2.2 Backpropagation

Backpropagation neural network(BPN) as shown in Fig. 2.3 learns a predefined set of input-output example pairs by using a two-phase propagate-adapt cycle [18]. After an input pattern has been applied as stimulus to the first layer of the network units, it is propagated through each upper layer until an output is generated. This output pattern is then compared to the desired output,and error signal is computed for each output unit. The error signals are then transmitted backward from the output layer to each node in the intermediate layer that contributes directly to the output. However, each node in the intermediate layer receives only a portion of the total error signal, based roughly on the relative contribution the unit made to the

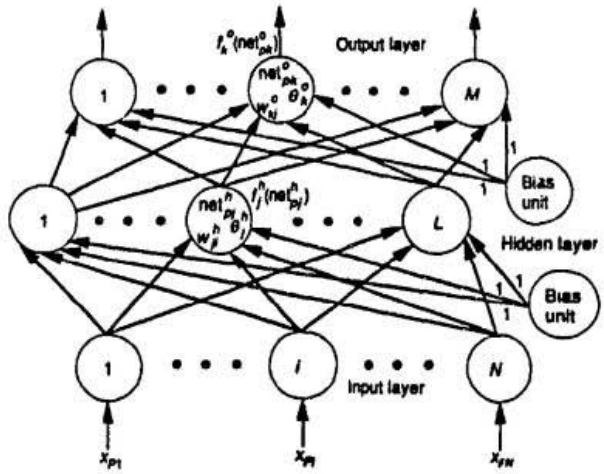

Figure 2.3: The three layer BPN architecture

original output. This process repeats, layer by layer until each node in the network has received an error signal that describes its relative contribution to the total error. The training procedure of a BPN can summarized in the following points:

1. The vector, $\mathbf{x}_p = (x_{p1}, x_{p2}, ..., x_{pn})^t$ is applied to the input units.

2. The net-input values to the hidden layer units are calculated:

$$net^h_{pj} = \sum_{i=1}^{N} w^h_{ji} x_{pi} + \theta^h_j \tag{2.10}$$

3. The outputs from the hidden layer are calculated:

$$i_{pj} = f^h_j(net^h_{pj}) \tag{2.11}$$

4. For output layer the net-input values to each unit is calculated:

$$net^o_{pk} = \sum_{j=1}^{L} w^o_{kj} i_{pj} + \theta^o_k \tag{2.12}$$

5. The outputs are calculated:

$$O_{pk} = f_k^o(net_{pk}^o) \tag{2.13}$$

6. The error terms of the output units are calculated:

$$\delta_{pj}^h = f_j^{h'}(net_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \tag{2.14}$$

7. The weight on the output layer is updated:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} \tag{2.15}$$

8. Weights in the hidden layer are updated:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \eta \delta_{pj}^h x_i \tag{2.16}$$

The following equation is the measure of how well the network is learning.

$$E_p = \frac{1}{2} \sum_{k=1}^{M} \delta_{pk}^2 \tag{2.17}$$

When the error is acceptably small for each of the training-vector pairs, training can be discontinued.

2.3 The Binary Associative Memory(BAM) and the Hopfield Memory

A type of memory called an associative memory is a subject matter of this section. In fact, the concept of associative memory is a fairly intuitive one: associative memory appears to be one of the primary functions of the brain [18].

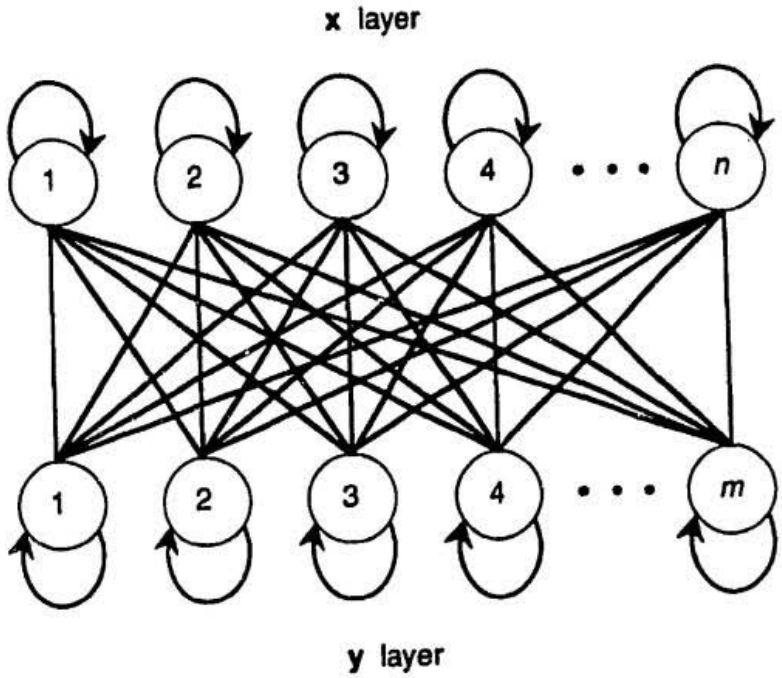

Figure 2.4: BAM architecture

2.3.1 The BAM

The BAM consists of two layers of processing elements that are fully interconnected between the layers. The units may, or may not, have feedback connection to themselves. The general case is shown in Fig. 2.4. For the L vector pairs that constitute the set of examplers should be stored, the following matrix can be constructed:

$$\mathbf{w} = \mathbf{y}_1\mathbf{x}_1^t + \mathbf{y}_2\mathbf{x}_2^t + ... + \mathbf{y}_L\mathbf{x}_L^t \tag{2.18}$$

This equation gives the weights on the connections from the x layer to the y layer. To construct the weights for the x layer units, it is necessary simply to take the transpose of the weight matrix, $\mathbf{w}^t$.

BAM Mathematics

On the y layer

$$\mathbf{net}^y = \mathbf{wx} \tag{2.19}$$

where $\mathbf{net}^y$ is the vector of the net-input values on the y layer. In terms of the individual units, y_i,

$$net_i^y = \sum_{j=1}^{n} w_{ij} x^j \tag{2.20}$$

On the x layer

$$\mathbf{net}^x = \mathbf{w}^t\mathbf{y} \tag{2.21}$$

$$net_i^x = \sum_{j=1}^{m} y_j w_{ji} \tag{2.22}$$

The quantities n and m are dimensions of the x and y layers, respectively. The output values for each processing element depends on the net input value, and on the current output value of the layer. The new value of y at time step $t+1$, $\mathbf{y}(t+1)$ is related to the value of y at time step t, $\mathbf{y}(t)$ by

$$y_i(t+1) = \begin{cases} +1, & net_i^y > 0 \\ y_i(t), & net_i^y = 0 \\ -1, & net_i^y < 0 \end{cases} \tag{2.23}$$

Similarly, $\mathbf{x}(t+1)$ is related to $\mathbf{x}(t)$ by

$$x_i(t+1) = \begin{cases} +1, & net_i^x > 0 \\ x_i(t), & net_i^x = 0 \\ -1, & net_i^x < 0 \end{cases} \tag{2.24}$$

BAM Processing

To recall the information using the BAM, the following steps should be performed:

1. The initial vector pair, $(\mathbf{x_0}, \mathbf{y_0})$ is applied to the input elements of the BAM.

2. Information is propagated from the **x** layer to the **y** layer; and the values on the **y**–layer are updated.

3. The updated **y** information is propagated back to the **x** layer and the units are updated.

4. Steps 2 and 3 are repeated until there is no further change in the units on each layer.

2.3.2 The Hopfield Memory

Hopfield memory(HM) can described as a derivative of the BAM [18]. There are two types of Hopfield memory as described below.

Discrete Hopfield Memory

Fig. 2.5 illustrate the structure of discrete Hopfield Memory.

Continuous Hopfield Memory

Continuous Hopfield Memory has same useful properties of associative memory but it can accept analog input making it closer to natural neuron. Moreover, it can be represented by analog eletronic circuit making it suitable for VLSI implementation.

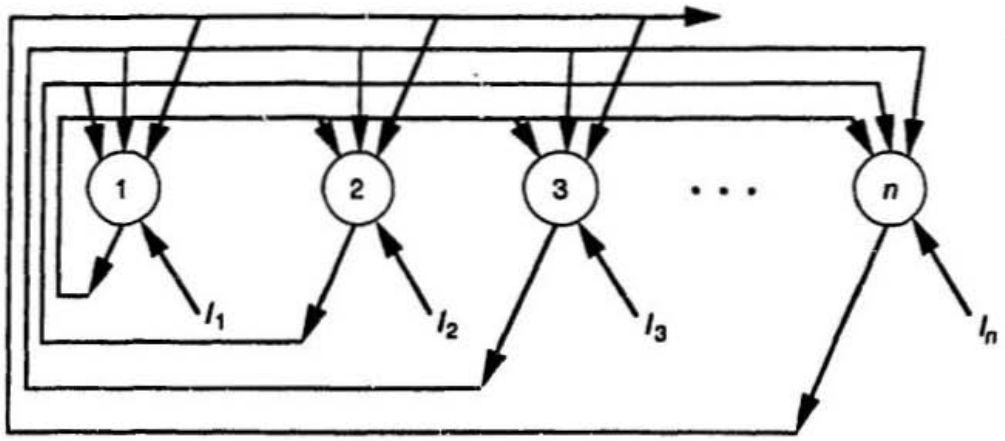

Figure 2.5: Discrete Hopfield Memory

2.4 Simulated Annealing

It is possible to extend the analogy between information theory and statistical mechanics in order to place neural network [18] in contact with a heat reservoir at some, as yet undefined temperature. If so, then it is possible to perform a simulated annealing process whereby gradually lowering the temperature while processing takes place in the network, in the hopes of avoiding a local minimum on the energy landscape as shown in Fig. 2.6. This situation can be better explained in the neural network known as Boltzman Machine.

2.4.1 The Boltzman Machine

The basic architecture of this type of neural network can be explained by Fig. 2.7 [18].

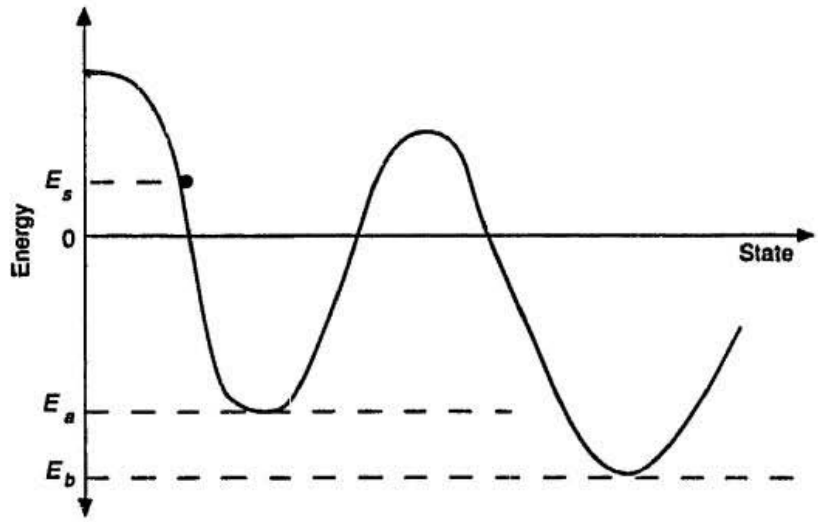

Figure 2.6: A simple energy landscape with two minimal, a local minimum and a global minimum

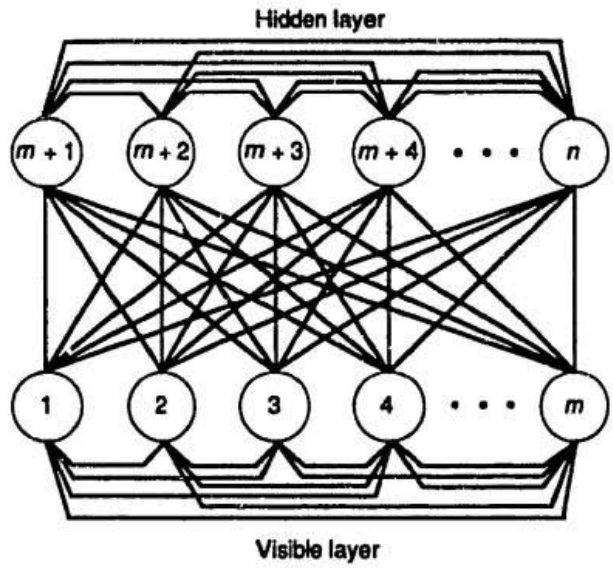

Figure 2.7: The Botzman Completion Architecture

As the discrete HM, the system energy can be calculated from

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1,j\neq i} w_{ij}x_i x_j. \tag{2.25}$$

Where n is the total number of units in the network, and x_k is the output of the kth unit. The energy difference between the system with $x + k = 0$ and $x_k = 1$ is given by

$$\Delta E_k = net_k \tag{2.26}$$

The recall procedure is done by the simulated annealing procedure with $\mathbf{x}'$ as the starting vector on the visible units. The procedure is described by the following algorithm:

1. All the outputs of all known visible units are forced to the values specified by the initial input vector, $\mathbf{x}'$
2. All unknown visible units and all hidden units are assigned random output values from set 1,0.
3. A unit, x_k, at random is selected and its net-input, net_k is calculated.
4. Steps 3 and 4 are repeated until all units have had some probability of being selected for update. This number of unit-updates defines a processing cycle.
5. Step 5 is repeated for several processing cycle, until thermal equilibrium has been reached at given temperature, T.
6. Temperature T is lowered and step 3 through 7 are repeated.

Learning in Boltzman Machines

A reasonable approach to train a Boltzman machine can summarized in the following way:

1. Artificially the temperature of the Boltzman machine is raised to some finite value.

2. The system is annealed until the equilibrium is reached at some low temperature.

3. The weights of the network is adjusted so that the difference between the observed probability distribution and canonical distribution is reduced.

4. Steps 1 through 3 are repeated until the weights no longer change.

2.5 The Counter Propagation Network

For a given set of vector pairs, $(x_1,y_1),(x_2,y_2),.....,(x_n,y_n)$, the counter propagation network(CPN) can learn to associate an vector $\mathbf{x}$ on the input layer with a vector $\mathbf{y}$ at the output layer [18]. If the relationship between $\mathbf{x}$ and $\mathbf{y}$ can be described by a continuous function ϕ , such that $\mathbf{y} = \phi(\mathbf{x})$, then CPN will learn to approximate this mapping for any value of $\mathbf{x}$ in the range specified by the set of training vectors. This situation is known as forward mapping of CPN and its structure is shown in Fig. 2.8.

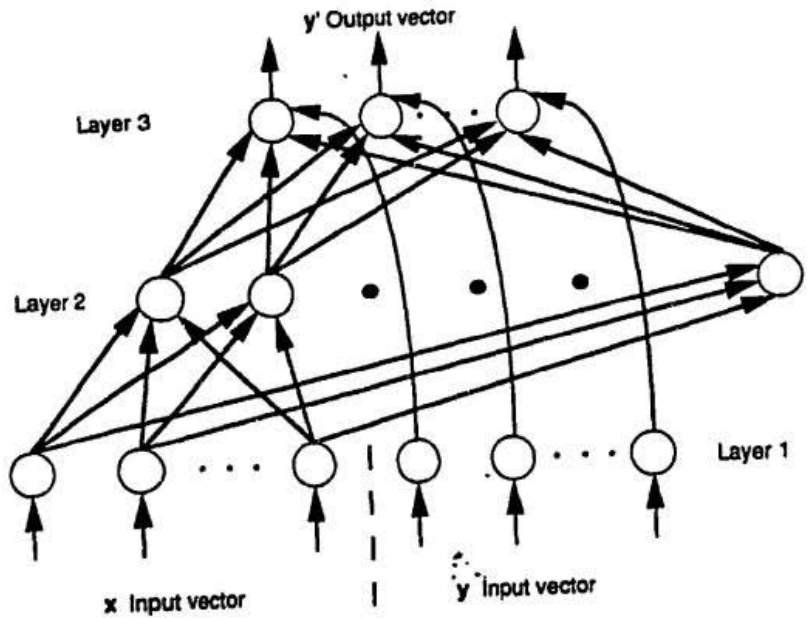

Figure 2.8: Forward-mapping CPN

2.5.1 CPN Building Blocks

The building blocks of CPN are explained in following section:

The Input Layer

The input layer of processing elements is shown in Fig. 2.9. The total input pattern intensity is given by $I = \sum_i I_i$. Corresponding to each I_i, a quantity can be defined

$$\Theta_i = I_i \left(\sum_i I_i \right)^{-1} \tag{2.27}$$

The vector, $(\Theta_1, \Theta_2, ..., \Theta_n)^t$ is called a reflectance pattern. It should be noted that this pattern is normalized in the sense that $\sum_i \Theta_i = 1$.

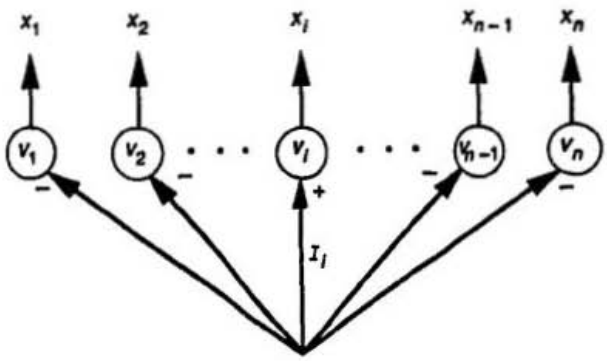

Figure 2.9: Layer of input units of a CPN

The Instar

The **instar** is a single processing element as shown in Fig. 2.10

Assuming the initial output is zero, and that a nonzero input vector is present from time $t = 0$ until time, t when the output can be defined

$$y(t) = \frac{b}{a} net(1 - e^{-at}) \tag{2.28}$$

The equilibrium value of $y(t)$ is defined by

$$y^{eq} = \frac{b}{a} net. \tag{2.29}$$

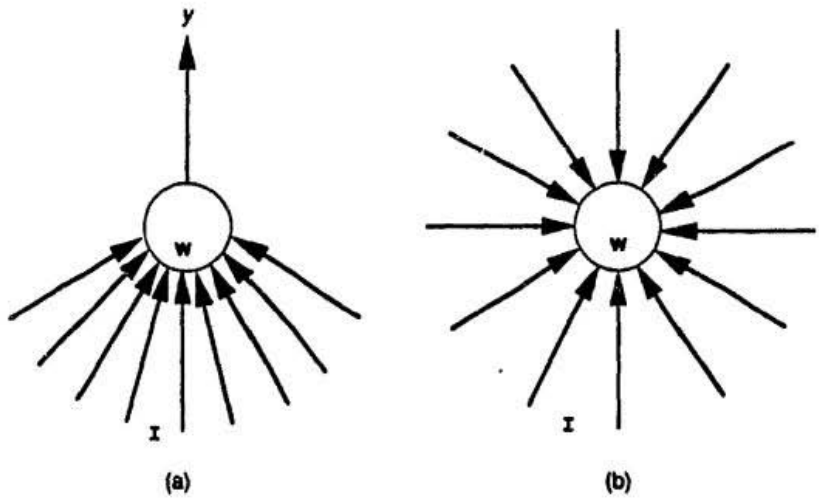

Figure 2.10: This figure shows (a) the general form of the processing elements (b) the instar form of processing elements

Competitive Networks

Fig. 2.11 illustrates the interconnection that implements competition among the instars. The unit activations are determined by differential equations and simplest form is defined by

$$\dot{x}_i = -Ax_i + (B - x_i)\,[f(x_i) + net_i] - x_i \left[\sum_{k \neq i} f(x_k) + \sum_{k \neq i} net_k\right] \qquad (2.30)$$

The Outstar

Fig. 2.12 shows an outstar. It is composed of all of the units in CPN outer layer and a single hidden-layer unit. During the training process, the output values of the outstar can be calculated from

$$\dot{y}_i = -\dot{y}_i + by_i + cnet_i. \qquad (2.31)$$

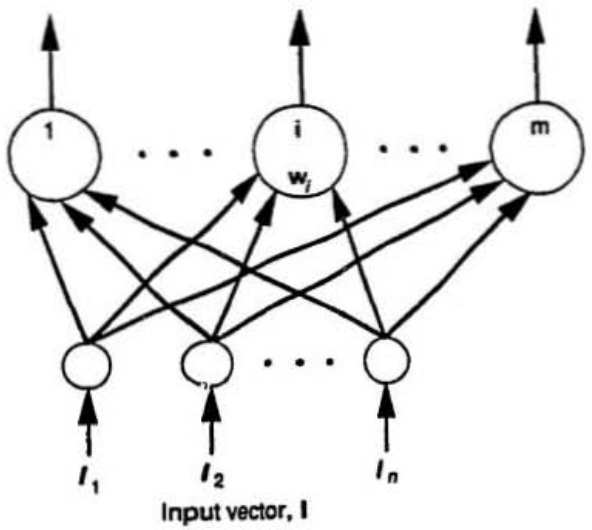

Figure 2.11: A layer of instars arranged in c CPN

2.5.2 Training the CPN

The training procedure of CPN can be summarized in the following points:

1. An input vector is selected from all the input vectors to be used for the training.

2. Input vector is normalized and is applied to the CPN competitive layer.

3. The winner should be determined .

4. For the winning unit, only, $\alpha(\mathbf{x} - \mathbf{w})$should be calculated and unit's weight should be updated according to the following equation:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha(\mathbf{x} - \mathbf{w}) \tag{2.32}$$

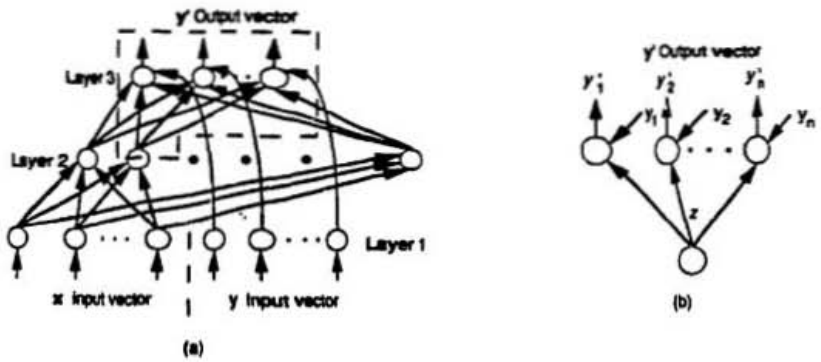

Figure 2.12: Outstar and its relationship to the CPN architecture. (a). Outstar structures in CPN network (b). A single outstar unit is shown

5. Steps 1 through 4 should be repeated until all input vectors have been processed once.

6. Step 5 should be repeated until all input vectors have been classified properly.

7. The network should be tested to see the effectiveness.

2.5.3 Forward Mapping

It has been assumed that all training has occurred and that the network is now in a production mode. For the input vector **I** it is necessary to find the corresponding **Y** vector. The required processing can be done by the following algorithm:

1. The input vector should be normalized, $x_i = I_i/(\sqrt{\sum_n I_n^2})$

2. Input vector should be applied to the x-vector portion of layer 1 and a zero vector should be applied to the y-vector portion of the same layer.

3. Since the input vector is already normalized, the input layer only distributes it to the units on layer 2.

4. Layer 2 is a winner-take-all competitive layer. The output of each unit can be calculated as follows

$$z_i = \begin{cases} 1, & \|net_i > \|net_j\| \; for \; all \; j \neq i \\ 0, & otherwise \end{cases} \tag{2.33}$$

5. The single winner on layer 2 excites an outstar

2.6 Self-Organizing Maps

In Self-Organizing Maps(SOM), the CPN network is modified such that, during the learning process, the positive feedback will extend from the central(the winning) unit to the other units in some finite neighborhood around the central unit [18]. In the competitive layer of the CPN, only the winning unit was allowed to learn; in the SOM, all the units in the neighborhood that receive positive feedback from the winning unit participate in the learning process.

2.6.1 Unit Activations

The following equation defines the activation of the processing elements

$$\dot{y}_i = -r_i(y_i) + net_i + \sum_j z_{ij} y_j \tag{2.34}$$

The function $r_i(y_i)$ is a general form of a loss term. If z_{ij} takes the form of the Mexicanhat function, then the network will exhibit a bubble of activity around the unit with the largest value of net input.

2.6.2 The SOM Learning Algorithm

The learning process can be defined by the following equation

$$\dot{\mathbf{w}}_i = \alpha(t)(\mathbf{x} - \mathbf{w}_i)U(y_i) \tag{2.35}$$

where the $\mathbf{w}_i$ is the weight vector of the ith unit and $\mathbf{x}$ is the input vector. For an input vector $\mathbf{x}$, the winning unit can be determined by

$$\|\mathbf{x} - \mathbf{w}_c\| = min_i \|\mathbf{x} - \mathbf{w}_i\| \tag{2.36}$$

where index c refers to the winning unit. This can be explained as

$$\mathbf{w}_i(t+1) = \begin{cases} \mathbf{w}_i(t) + \alpha(t)(\mathbf{x} - \mathbf{w}_i(t) & i \in N_c \\ 0 & otherwise \end{cases} \tag{2.37}$$

2.7 Spatiotemporal Pattern Classification

Neural networks as described previously are suitable for the recognition of spatial information patterns. Spatiotemporal pattern classifier can classify

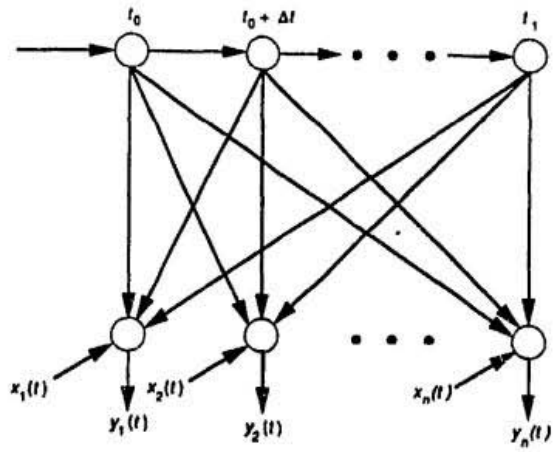

Figure 2.13: Grossberg's formal avalanche structure

time-correlated sequence of spatial patterns [18].

2.7.1 The Formal Avalanche

The foundation for the development of the network architectures discussed in this section is the formal avalanche structure by Grossberg as shown in Fig. 2.13.

2.7.2 Architectures of Spatiotemporal Networks(STNS)

Fig. 2.14 shows an arrangement that generates the spatiotemporal patterns (STPs) from spoken word. At each instant of time, the output of the spectrum analyzer consists of a vector whose components are the powers in the

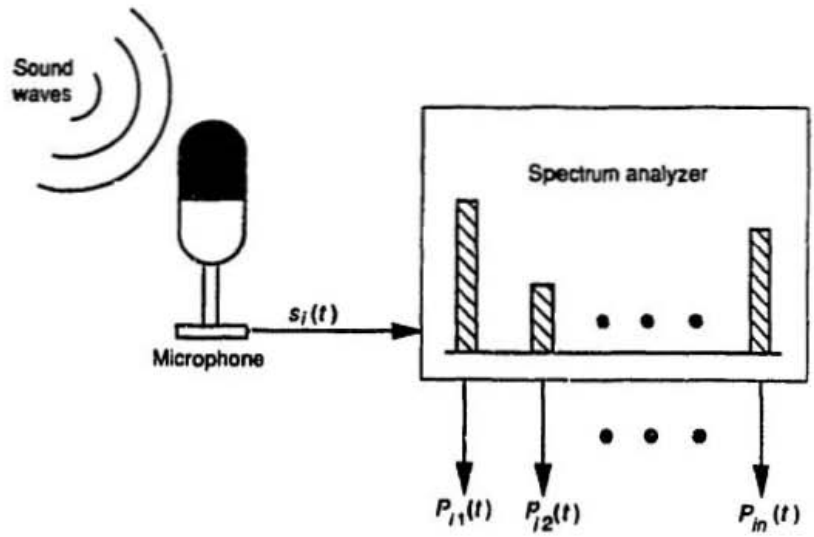

Figure 2.14: Power Spectra generated from speech

various channels.

2.8 The Neocognitron

This is a special type of neural network tailored for the recognition of hand written characters [18]. The main pathways for neuron leading from the retina back to area of the brain as the visual, or striate, cortex.

2.8.1 Neocognitron Architecture

The processing elements(PEs) of the neocognitron are organized into modules that shall refer to as levels.Each level consists of two layers: a layer of simple cells, or S-cells, followed by a layer of complex cells, or c-cells.

2.8.2 Neocognitron Data Processing

S-cell Processing

Here it has been considered that the index k_l refers to the kth plane on level l. Each cell on a plane can be labeled with a two-dimensional vector, with n indicating its position on the plane and v refer to the relative position of a cell in the previous layer lying in the receptive field of the unit n. The equation for the S-cell can be written as :

$$U_{sl}(k_l, \mathbf{n}) = r_l \phi \left[\frac{1 + \sum_{k_{l-1}=1}^{k_{l-1}} \sum_{v \epsilon A_l} a_l(k_{l-1}, v, k_l).U_{C_{l-1}}(k_{l-1}, \mathbf{n}+\mathbf{v})}{1 + \frac{r_l}{1+r_l} b_l(k_l).V_{Cl}n} \right] \quad (2.38)$$

where the function ϕ is a linear function given by

$$\phi(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (2.39)$$

C-cell Processing

Usually, units on a given C-plane receive input connections from one, or at most a small number of S-planes on the preceding layer. The output of a C-cell is given by

$$U_{Cl}(k_l, n) = \psi \left[\frac{1 + \sum_{k_l=1}^{k_l} jl(k_l, k_l) \sum_{v \epsilon D_l} d_l(v).U_{sl}(k_l, \mathbf{n}+\mathbf{v})}{1 + V_{sl}(n)} - 1 \right] \quad (2.40)$$

The function ψ is defined by

$$\psi(x) = \begin{cases} \frac{x}{\beta+x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.41}$$

Where β is constant.

2.9 Adaptive Resonance Theory(ART)

Adaptive resonance theory(ART) is an extension of the competitive-learning schemes(CPN) [19]. A key to solving the stability-plasticity dilemma is to add a feedback mechanism between the competitive layer and the input layer of a network. This feedback mechanism facilitates the learning of new information without destroying old information, automatic switching between stable and plastic modes, and stabilization of the encoding of the classes done by the nodes. The results from this approach are two neural network architectures that are particularly suited for the pattern classification problem in realistic environment. These network architectures are ART1 and ART2. ART1 and ART2 differ in the nature of their input patterns. ART1 networks require that the input vectors be binary . ART2 networks are suitable for processing analog or gray-scale patterns [20].

2.9.1 ART Network Description

Fig. 2.15 shows the basic features of the ART architecture. There are two types of memory,

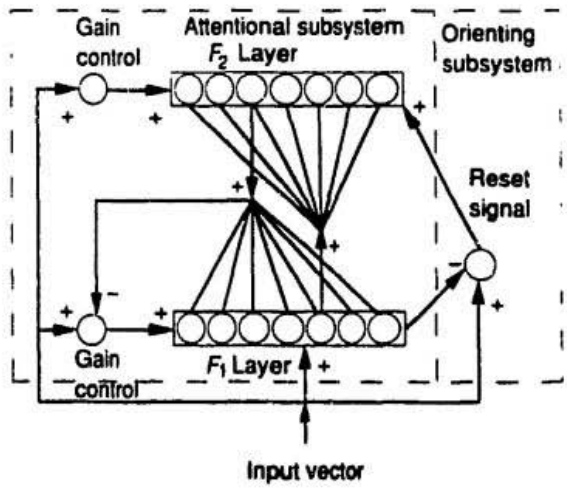

Figure 2.15: Model of ART system

1. short term memory(STM) that develops over the nodes in the two layers.

2. long term memory(LTM), top-down and bottom-up weight between F_1 and F_2 layers.

Pattern Matching in ART

The pattern matching cycle in ART can be defined by Fig. 2.16 [20].

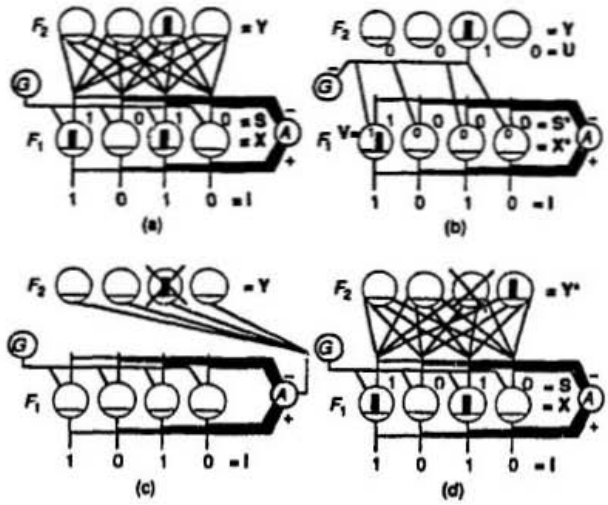

Figure 2.16: A pattern matching cycle in an ART. (a) pattern-matching attempt (b) reset in (c) final recognition (d) end of matching cycle

2.9.2 ART1

The input vector to ART1 is binary and it shares the common architecture of the ART. It's processing can summarized in the following points:

1. Input vector **I** is applied to F_1. F_1 activities are calculated as follows

$$x_{1i} = \frac{I_i}{1 + A_1(I_i + B_1) + C_1} \tag{2.42}$$

2. The output vector for F_1 is calculated as

$$s_i = h(x_{1i}) = \begin{cases} 1 & x_{1i} > 0 \\ 0 & x_{1i} \leq 0 \end{cases} \tag{2.43}$$

3. **S** is propagated forward to F_2 and the activities are calculated as

$$T_j = \sum_{i=1}^{M} s_i z_{ji} \tag{2.44}$$

4. Only the winning F_2 node has a nonzero output:

$$u_i = \begin{cases} 1 & T_j = max_k T_k \forall \\ 0 & otherwise \end{cases} \tag{2.45}$$

5. Output from F_2 is propagated back to F_1. Net inputs from F_2 on the units of F_1 are calculated as

$$V_i = \sum_{j=1}^{N} u_j z_{ij} \tag{2.46}$$

6. New activities are calculated as

$$x_{1i} = \frac{I_i + D_1 V_i - B_1}{1 + A_1(I_i + D_1 V_i) + C_1} \tag{2.47}$$

7. As in step 2 out on values s_i is calculated.

8. The degree of match between the input pattern and the top-down template is given by the following equation

$$\frac{\mid S \mid}{\mid I \mid} = \frac{\sum_{i=1}^{M} s_i}{\sum_{i=1}^{M} I_i} \tag{2.48}$$

9. If $\mid S \mid / \mid I \mid < \rho$, then v_J is marked as inactive, zero the outputs of F_2. and it is necessary to return to step1. If not then we have to continue.

10. Bottom-up weight has to be updated on v_J only

$$z_{Ji} = \begin{cases} \frac{L}{L-1+\mid s \mid} & if\ v_i\ is\ active \\ 0 & if\ v_i\ is\ inactive \end{cases} \tag{2.49}$$

11. Top-down weight is updated coming from v_J only to all $F1$ units

$$Z_{iJ} = \begin{cases} 1 & if\ v_i\ is\ active \\ 0 & if\ v_i\ is\ inactive \end{cases} \tag{2.50}$$

12. Input pattern should be removed. All inactive $F2$ units should restored

2.9.3 ART2

This neural network is similar to ART1. It only differs in the sense that its input pattern is analog signal.

2.10 Comparative analysis and selection of suitable Network

In case of incipient fault detection of induction motor based on spectral recognition, an ANN should have the following properties:

1. Low training time.
2. Ability to learn new knowledge while retaing the old one without any retraining of past pattern
3. Traning process should have certainity to reach global minima.
4. It should accept analog input pattern.
5. Input pattern should be in spatial domain
6. It should have ability to report if it can not classify a particular pattern.
7. It should be a general purpose ANN, so that necessary modifications can be done to make it suitable to the present problem domain.

To select a proper ANN for this purpose salient features of different ANNs have been summarized in the following Table. From the previous discussion it can be concluded that ART2 satisfies the necessary characteristics to be accepted as suitable neural network in incipent fault detection of an induction motor. Detailed software implementation and performance of ART2 are explained in the next chapter.

Table-2.1

Comparative Performances of Different ANN Techniques.

Type of ANN	Training time	Ability to learn new retaining old	Certainity to reach global minima	Type of input pattern	Domain input pattern	Ability to report if mitchmatch	Purpose
BPN	Ligh	No	No	Analog	Spatial	No	General
BAM	Low	No	No	Digital	Spatial	No	General
HM	Low	No	No	Digital	Spatial	No	General
BM	High	No	No	Digital	Spatial	No	General
CPN	Low	No	No	Digital	Spatial	No	General
SOM	Moderate	No	No	Analog	Spatial	No	General
STN	Moderate	No	No	Analog	Temporal	No	General
NC	Moderate	No	No	Analog	Spatial	No	Specheial
ART1	Low	Yes	Yes	Digital	Spatial	Yes	General
ART2	Low	Yes	Yes	Analog	Spatial	Yes	General

Chapter 3

ART2 Neural Network

3.1 Introduction

In response to the questions of dynamic updating and training time of conventional neural network, adaptive resonance theory(ART) has been proposed by Grossberg, Carpenter and others [19] . ART2 is a special version of ART having the property of analog input. In the implementation phase it has been modified to have an additional property of reporting if it cannot find a match for an input pattern.

3.2 ART2 Architecture

The structure of ART2 can be represented by Fig. 3.1. It consists of two subsystem known as *attentional subsystem(AS) and orienting subsystem(OS).* The AS consists of two layers of processing elements (PEs), $F1$ and $F2$ and

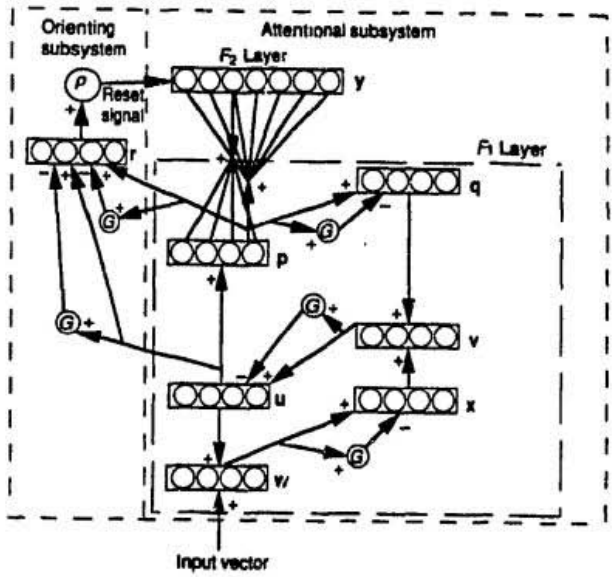

Figure 3.1: The overall structure of ART2

a gain-control system G.

3.2.1 The Attentional Subsystem

The activities of processing elements on the layers $F1$ and $F2$ can be defined by the following dynamic equation as

$$\varepsilon \dot{x}_k = -x_k + (1 - Ax_k)J_k^+ - (B + Cx_k)J_k^- \tag{3.1}$$

where J_k^+ and J_k^- are the excitory and inhibitory inputs to the kth unit, respectively. The precise definition of A, B *and* C depends upon the layer and for this case B and C have been considered to be zero. Here it has been considered that x_{1i} and x_{2j} refer to the activities on $F1$ and $F2$ layers, respectively. Here v_i represents the nodes on $F1$ and vj those on $F2$. The

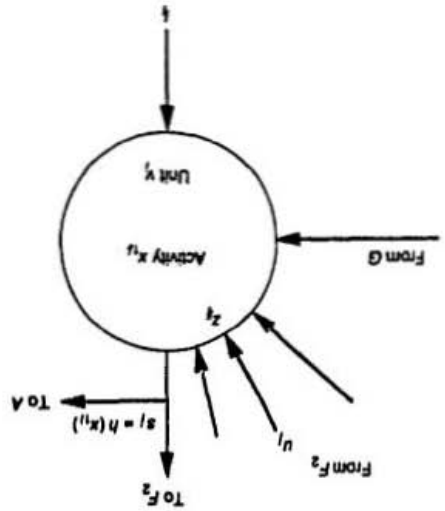

Figure 3.2: Structure of processing element on F_1 layer

constant ε determines how fast x_k reaches to equilibrium.

Processing on F1

A single processing element on $F1$ with its various inputs and weight vectors can be represented by Fig. 3.2 [19]. The units calculate a net-input value coming from $F2$ in the usual way:

$$V_i = \sum_i u_j z_{ij} \tag{3.2}$$

The values of individual quantities in the defining equations of $F1$ and $F2$ vary according to the sublayer being considered. For the sake of convenience, the appropriate values of the parameters for layer $F1$ have been

summarized in the Table 3.1. Based on the table, the activities on each of the six sublayers on $F1$ can be summarized by the following equations:

Table 3.1

Values of parameters on F-1 layer

			Quantity	
Layer	***A***	***D***	J_i^+	J_i^-
w	1	1	$I_i + au_i$	0
x	e	1	w_i	$\|\mathbf{w}\|$
u	e	1	v_i	$\|\mathbf{v}\|$
v	1	1	$f(x_i) + bf(q_i)$	0
p	1	1	$u_i + \sum_j g(y_j)z_{ij}$	0
q	e	1	p_i	$\|\mathbf{p}\|$
r	e	1	$u_i + cp_i$	$\|\mathbf{u}\| + c\mathbf{p}\|$

$$w_i = I_i + au_i \tag{3.3}$$

$$x_i = \frac{w_i}{e + \|\mathbf{w}\|} \tag{3.4}$$

$$v_i = f(x_i) + bf(q_i) \tag{3.5}$$

$$u_i = \frac{v_i}{e + \|\mathbf{v}\|} \tag{3.6}$$

$$p_i = u_i + \sum_j g(y_i)z_{ij} \tag{3.7}$$

$$q_i = \frac{p_i}{e + \|\mathbf{p}\|} \tag{3.8}$$

The form of the function $f(x)$ determines the nature of the contrast enhanchement that takes place on F_1. A sigmoid might be the logical choice for this function, but Carpenter's [20] choice is

$$f(x) = \begin{cases} 0 & 0 \leq x \leq \theta \\ x & x > \theta \end{cases} \tag{3.9}$$

where θ is a positive constant less than one.

Processing on F2

Fig. 3.3 shows a typical PE on F_2 layer. Bottom-up weights are calculated according to the following equation

$$T_j = \sum_i p_i z_{ji} \tag{3.10}$$

The output on $F2$ is given by the function

$$g(y_i) = \begin{cases} d & T_j = max_k T_k \ \forall K \\ 0 & otherwise \end{cases} \tag{3.11}$$

3.2.2 The Orienting Subsystem

From the parameter table and the defining equation of ART2, the actiuvites on the layer r on the orienting subsystem can be defined by

$$r_i = \frac{u_i + cp_i}{\|\mathbf{u}\| + \|c\mathbf{p}\|} \tag{3.12}$$

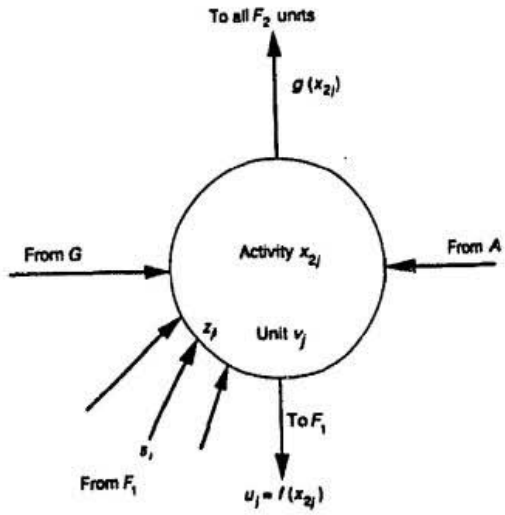

Figure 3.3: Structure of processing element on F_2 layer

Here it has been assumed that $e = 0$ and the condition for reset is

$$\frac{\rho}{\|\mathbf{r}\|} > 1 \tag{3.13}$$

It should be noted that two sublayers **p** and **u** participate in the matching process. As top-down weight changes on the **p** layer during learning, the activity of the units on the **p** layer also changes. The **u** layer remains stable during this process, so including it in the matching process prevents reset from occurring while learning of a new pattern is taking place.

3.2.3 Gain Control in ART2

The three gain control units on $F1$ nonspecifically inhibit the **x**,**u**, and **q** sublayers. The inhibitory signal is equal to the magnitude of the input vector

to those layers. The effect is that the activities of these three layers are normalized to unity by the gain control signals.

3.2.4 Least-mean-square Equations

Both bottom-up and top-down Least-mean-square equations have the same form as shown below:

$$\dot{z}_{ji} = g(y_i)(p_i - z_{ji}) \tag{3.14}$$

for the bottom-up weights from v_i on F_1 to v_j on F_2, and

$$\dot{z}_{ji} = g(y_i)(p_i - z_{ji}) \tag{3.15}$$

for top-down weights from v_j on F_2 to v_i on F_1. If v_J is winning node, then from the previous equations it can be shown that

$$\dot{z}_{Ji} = d(u_i + dz_{iJ} - z_{Ji}) \tag{3.16}$$

and similiarly

$$\dot{z}_{iJ} = d(u_i + dz_{iJ} - z_{iJ}) \tag{3.17}$$

with all other $\dot{z}_{ij} = \dot{z}_{ji} = 0$ *for* $j \neq J$. For the fast-learning case for the equilibrium values of the weights:

$$z_{Ji} = z_{iJ} = \frac{u_i}{1-d} \tag{3.18}$$

where it has assumed that $0 < d < 1$

3.2.5 Bottom-Up Least-mean-square Initialization

The bottom-up weight vectors can be initialized by the following relation:

$$\|\mathbf{z}\| < \frac{1}{1-d} \tag{3.19}$$

It is also possible to accomplish the initialization process by setting the weights to small random numbers. Alternatively, the initialization process can be performed also by the relation

$$z_{ji}(0) \leq \frac{1}{(1-d)\sqrt{M}} \tag{3.20}$$

3.2.6 ART2 Processing Summary

In this situation only the asymtotic solutions to the dynamic equations, and the fast-learning mode have been considered and it has also been considered that M be the number of units in each F_1 sublayer, and N be the number of units on F_2 . Parameters are chosen according to the relations $a, b > 0; 0 \leq d \leq 1; \frac{cd}{1-d} \leq 1; 0 \leq \theta \leq 1; 0 \leq \rho \leq 1; e << 1$.Top-down weights are initialized to zero $z_{ij} = 0$ and bottom-up weights are initialized according to

$$z_{ij}(0) \leq \frac{1}{(1-d)\sqrt{M}} \tag{3.21}$$

The processing can be now summarized in the following points [18]:

1. It is necessary to initialize all layer and sublayer outputs to zero vectors, and to establish a cycle counter to a value of one.

2. An input pattern **I** should be applied to the **w** layer on F_1. The output of this layer is

$$w_i = I_i + au_i \tag{3.22}$$

3. Forward propagation to the **x** sublayer should be done:

$$x_i = \frac{w_i}{e + \|\mathbf{w}\|} \tag{3.23}$$

4. Propagation forward to the **v** sublayer is done

$$v_i = f(x_i) + bf(q_i) \tag{3.24}$$

5. The result should be propagated to the **u** sublayer,

$$u_i = \frac{v_i}{e + \|\mathbf{v}\|} \tag{3.25}$$

6. Forward propagation to the **p** is done by

$$p_i = u_i + dz_{iJ} \tag{3.26}$$

where the Jth node on the F_2 is the winner of the competition on that layer. If F_2 is inactive, $p_i = u_i$.

7. It is necessary to propagate to the **q** sublayer

$$q_i = \frac{p_i}{e + \|\mathbf{p}\|} \tag{3.27}$$

8. Steps 2 through 7 should be repeated to stabilize the values on F_1

9. The output of r layer is calculated

$$r_i = \frac{u_i + cp_i}{e + \|\mathbf{u}\| + \|c\mathbf{p}\|} \tag{3.28}$$

10. Whether a reset condition is indicated should be determined. If $\rho/(e + \|\mathbf{r}\|) > 1$, then a reset signal to F_2 should be sent. Any active node on F_2 should be marked ineligible for the competetion and the cycle counter should be set to one and should be returned to step 2. If there is no reset, and the cycle counter is one, cycle counter should be incremented and continue with step 11. If there is no reset, and the cycle counter is greater than one, then it necessary to skip to step 14, as resonance has been established.

11. The output of the p sublayer should be propagated to the F_2 layer and the net inputs to F_2

$$T_j = \sum_{i=1}^{M} p_i z_{ji} \tag{3.29}$$

12. Only the winning F2 node has nonzero output.

$$g(T_j) = \begin{cases} d & T_j = max_k T_k \\ 0 & otherwise \end{cases} \tag{3.30}$$

13. Step 6 through 10 should be repeated.

14. Bottom-up weights on F_2 unit should be modified

$$z_{Ji} = \frac{u_i}{1 - d} \tag{3.31}$$

15. Top-down weights from the winning F_2 unit should be modified

$$z_{iJ} = \frac{u_i}{1 - d} \tag{3.32}$$

16. Input vector should be removed and all inactive F_2 units should be restored and now it is the time to return to step 1 with new input pattern.

3.3 ART2 Simulator

ART2 has been implemeted using *Object Oriented* software development methodology. It has been considered as an independent class structure so that an instance of this class can be easily used to an application software like on-line condition monitoring system of induction motor.

3.3.1 Model of ART2 as an Object

The model of ART2 as an object can be shown in Fig. 3.4. Salient features of this object are pattern encoding and decoding through training and recalling, dynamic addition and deletion of neurons in the network. Fig. 3.5 shows the flow chart of training algorithm of ART2.The detailed program listing is given in Appendix-A.

3.3.2 Modified Structure of Training and Recalling Pattern in the Network

To add the property to give a unmatched signal if it cannot find a pattern in the network and also to to make the training process faster, the pattern encoding and decoding technique has been slightly modified as shown in Fig. 3.6.

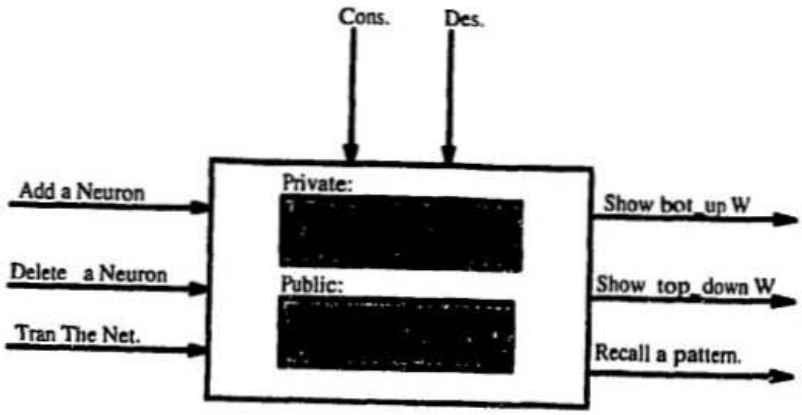

Figure 3.4: Model of ART2 as an object

3.3.3 Dynamic Updating

In the object model of the network, it should be noted that there are provisions for addition and deletion of nodes. So, the structure of the ANN is not determined by the initial parameters. Due to this advantage, ART2 type neural network is suitable for a dynamic scenario like incipient fault detection, which does not require retraining of the already trained patterns. Moreover, using this dynamic property it is possible to make optimum use of the computing resources.

3.4 Experimental Varification of Performance

For the experimental purpose, the number of neurons in layer F_2 and F_1 have been considereset to 6 and 9, respectively. The following parameters have been initialized with specified values:
$a = 10; b = 0.10; c = 0.08; \theta = 0.2; e = 0.0; \rho = 0.995$. Initial bottom-up weight=2.08, initial top-down weight=0.0.

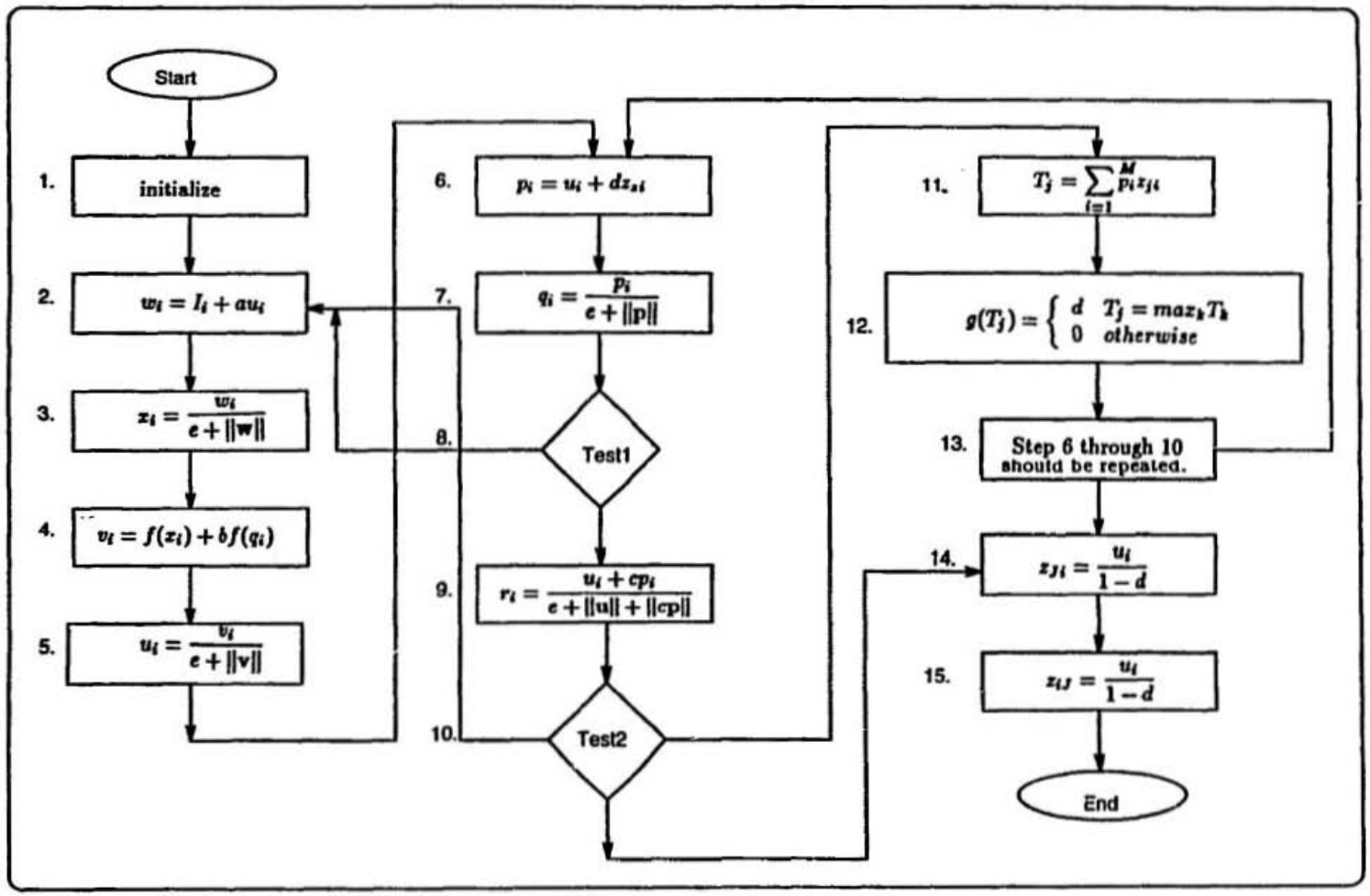

Figure 3.5: Flow chart of training algorithm of ART2

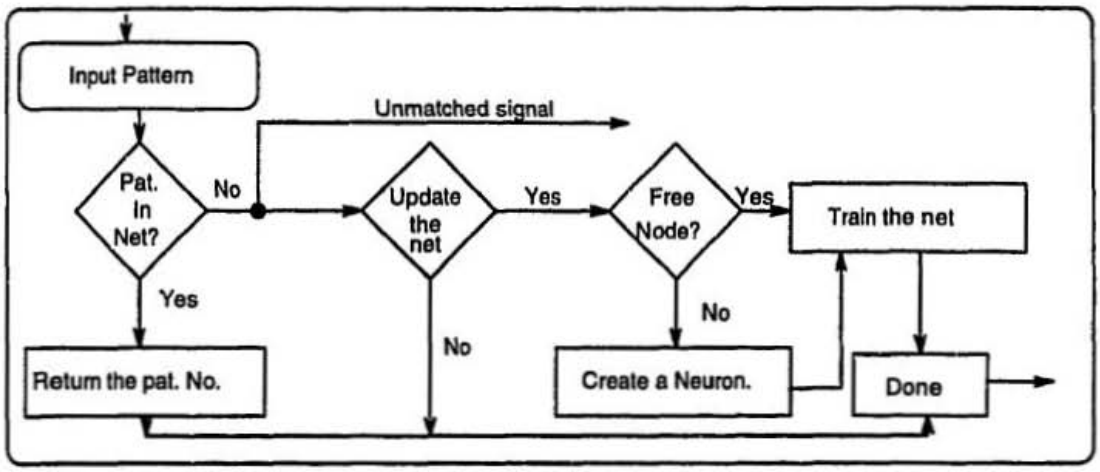

Figure 3.6: Modified Training and Recall Algorithm of ART2

3.4.1 Training of the Neural Network with test Pattern

Pattern of the exemplars and corresponding training results are shown in Table 3.2. Table 3.3 and Table 3.4 represent the bottom-up and top-down weight matrices, respectively. In the training session it should be noted that the network has stored both the patterns 2 and 3 in the same class, though the vectors are seperated by euclidian distance $\sqrt{(65-60)^2+(15-13)^2} = 5.38$. This type of behaviour is suitable to classify like patterns in same class to realize fuzzy nature of the problem. But when two vectors are far apart as in case of patterns 3 and 4, the network classifies them in seperate classes.

Table 3.2

Set of Training examples Used to Train ART2.

No.	Pattern:	Training Result.
1.	60.0 0.0 20.0 11.0 20.0 0.0 10.0 0.0 18.0	stored in class 0
2.	60.0 16.0 31.0 0.0 17.0 0.0 0.0 0.0 13.0	stored in class 1
3.	65.0 16.0 31.0 0.0 17.0 0.0 0.0 0.0 15.0	stored in class 1
4.	50.0 11.0 31.0 0.0 0.0 0.0 23.0 0.0 12.0	stored in class 2
5.	50.0 0.0 25.0 0.0 22.0 30.0 20.0 0.0 18.0	stored in class 3
6.	67.0 25.0 44.0 15.0 0.0 10.0 11.0 0.0 17.0	stored in class 4
7.	68.0 26.0 48.0 21.0 20.0 0.0 0.0 10.0 0.0	stored in class 5
8.	17.0 27.0 42.0 16.0 27.0 20.0 40.0 0.0 13.0	No more neuron

Table 3.3

Bottom-up weight matrix after training.

10.19	10.50	9.89	8.64	9.97	9.25
0.0	2.80	0.0	0.0	3.72	3.54
3.64	5.42	6.13	4.32	6.55	6.53
0.0	0.0	0.0	0.0	0.0	2.86
3.64	2.97	0.0	3.80	0.0	2.72
0.0	0.0	0.0	5.18	0.0	0.0
0.0	0.0	4.55	3.46	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0
3.27	0.0	0.0	3.11	0.0	0.0

Table 3.4

Top-down weight matrix after training.

10.91	0.0	3.64	0.0	3.64	0.0	0.0	0.0	3.27
10.50	2.80	5.42	0.0	2.97	0.0	0.0	0.0	0.0
9.89	0.0	6.13	0.0	0.0	0.0	4.55	0.0	0.0
8.64	0.0	4.32	0.0	3.80	5.18	3.46	0.0	3.11
9.97	3.72	6.55	0.0	0.0	0.0	0.0	0.0	0.0
9.25	3.54	6.53	2.86	2.72	0.0	0.0	0.0	0.0

3.4.2 Dynamic Neuron Addition

As number of neurons in the output layer F_2 is 6, so it can classify only six classes of patterns. Now in case of a dynamic system the network should have the ability to update it to handle this new situation. Using Add-Neuron behaviour of the network this can be accomplished as shown in Table 3.5 after addition of a new neuron. No retraining of previous patterns is necessary. Now, after the addition of last pattern new bottom-up and top-down weight matrices are given in Table 3.6 and Table 3.7, respectively.

Table 3.5

New training vector to train after neuron addition.

No.	Pattern:	Training Result.
8.	17.0 27.0 42.0 16.0 27.0 20.0 40.0 0.0 13.0	stored in class 6

Table 3.6

Bottom-up weight matrix after training.

10.19	10.50	9.89	8.64	9.97	9.25	2.80
0.0	2.80	0.0	0.0	3.72	3.54	4.44
3.64	5.42	6.13	4.32	6.55	6.53	6.91
0.0	0.0	0.0	0.0	0.0	2.86	2.63
3.64	2.97	0.0	3.80	0.0	2.72	4.44
0.0	0.0	0.0	5.18	0.0	0.0	3.29
0.0	0.0	4.55	3.46	0.0	0.0	6.58
0.0	0.0	0.0	0.0	0.0	0.0	0.0
3.27	0.0	0.0	3.11	0.0	0.0	0.0

Table 3.7

Top-down weight matrix after training.

10.91	0.0	3.64	0.0	3.64	0.0	0.0	0.0	3.27
10.50	2.80	5.42	0.0	2.97	0.0	0.0	0.0	0.0
9.89	0.0	6.13	0.0	0.0	0.0	4.55	0.0	0.0
8.64	0.0	4.32	0.0	3.80	5.18	3.46	0.0	3.11
9.97	3.72	6.55	0.0	0.0	0.0	0.0	0.0	0.0
9.25	3.54	6.53	2.86	2.72	0.0	0.0	0.0	0.0
2.80	4.44	6.91	2.63	4.44	3.29	6.58	0.0	0.0

3.4.3 Pattern Recall from the Network

The performances of the network in pattern matching are given in Table 3.8. From Table 3.8, pertinent information can be highlighted. In case of pattern 1 ,the euclidian distance of the recall pattern from the closest trained pattern is $\sqrt{(5.0-0.0)^2+(15-10)^2} = 7.07$, but the network has classified it as member of class 0. But when the distance is very high as in case of patterns 5 and 7, the network has reported that it is unable to classify, instead of giving some false classification result. This is one of the most important behaviours from the neural network for the diagnostic purpose of incipient faults of an induction motor.

Table 3.8

Pattern matching result of the trained network.

No.	Pattern:	Training Result.
1.	60.0 5.0 20.0 11.0 20.0 0.0 15.0 0.0 18.0	classified as class 0
2.	60.0 16.0 31.0 0.0 17.0 10.0 0.0 0.0 13.0	classified as class 1
3.	65.0 16.0 31.0 0.0 17.0 0.0 0.0 0.0 15.0	classified as class 1
4.	50.0 11.0 31.0 0.0 0.0 0.0 23.0 0.0 12.0	classified as class 2
5.	50.0 0.0 25.0 0.0 22.0 0.0 110.00.0 18.0	unable to classify
6.	67.0 25.0 44.0 15.0 0.0 10.0 11.0 0.0 17.0	classified as class 4
7.	68.0 26.0 8.0 21.0 20.0 0.0 0.0 10.0 0.0	unable to classify
8.	17.0 27.0 42.0 16.0 27.0 20.0 40.0 0.0 13.0	classified as class 6

Chapter 4

Fault Related Information Collection

4.1 Introduction

From study it has been noticed that the waveform of the stator current of an induction motor carries the signature of internal status of the machine [15]. The frequency spectra of the stator current can be considered as the information carrier of the incipient faults [13].

4.2 Model of Spectra Collection

The input current to an induction machine is an analog signal. But due to the rapid development of digital signal processing(DSP) techniques, it is comparatively easy now to use these tools to get the frequency spectra of the stator current. The model of spectra collection can be explained by the Fig. 4.1

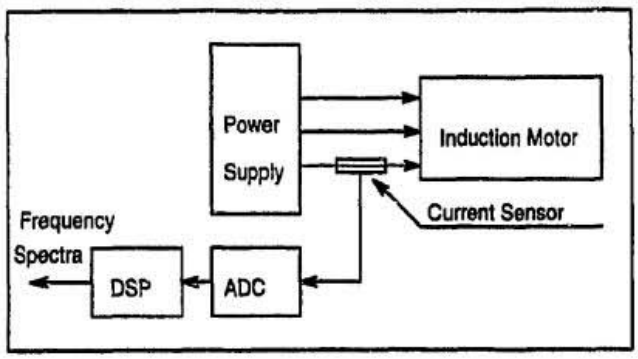

Figure 4.1: Model of Fault Related Spectra Collection

4.3 Discrete Time Signals and Systems

A signal can be defined as a function that conveys the information, generally about the state or behaviour of a physical system [21].The independent variable in the mathematical representation of a signal may be either continuous or discrete. Continuous-time signals are defined along a continuum of time and thus are represented by a continuous independent variable. Continuous time signals are often referred to as analog signals. Discrete-time signals are defined at discrete times and thus the independent variable has discrete values. Digital signals are those for which both time and amplitude are discrete.

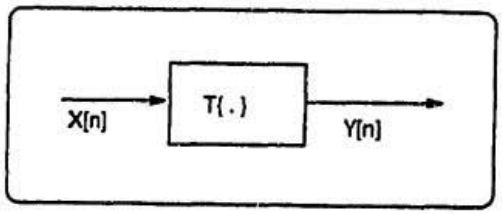

Figure 4.2: Representation of discrete-time system

4.3.1 Discrete-Time Signals: Sequences

Discrete-time signals are represented mathematically as sequences of numbers. A sequence of numbers x, in which the nth number in the sequence is denoted by $x[n]$, is formally written as

$$x = x[n], -\infty < n < \infty \tag{4.1}$$

where n is an integer.

4.3.2 Discrete-Time Systems

A discrete-time system is defined mathematically as a transformation or operator that maps an input sequence with values $x[n]$ into an output sequence with values $y[n]$ [21]. This can be denoted as

$$y[n] = Tx[n] \tag{4.2}$$

and is indicated pictorially in Fig. 4.2.

4.3.3 Sampling of Continuous-Time Signals

Typical method of obtaining a discrete-time signal representation of a continuous-time signal is through periodic sampling, wherein a sequence $x[n]$ is obtained from a continuous-time signal $x_c(t)$ according to the relation

$$x[n] = x_c(nT), -\infty, n, \infty \tag{4.3}$$

where T is the sampling period, and its reciprocal, $f_s = 1/T$, is the sampling frequency, in samples per second.

Nyquist Sampling Theorem:

Let $x_c(t)$ be a band-limited signal with

$$X_c(j\Omega) = 0 \quad for \quad |\Omega| > \Omega_N \tag{4.4}$$

Then $x_c(t)$ is uniquely determined by its samples
$x[n] = x_c(nT), n = 0, \pm 1, \pm 2, ..., if$

$$\Omega_s = \frac{2\pi}{T} > 2\Omega_N \tag{4.5}$$

The frequency Ω_N is commonly referred as the Nyquist frequency, and the frequency $2\Omega_N$ that must be exceeded by the sampling frequency is called the Nyquist rate.

4.3.4 The Discrete Fourier Transform(DFT)

For a finite length sequence $x[n]$ of length N generally the DFT analysis and synthesis equations are written as [21]

$$Analysis\ \ equation:\ X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn},\ \ k = 0, 1, ..., N-1 \quad (4.6)$$

$$Synthesis\ \ equation:\ x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn},\ \ n = 0, 1, ..., N-1 \quad (4.7)$$

Here $W_N = e^{-j(2\pi/N)}$

4.3.5 Computation of the Discrete Fourier Transform

The DFT is an important component in many practical applications of discrete-time systems. To make the DFT computation faster a number of efficient algorithms have been developed collectively known as fast Fourier transform(FFT).

Decimation-In-Time FFT Algorithm

Algorithms in which the decomposition is based on decomposing the sequences $x[n]$ into successively smaller subsequences, are called decimination-in-time algorithms. An $(N/2)$ point DFT can be represented by

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r]W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1]W_{N/2}^{rk} \quad (4.8)$$

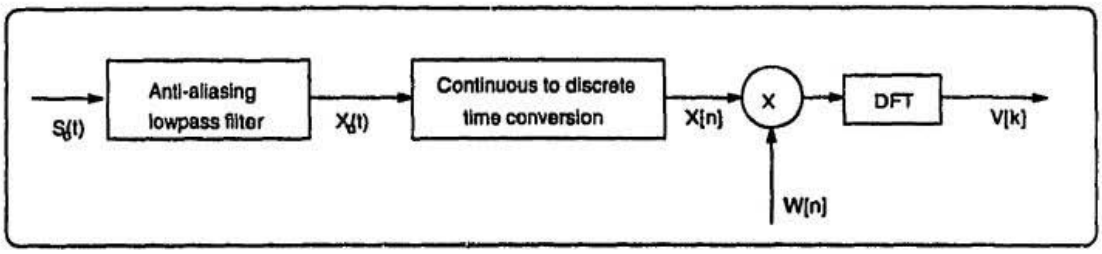

Figure 4.3: Processing steps in the discrete-time Fourier analysis of a continuous- time signal

Decimation-In-Frequency FFT Algorithm

For a sequence $x[n]$ the output sequence $X[k]$ can be divided into smaller and smaller subsequences in the same manner. FFT algorithms based on this procedure are commonly called decimation-in-frequency algorithm. The $N/2$ point DFT of the $N/2$-point sequence can be represented by

$$X[2r] = \sum_{n=0}^{(N/2)-1} (x[n] + x[n + (N/2)])W_{N/2}^{rn}, r = 0, 1, 2, ..., (N/2) - 1 \quad (4.9)$$

4.3.6 Fourier Analysis of Signals Using the Discrete Fourier Transform

The consistency between the finite-length requirement of the DFT and the reality of indefinitely long signals can be accommodated exactly or approximately through the concepts of windowing, block processing, and the time-dependent Fourier transform. The processing steps in the discrete Fourier analysis of a continuous time signal can be represented by Fig. 4.3. Fig. 4.4 is the illustration of the Fourier transforms .

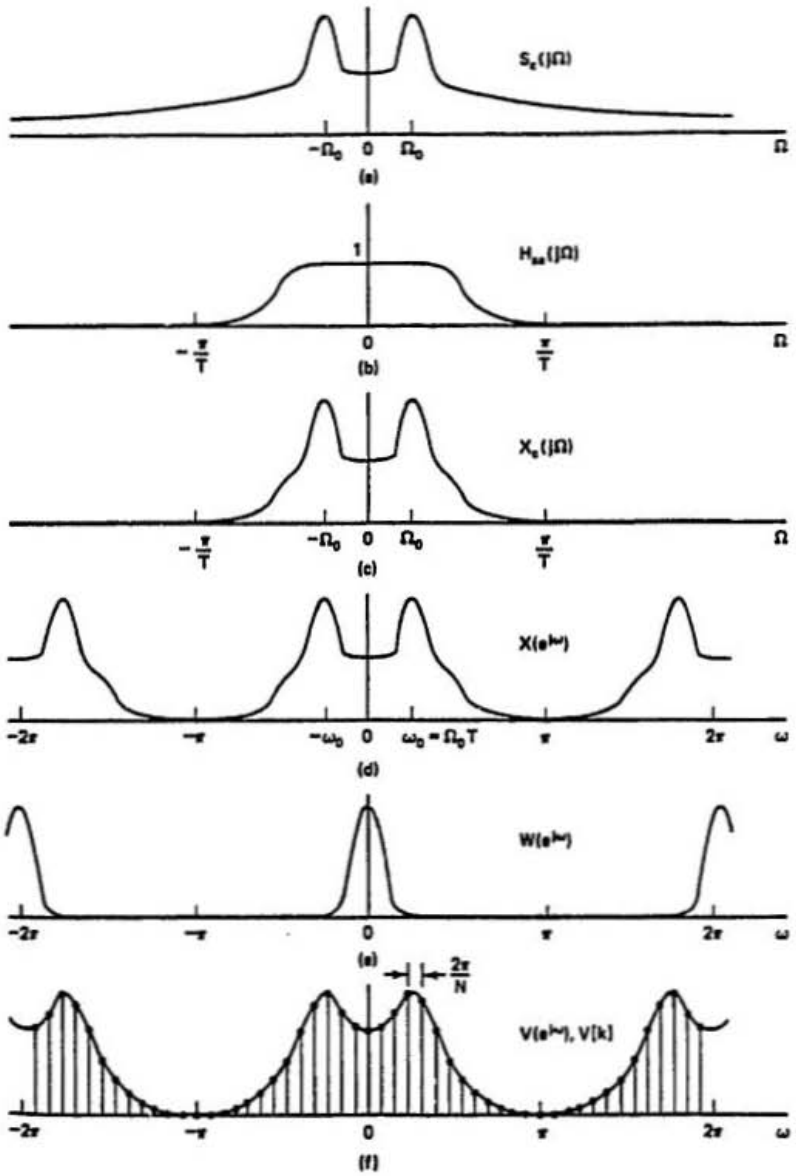

Figure 4.4: Illustration of the Fourier transforms in the system : (a) Fourier transform of continuous-time input signal. (b) Frequency response of anti aliasing filter. (c) Fourier transform output of anti aliasing filter. (d) Fourier transform of sampled signal (e) Fourier transform of window sequence (f) Fourier transform of windowed signal segment and frequency samples obtained using DFT samples

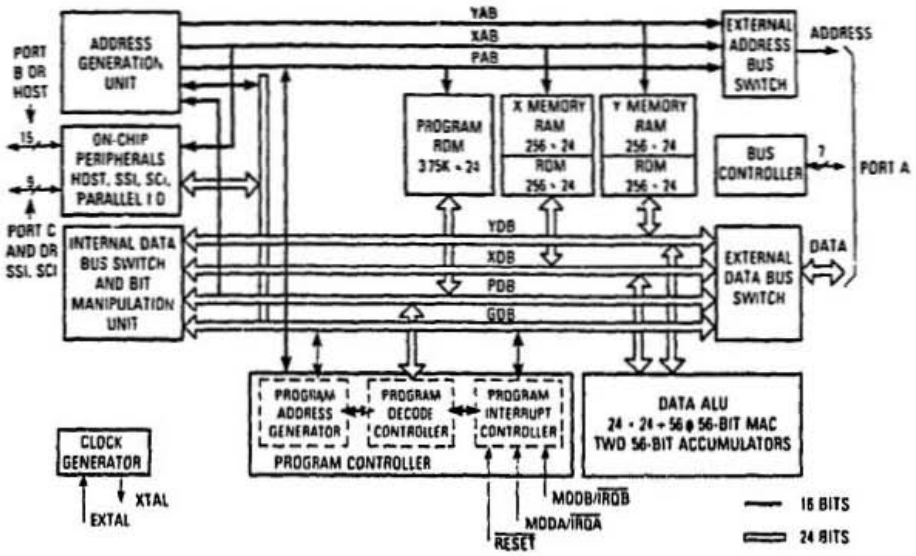

Figure 4.5: DSP56000 Block Diagram

4.4 Motorola's DSP56000 DSP Family

The DSP56000 and DSP56001 are the two members of Motorola's Family of HCMOS, low-power, general-purpose DSPs [22]. Block diagram of the DSP56000 is shown in Fig. 4.5. The DSP56001 features 512 words of full-speed, on-chip, program RAM, two preprogrammed data ROMs, and special on-chip bootstrap hardware to permit convenient loading of user programs into the program RAM. The DSP56001 is an off-the-shelf item since there are no user-programmable on-chip ROMs. The DSP56000 features 3.75k words of full speed, on-chip, program ROM instead of 512 words of program RAM.

The central part of the processor consists of three execution units operating in parallel:

1. the data arithmetic logic unit(ALU)

2. the address generation unit(AGU)

3. the program controller

The DSP56000/DSP56001 has microcontroller unit (MCU)-style on-chip peripherals, program memory, data memory, and a memory expansion port. The microprocessor unit (MPU)-style programming model and instruction set allow straightforward generation of efficient and compact code. The high throughput of the DSP56000/DSP56001 makes it well-suited for communication, high-speed control, numeric processing, computer applications, and audio applications. The main features making this throughput are as follows:

- **Speed:** At 1025-million instruction per second(MIP), the DSP56000/DSP56001 can execute a 1024-point complex FFT in 3.23 ms.

- **Precision:** The data paths are 24 bits wide, providing 144 dB of dynamic range; intermediate results held in the 56-bit accumulators can range over 336 dB.

- **Parallelism:** Each on-chip execution unit(AGU, program controller,data ALU), memory, and peripheral operates independently and in parallel with the other units through a sophisticated bus system. The data ALU, AGUs and program controller operate in parallel so that an instruction prefetch, a 24-bitx24-bit multiplication, a 56-bit addition, two data moves, and two address pointer updates using one of the three types of arithmetic can be executed in a single instruction cycle. This parallelism allows a four-coefficient infinite impulse response(IIR) filter

section to be executed in only four cycles, the theoretical minimum for single-multiplier architecture. At the same time, two serial controllers can send and receive full-duplex data, and the host port can send/receive simplex data.

- **Integration:** In addition to the three independent execution units, the DSP56000/DSP56001 has six on-chip memories, three on-chip MCU-style peripherals (serial communication interface(SCI), synchronous serial interface(SSI), and host interface), a clock generator , and seven buses (three address and four data), making the overall system low cost, low power, and compact.

- **Invisible Pipeline:** The three-stage instruction pipeline is essentially invisible to the programmer, allowing straightforward program development in either assembly language or a high-level language.

- **Instruction Set:** The 62 instruction mnemonics are MCU-like, making the transition from programming microprocessor to programming the DSP56000/DSP56001 as easy as possible. The orthogonal syntax supports controlling the parallel execution units. The hardware Do Loop instruction and the repeat instruction make writing straight line code obsolete.

- **DSP56000/DSP56001 Compatibility:** The DSP56000 is identical to the DSP56001 except for the following features:
  – 512-word x 24-bit, on-chip program RAM instead of 3.75k program ROM.

- 32-word x 24-bit bootstrap ROM for loading the program RAM from either a byte wide, memory-mapped ROM or via the host interface.

- **Low Power:** As a CMOS part, the DSP56000/DSP56001 is inherently very low power; however, three other features can reduce power consumption to exceptionally low levels.
  - The WAIT instruction shuts off the clock in the internal processor portion of the DSP56000/DSP56001.
  - The STOP instruction halts the internal oscillator.
  - Power increases linearly with frequency; thus, reducing the clock frequency reduces power consumption.

Fig. 4.6 is a block diagram of DSP56001 based board PC-56 [23]. The main subsystems are as follows:

1. The DSP56001 processor.

2. External 16k data RAM.

3. Parallel data port that is between the PC and the board PC-56.

4. Single channel analog interface.

5. Interrupt-driven interface to the outside world.

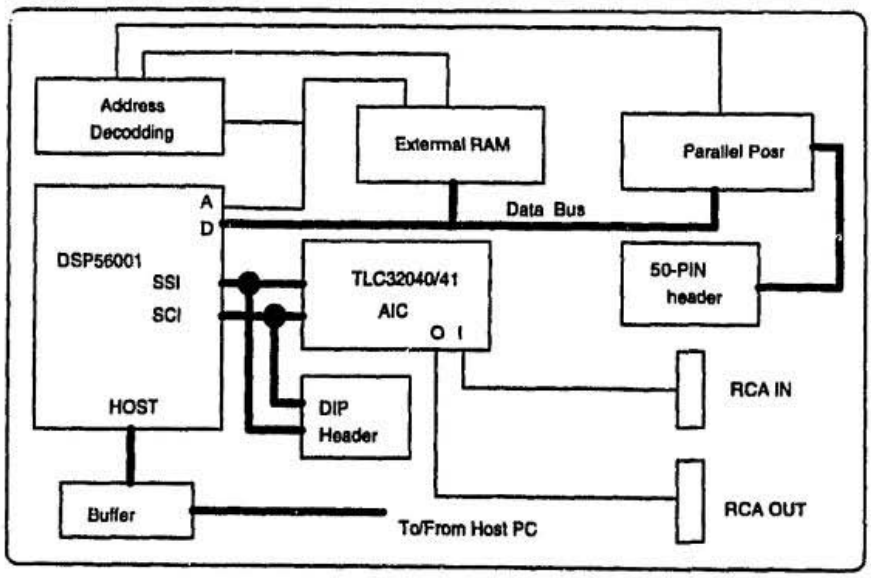

Figure 4.6: PC-56 Block Diagram

4.5 Ariel's Interface and DSP Library

A good set of routines for communication of the board with the PC and digital signal processing are provided by Ariel Corporation [24] in a C call-able software library. A brief description of the library is given in the subsequent sections:

4.5.1 Interface Library

The PC-56 interface software provides a means to configure and control the PC-56 for user applications through the use of C call-able routines [24]. The package consists of both 56000 and PC based routines, which when used together allow straightforward control over a user's 56000 application program.

Ariel Monitor(ARIELMON) is the required 56000 monitor. It is a super-

- forward and inverse FFT,
- magnitude, phase, dB computation (log),
- windowing, scaling, and data acquisition.

Processing of data consists of executing one or more of these commands in a given sequence.

4.6 Application Software for Spectra Collection

With the proper use of interface and DSP library, an application program has been developed as part of this work to collect frequency spectra of stator current at different fault conditions. The salient features of algorithm is shown in flowchart form in Fig. 4.7. The detailed program listing is given in Appendix-B and Appendix-C.

4.7 Experimental Setup

To acquire different faults related spectra of the stator current of an induction motor following equipment were used:

- Induction Motor.
- Current Probe.
- DSP board

set of debug monitor(DEGMON), and provides the means for communicating with the PC in the form of up-load/down-load of program and data memory, as well as user program initialization and termination. It is tied specifically to the PC routines, which are compatible with Microsoft C V5.1 and Quick C. Other Microsoft languages can use these routines by using the appropriate declarations for setting up the proper C calling convention.

4.5.2 DSP-Library

The FFT_{56} is a software package which enables the PC-56 or DSP-56 boards to function as FFT co-processors in an IBM PC/XT/AT or compatible computer. In addition to processing data from the PC, it also supports real time data acquisition using the analog interfaces available on the board.

User has access to the FFT_{56} functions by using low level drivers in conjunction with a set of 56000 programs and data files. The general purpose drivers perform basic functions necessary to control the 56000 and up-load/down-load data to the co-processor board.

56000 software consists of a monitor program, which is loaded during the boot cycle, and an application program, loaded by the monitor under control of the PC. The monitor provides the means for communicating with the PC (i.e. up-load/down-load of program and data memory) and is tied specifically to the PC driver routines. The application program consists of a set of FFT related functions which can be initiated by the PC through Host commands. These include:

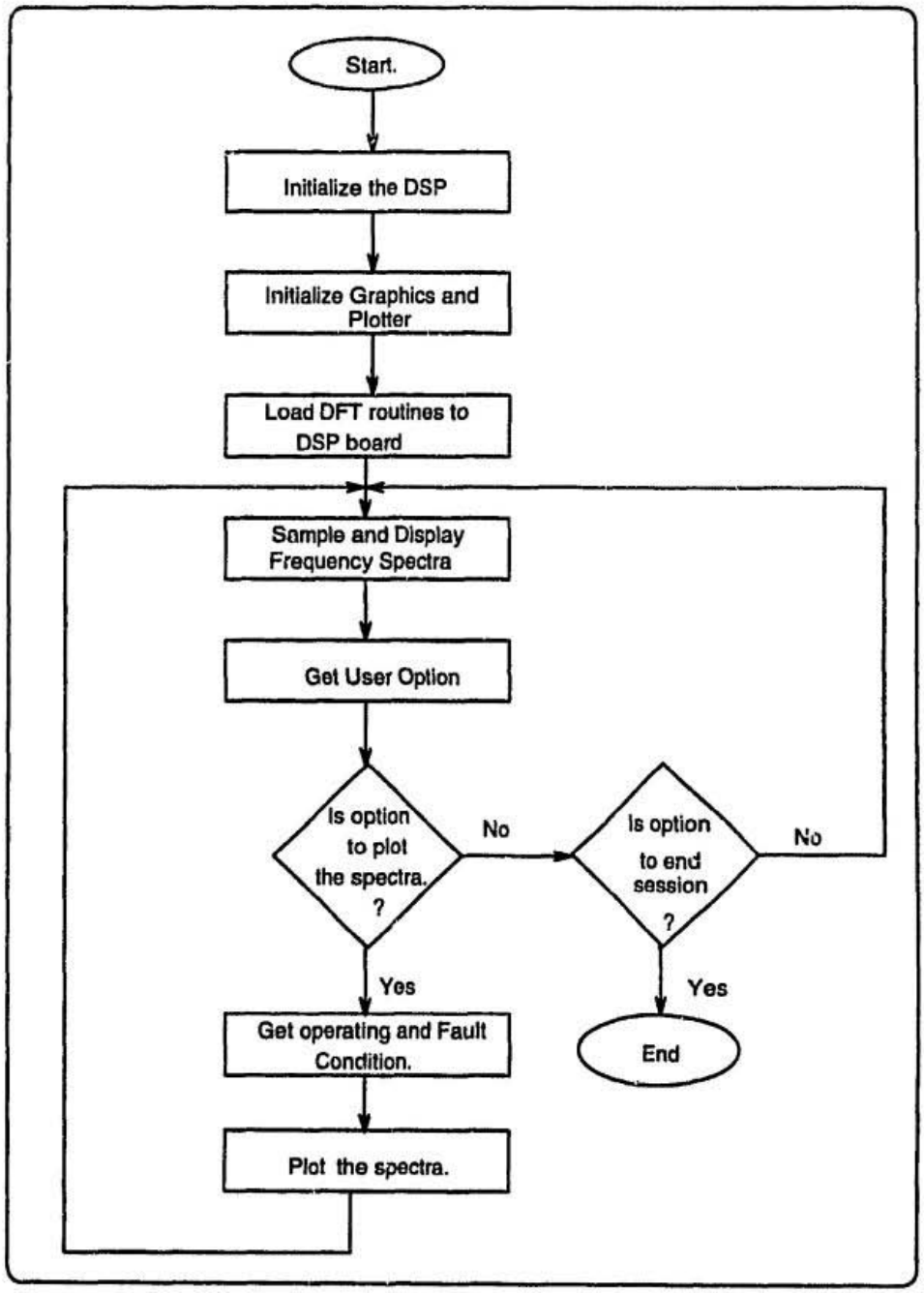

Figure 4.7: Flowchart of the application program

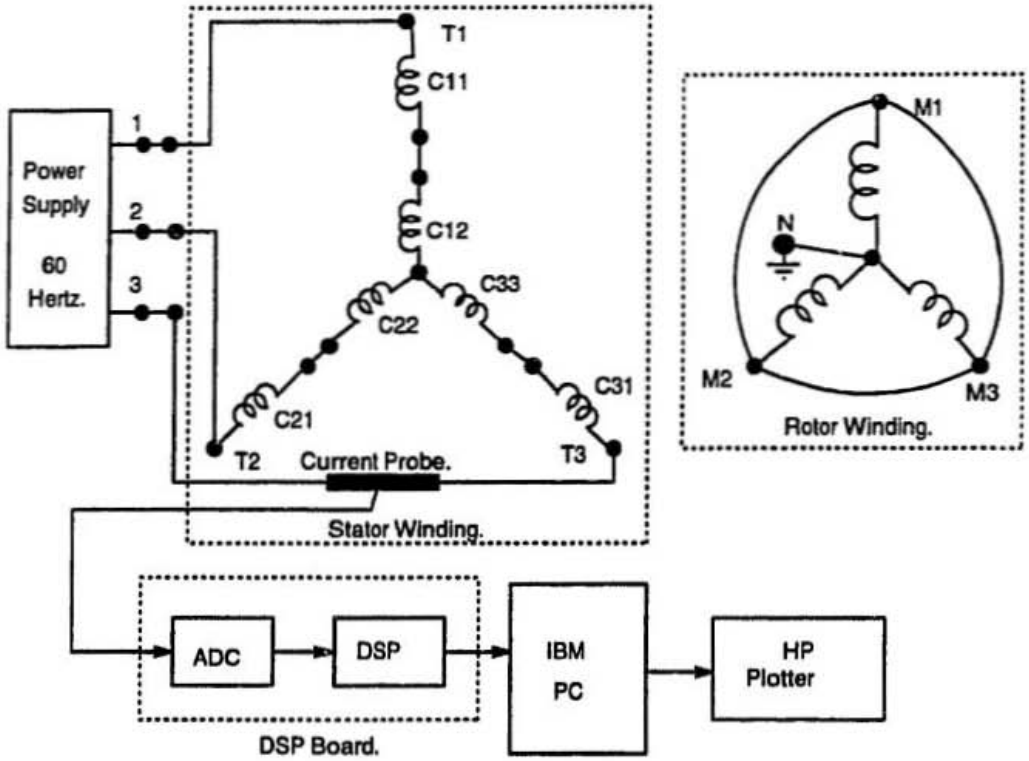

Figure 4.8: Block diagram of the experimental setup.

- IBM PC
- HP-plotter

The experimental setup is shown in Fig. 4.8. In this figure C11, C12, C21, C22, C23 and C33 are individual coils in the stator winding.

4.8 Spectra for Different Fault Conditions

In the laboratory, different types of faults have been occurred in an induction machine. The laboratory wound-rotor induction motor has the following specification:

Specification:

- **Phase:** Three
- **Voltage:** $\frac{120/208}{220/380}$ volts
- **Frequency:** 60/50 Hz
- **Speed:** 1725/1425 r.p.m
- **Full-load Currents:** $\frac{19/11}{8.7/1.5}$ A
- **Horse Power:** 2.5/2 H.P.
- **Poles :** 4

The motor was operated at 208v and 60 Hz at no-load condition. Spectra related to different fault currents are shown in the subsequent sections.

4.8.1 No Fault condition:

This is the operating condition which is identical to the set-up as shown in Fig. 4.8. The frequency spectra of the stator current is shown below. Here, in addition to the fundamental component there are also three higher frequencies as the stator current is not perfectly sinusoidal.

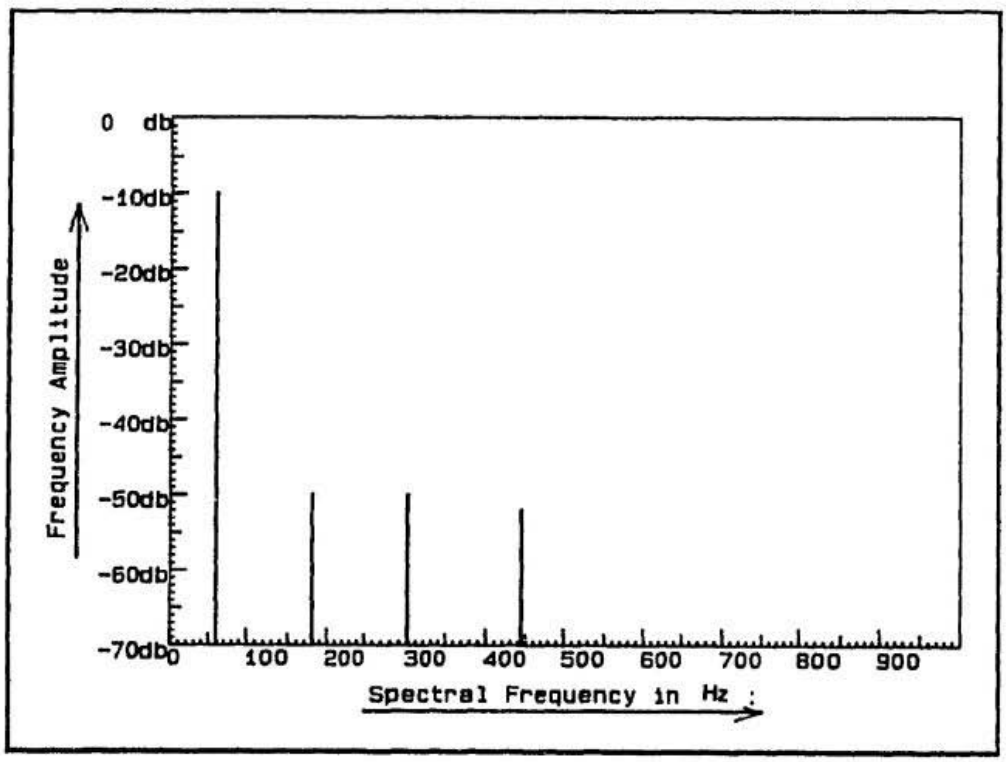

Figure 4.9: Frequency spectra at no fault condition.

4.8.2 Stator phase 1 is open:

Under this operating condition, the phase 1 of the stator has been disconnected from the power supply. The corresponding current spectra is shown below.

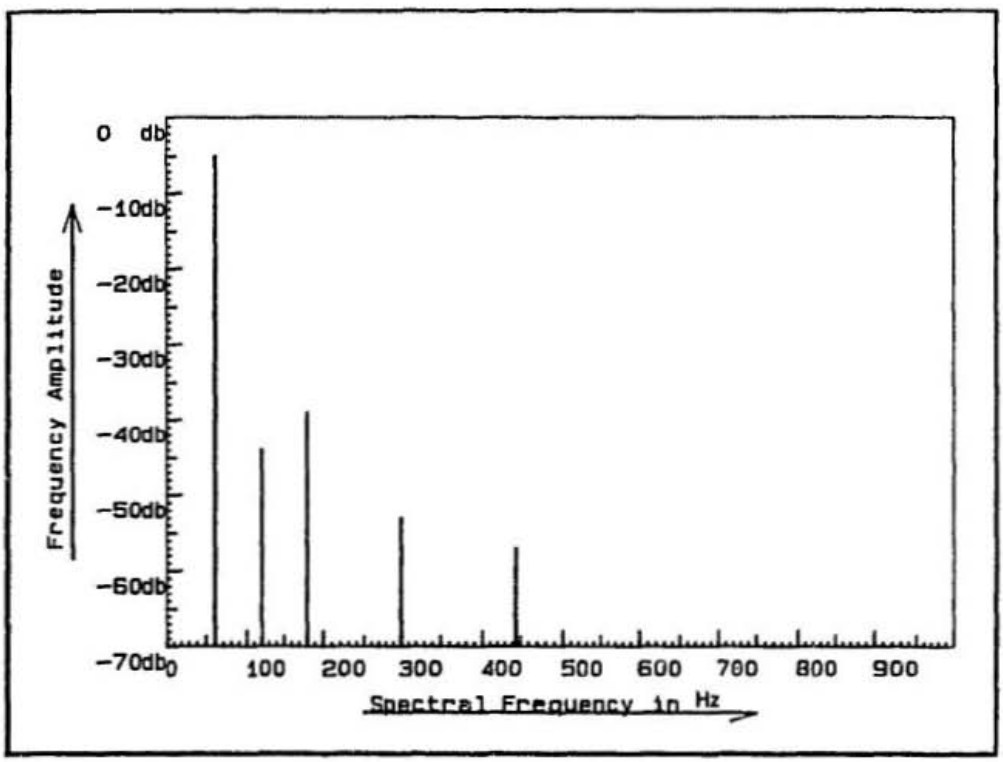

Figure 4.10: Frequency spectra when stator phase 1 is open.

4.8.3 Stator phase 2 is open:

To do it in the Laboratory the phase 2 of the stator was disconnected from the power supply. The corresponding current spectra is shown below.

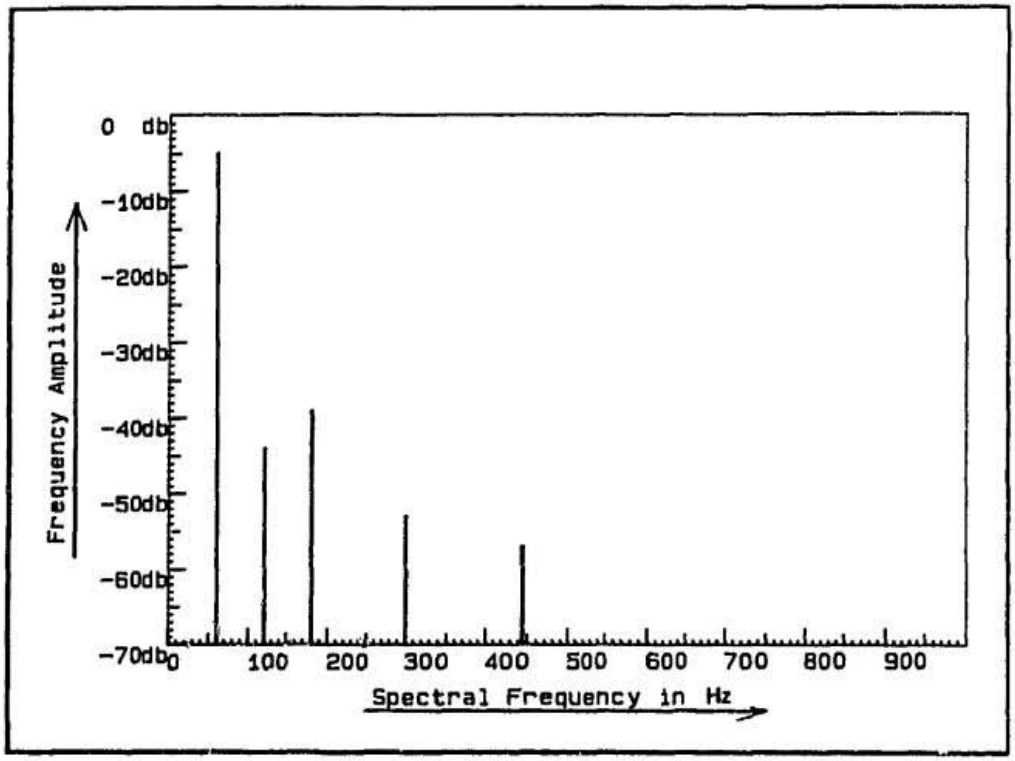

Figure 4.11: Frequency spectra when stator phase 2 is open.

4.8.4 Stator phase 3 is open:

Under this condition as current probe is in phase 3 as shown in Fig. 4.8, every frequency component has zero value .

4.8.5 Short circuit fault through resistance in Stator Phase 1:

This is the condition when coil C11 as shown in Fig. 4.8 was disconnected and a resistor of value 9.5 ohm was connected externally in its position. The corresponding frequency spectra is shown below.

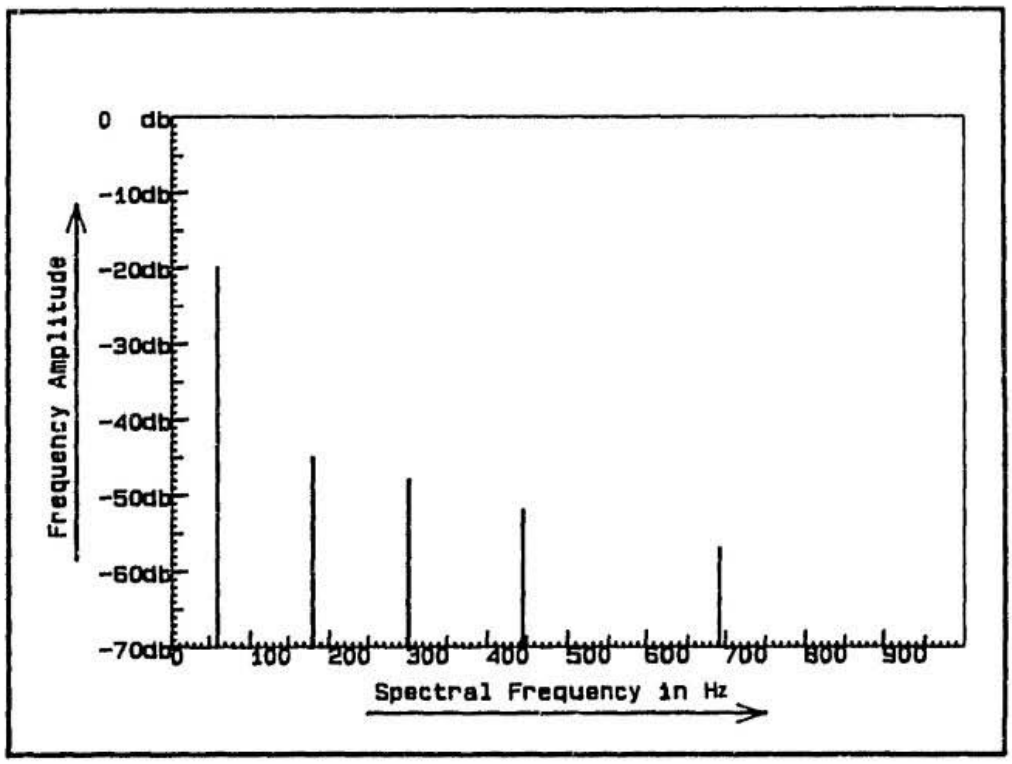

Figure 4.12: Frequency spectra when stator one coil of phase 1. has been externally replaced by a resistor.

4.8.6 Short circuit fault through resistance in Stator Phase 2:

Under this condition coil C21 as shown in Fig. 4.8 was disconnected and a resistor of value 9.5 ohm was connected externally in it's position. The corresponding frequency spectra is shown below.

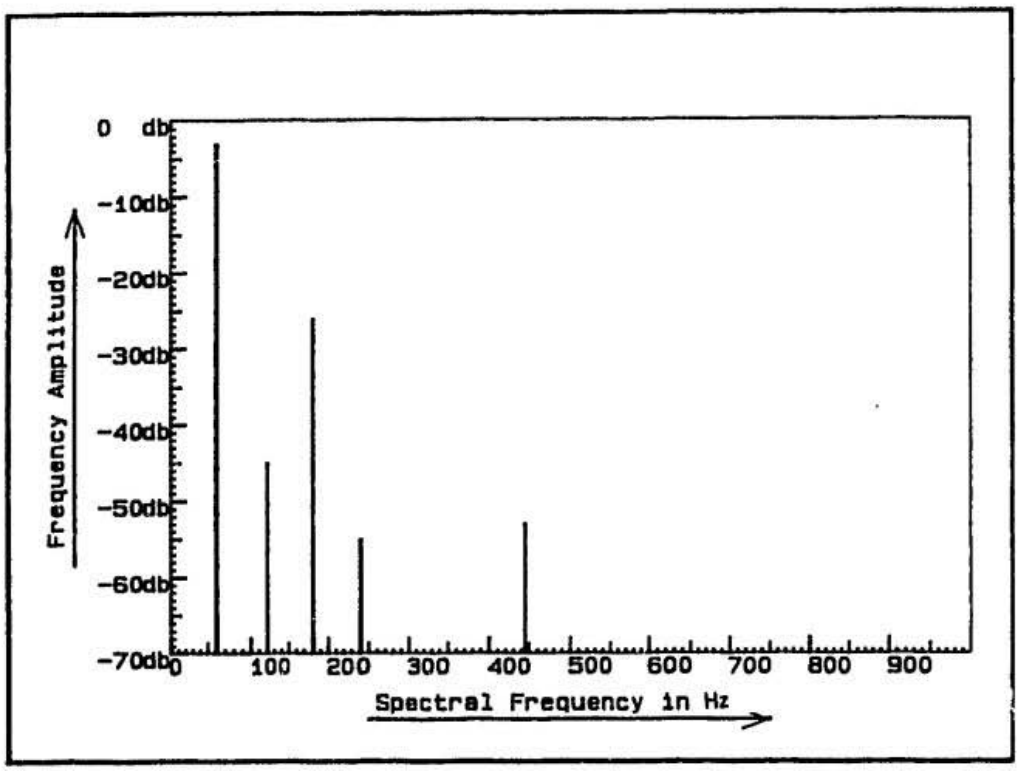

Figure 4.13: Frequency spectra when stator one coil of phase 2. has been externally replaced by a resistor.

4.8.7 Short circuit fault through resistance in Stator Phase 3:

At this condition coil C31 as shown in Fig. 4.8 was disconnected and a resistor of value 9.5 ohm was connected externally in its position. The corresponding frequency spectra is shown below.

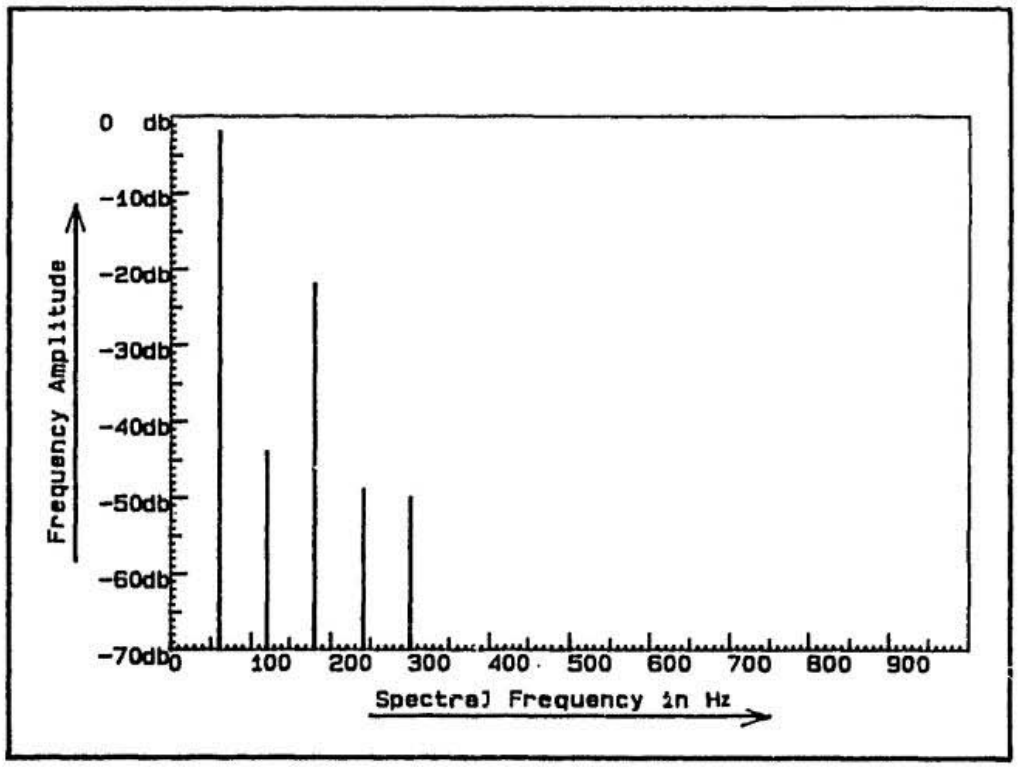

Figure 4.14: Frequency spectra when stator one coil of phase 3. has been externally replaced by a resistor.

4.8.8 Short circuit fault in Stator Phase 1:

This is the condition when coil C11 as shown in Fig. 4.8 was disconnected and coil C12 was directly connected to T1. The frequency spectra under this operating condition is shown below.

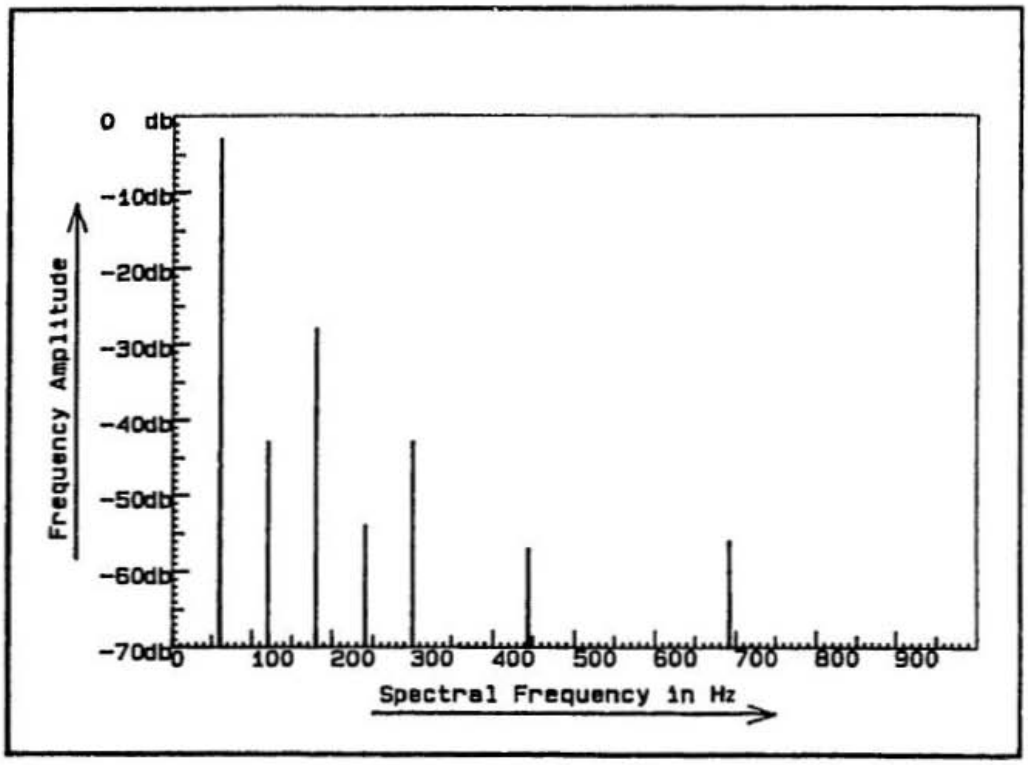

Figure 4.15: Frequency spectra when stator one coil of phase 1 was shorted.

4.8.9 Short circuit fault in Stator Phase 2:

Under this condition coil C21 as shown in Fig. 4.8 was disconnected and coil C22 was directly connected to terminal T2. The frequency spectra is shown below.

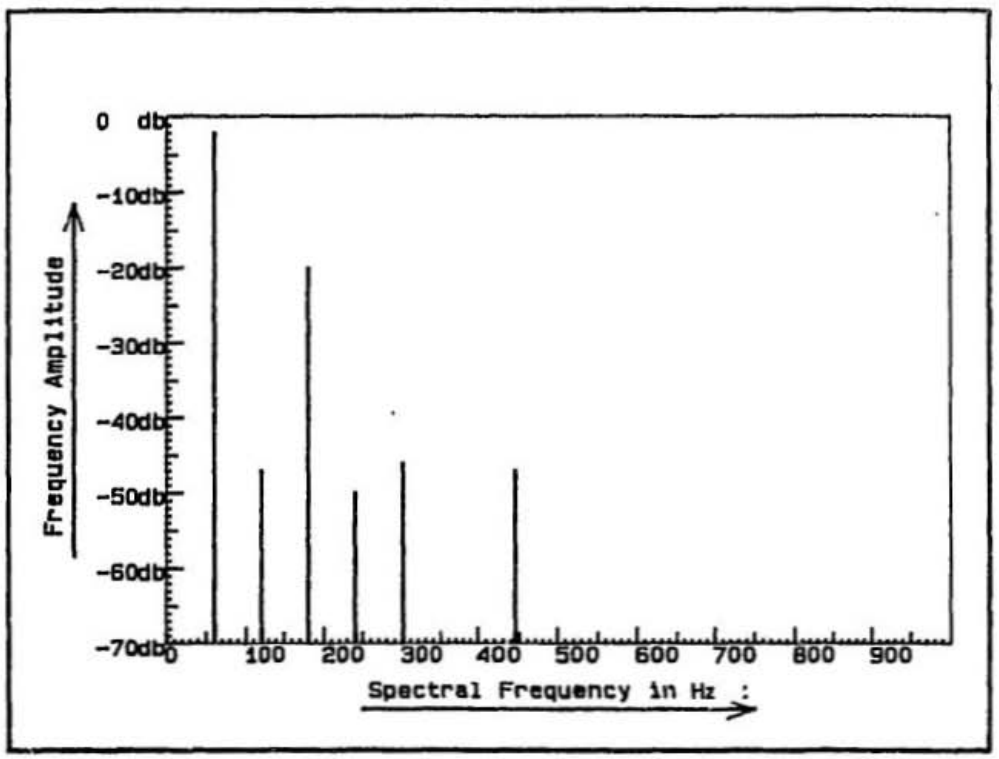

Figure 4.16: Frequency spectra when stator one coil of phase 2 was shorted.

4.8.10 Short circuit fault in Stator Phase 3:

At is the condition coil C31 as shown in Fig. 4.8 was disconnected and coil C33 was directly connected to terminal T3. The corresponding frequency spectra is shown below.

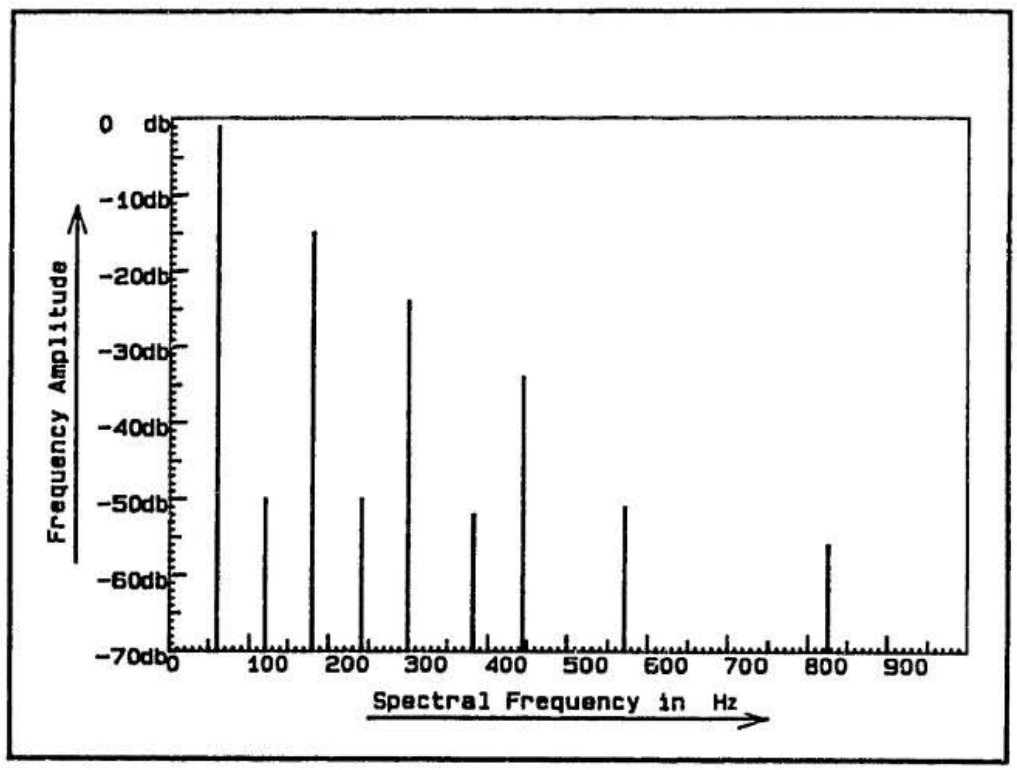

Figure 4.17: Frequency spectra when stator one coil of phase 3 was shorted.

4.8.11 Rotor open circuit fault in phase 1:

According to the Fig. 4.8 the rotor terminals M1,M2 and M3 are short circuited. To make an open circuit fault the rotor's phase M1 was disconnected. Under this condition, the frequency spectra of the stator current is shown below:

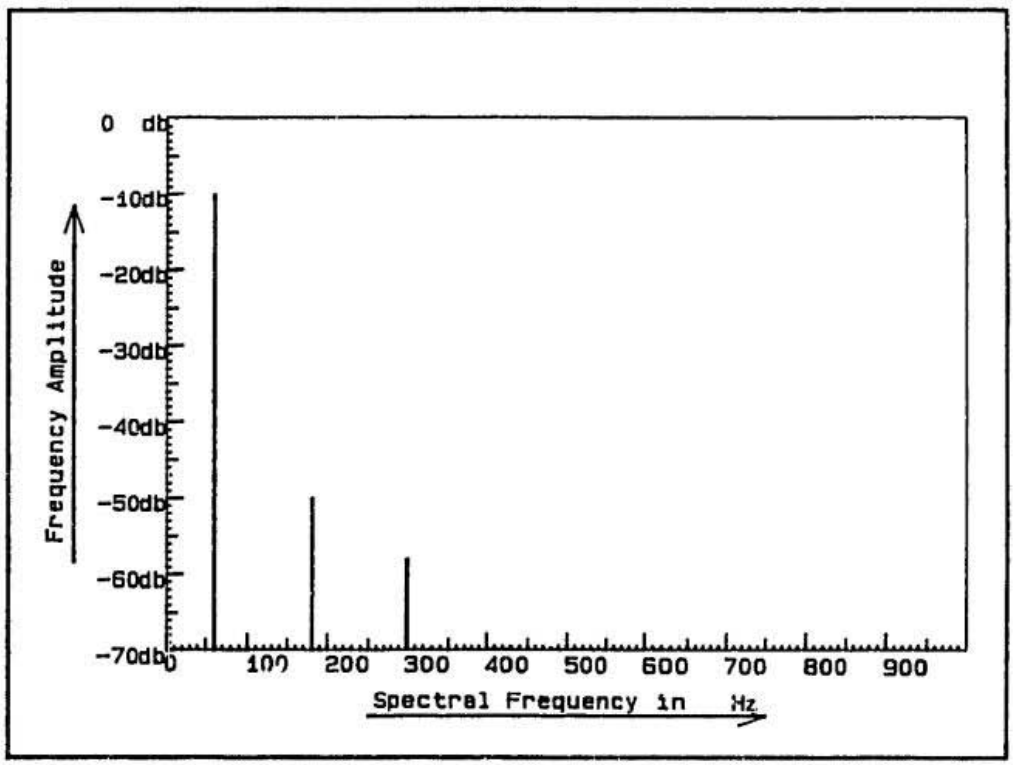

Figure 4.18: Frequency spectra of the stator current when rotor one phase M1 was open circuited.

4.8.12 Rotor open circuit fault in phase 2:

At this condition M2 was disconnected from the close loop of M1, M2 and M3 in rotor's circuit. Under this condition the frequency spectra of the stator current is shown below:

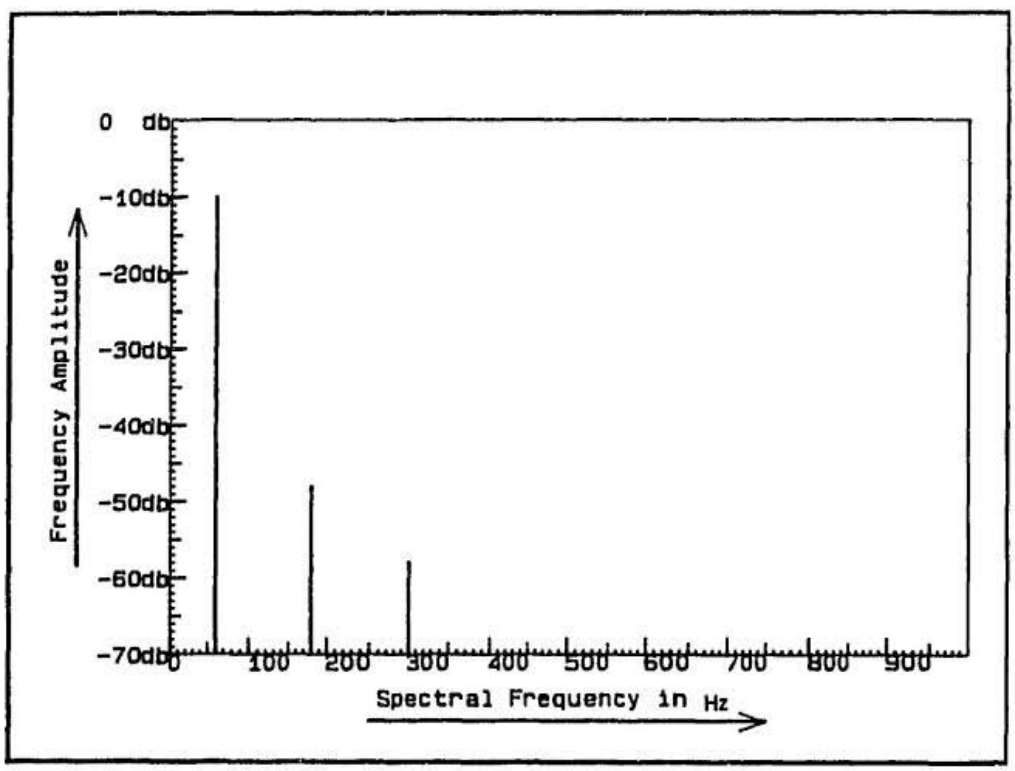

Figure 4.19: Frequency spectra of the stator current when rotor phase M2 was open circuited.

4.8.13 Rotor open circuit fault in phase 3:

To make an open circuit fault in rotor's phase M3, only M1 and M2 were connected and M3 was left open. Under this condition the frequency spectra of the stator current is shown below:

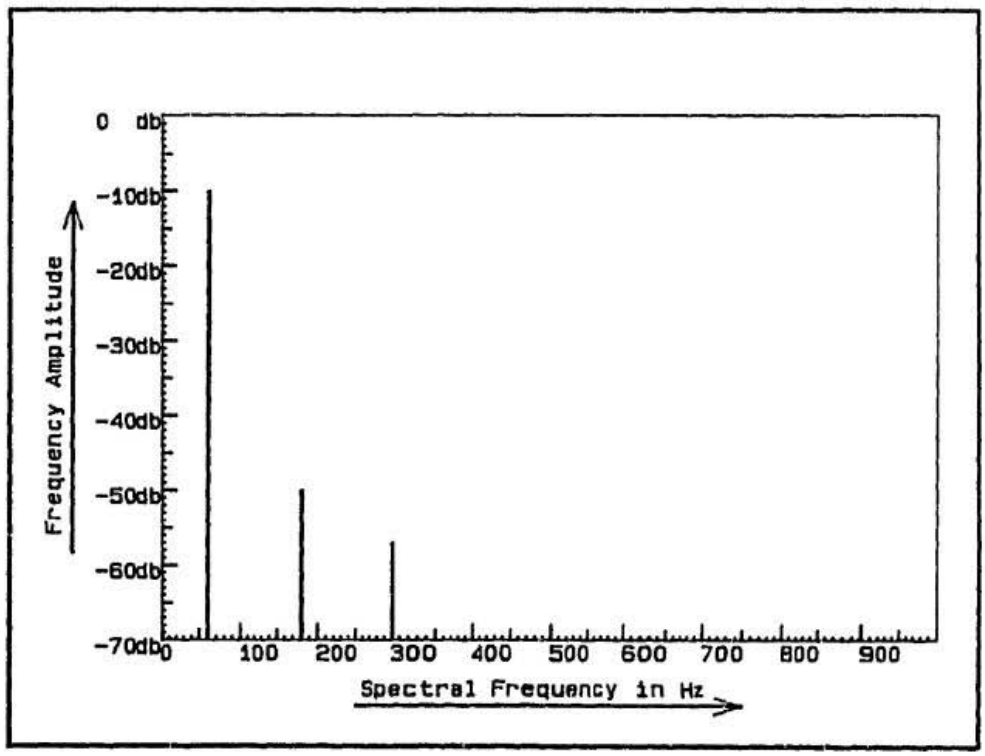

Figure 4.20: Frequency spectra of the stator current when rotor phase M3 was open circuited.

4.8.14 Rotor phase 1 is shorted to neural:

Under the no fault condition, the rotor's terminals M1, M2 and M3 are connected in a single loop as shown in Fig. 4.8. Now to make this fault M1 was disconnected from the loop and it was hooked up to the neutral point of the rotor. It was noted that this type of fault has little impact on wave shape of the stator current whose frequency spectra is shown below:

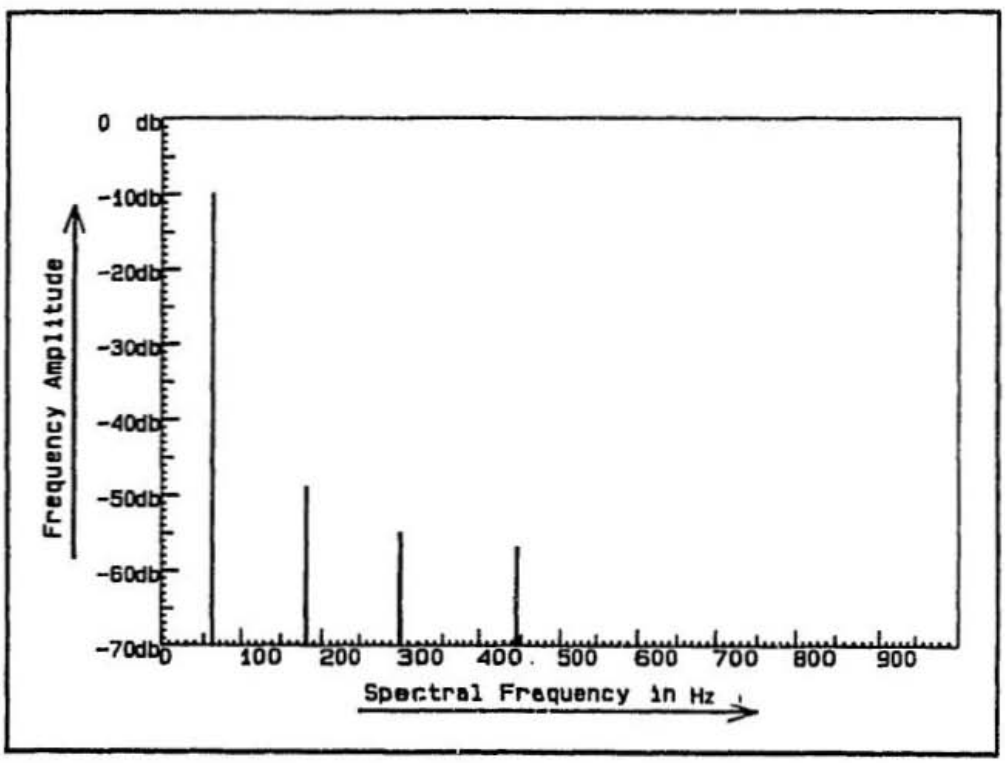

Figure 4.21: Frequency spectra of stator current when rotor phase M1 is shorted to neutral.

4.8.15 Rotor phase 2 is shorted to neural:

At this condition M2 was disconnected form the close loop and it was connected to the neutral point of the rotor. The corresponding frequency spectra is shown in figure below.

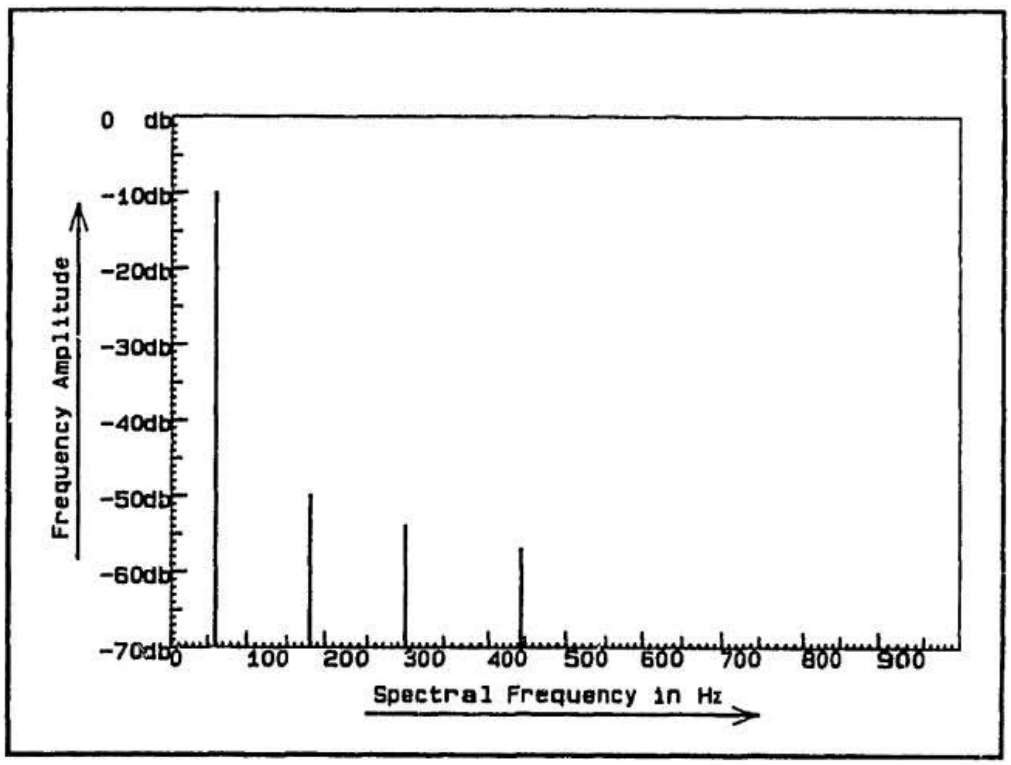

Figure 4.22: Frequency spectra of stator current when rotor phase M2 is shorted to neutral.

4.8.16 Rotor phase 3 is shorted to neural:

Here the rotor terminal M3 was connected to neutral and the terminals M1 and M2 are shorted. Following figure shows the frequency spectra of stator current under this condition.

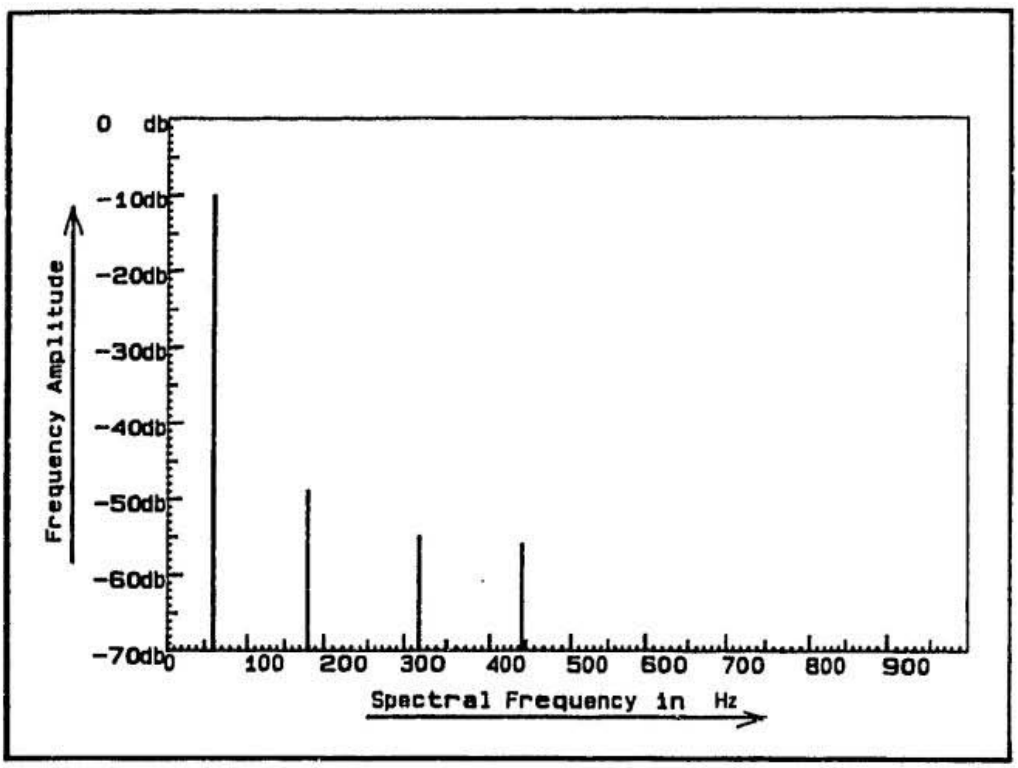

Figure 4.23: Frequency spectra of stator current when rotor phase M3 is shorted to neutral.

4.8.17 Rotor short circuit fault through resistance in phase 1 :

Under this situation it has been considered that phase one of rotor was short through resistance. To make this fault in laboratory M1 was disconnected from the loop as shown in Fig. 4.8 and a resistor of value 6 ohms was connected externally in parallel to M2 and M3. The corresponding frequency spectra is shown below.

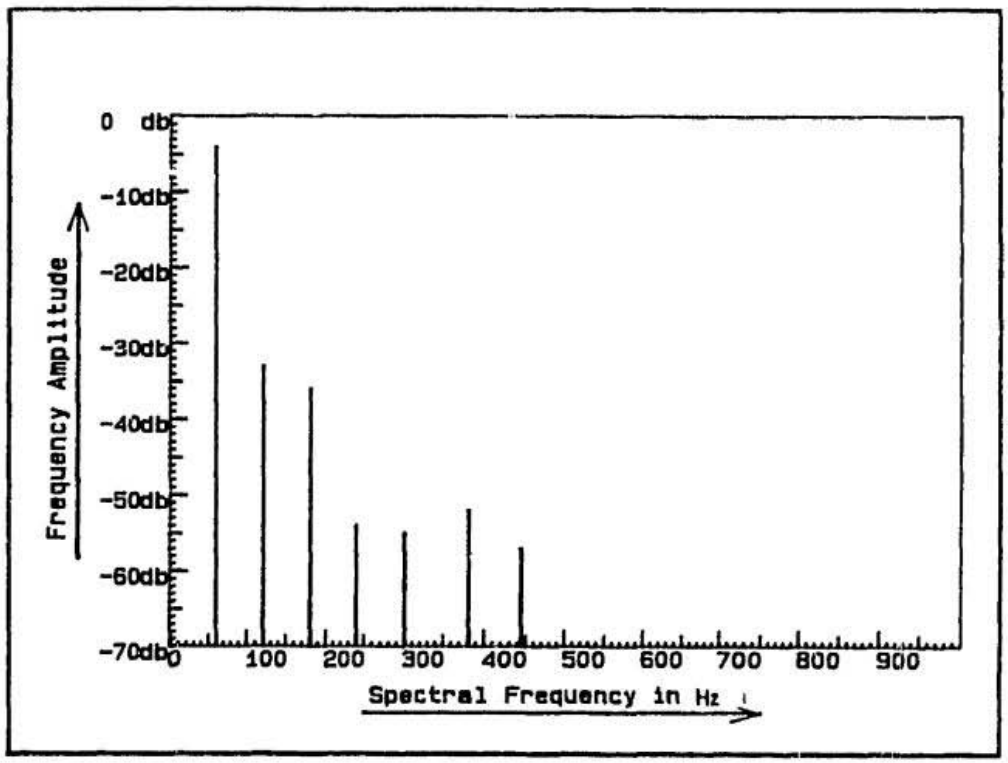

Figure 4.24: Frequency spectra of stator current at rotor short circuit fault through resistance in phase M1.

4.8.18 Rotor short circuit fault through resistance in phase 2 :

M2 was disconnected from the loop and a resistor of value 6 ohms was connected externally in parallel to M1 and M3. The corresponding frequency spectra is shown below.

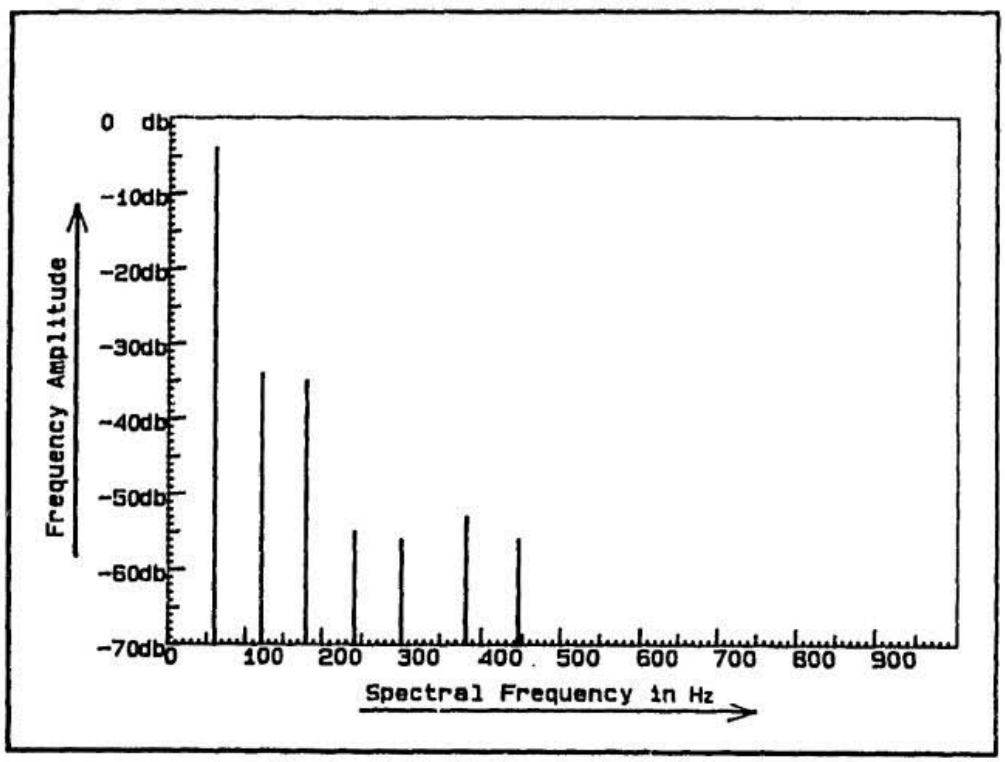

Figure 4.25: Frequency spectra of stator current at rotor short circuit fault through resistance in phase M2 .

4.8.19 Rotor short circuit fault through resistance in phase 3 :

To make this situation in the laboratory M3 was disconnected from the loop and an external resistor of 6 ohms was connected in parallel to M1 and M2. The corresponding frequency spectra is shown below.

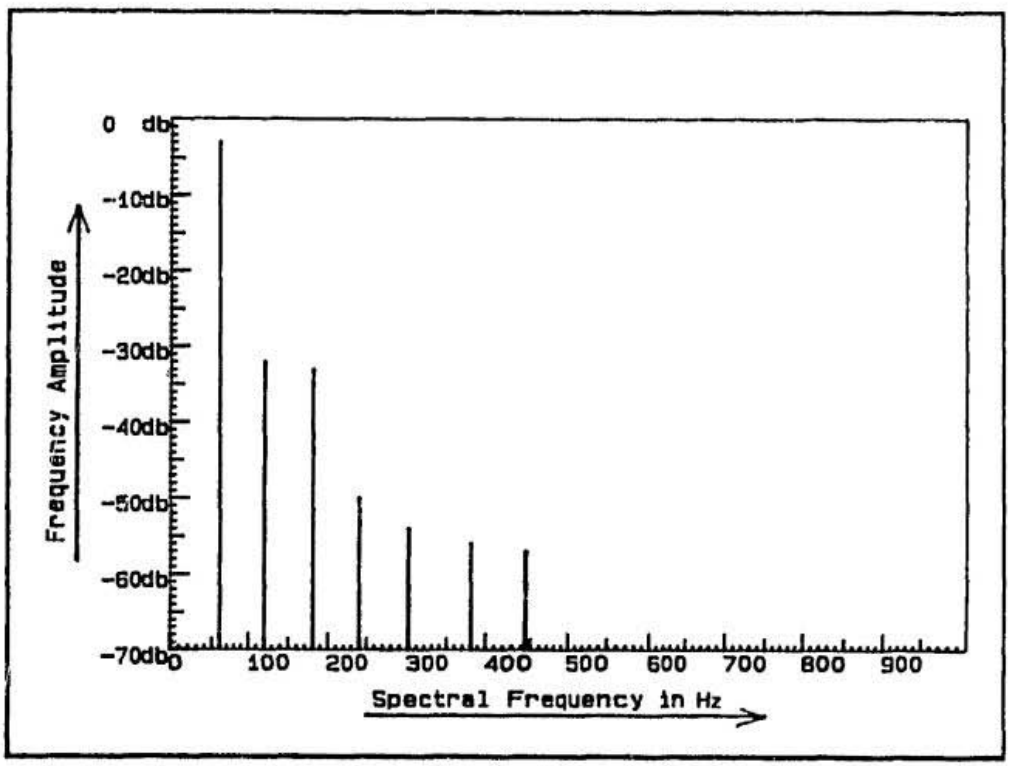

Figure 4.26: Frequency spectra of stator current at rotor short circuit fault through resistance in phase M3.

4.8.20 Rotor phase 1 is unbalanced through an external resistor:

In this situation a resistor of 6 ohms was connected in series with the coil M1 without disturbing the close loop situation. The following figure shows the corresponding spectra.

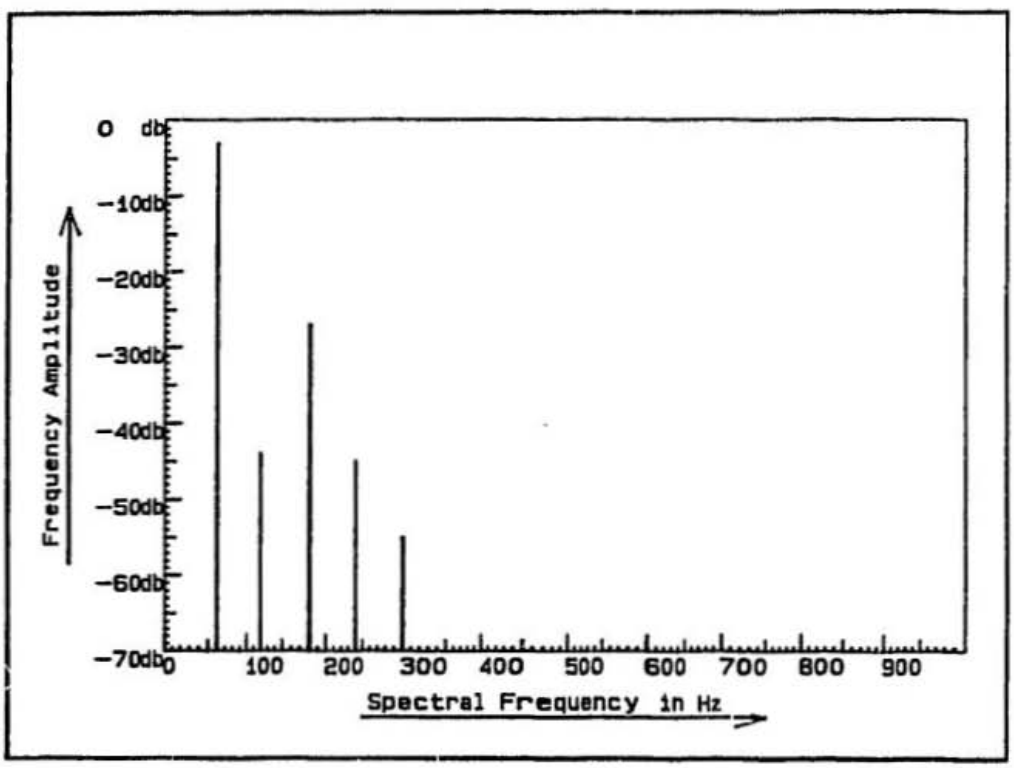

Figure 4.27: Frequency spectra of stator current when there is an unbalance in rotor circuit due to an external resistor in phase M1.

4.8.21 Rotor phase 2 is unbalanced through an external resistor:

To make it happen in the laboratory a resistor of 6 ohms was connected in series with M2 in the same way as the previous one. The corresponding frequency spectra is shown below.

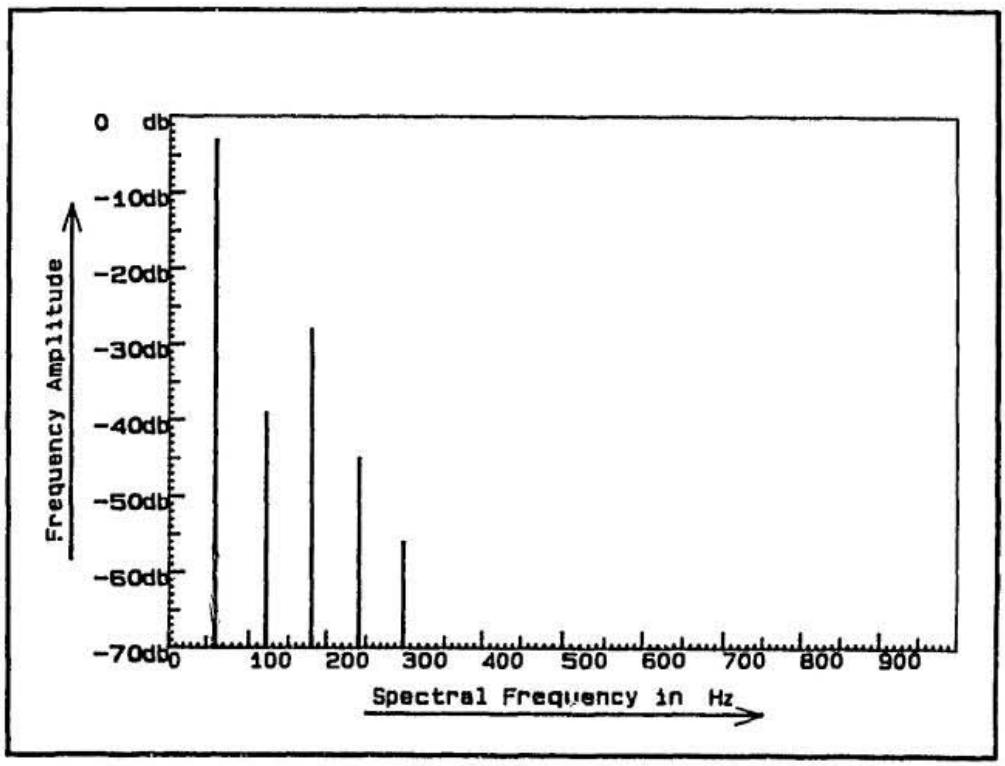

Figure 4.28: Frequency spectra of stator current when there is an unbalance in rotor circuit due to an external resistor in phase M2.

4.8.22 Rotor phase 3 is unbalanced through an external resistor:

A resistor of 6 ohms was connected in series with M3 . The corresponding spectra is shown below.

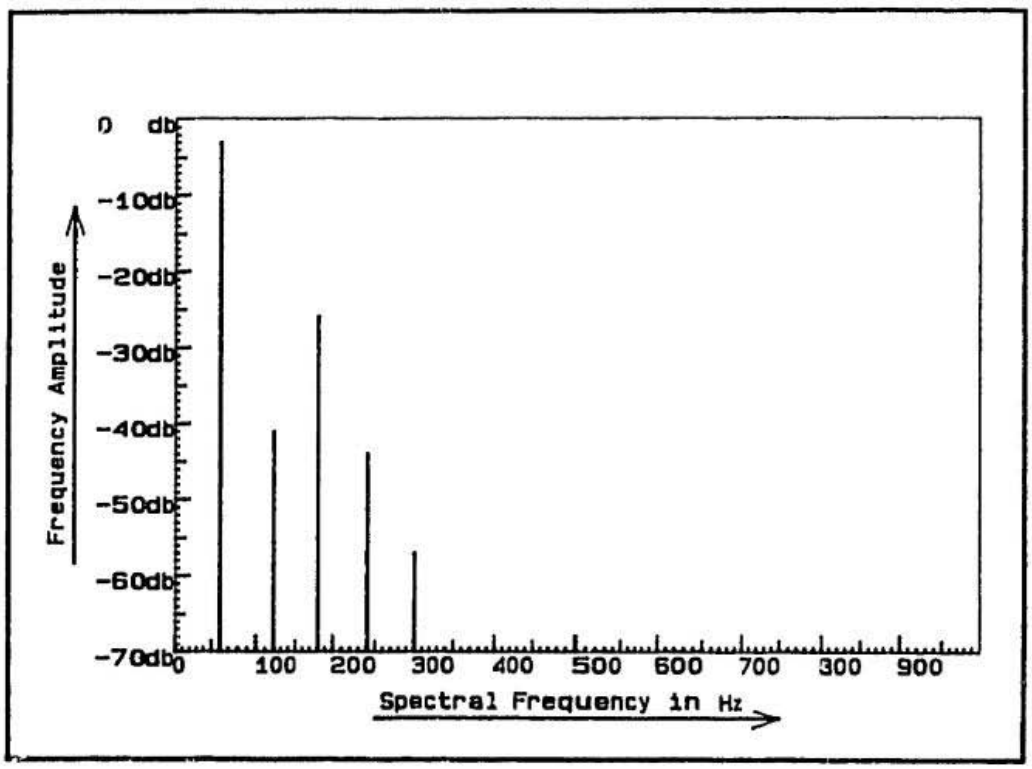

Figure 4.29: Frequency spectra of stator current when there is an unbalance in rotor circuit due to an external resistor in phase M3.

4.8.23 Simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 1:

To make this happen in the laboratory M2 was disconnected from the close loop of the rotor's circuit and phase one of stator was disconnected from the supply. The corresponding frequency spectra of the stator current is shown below.

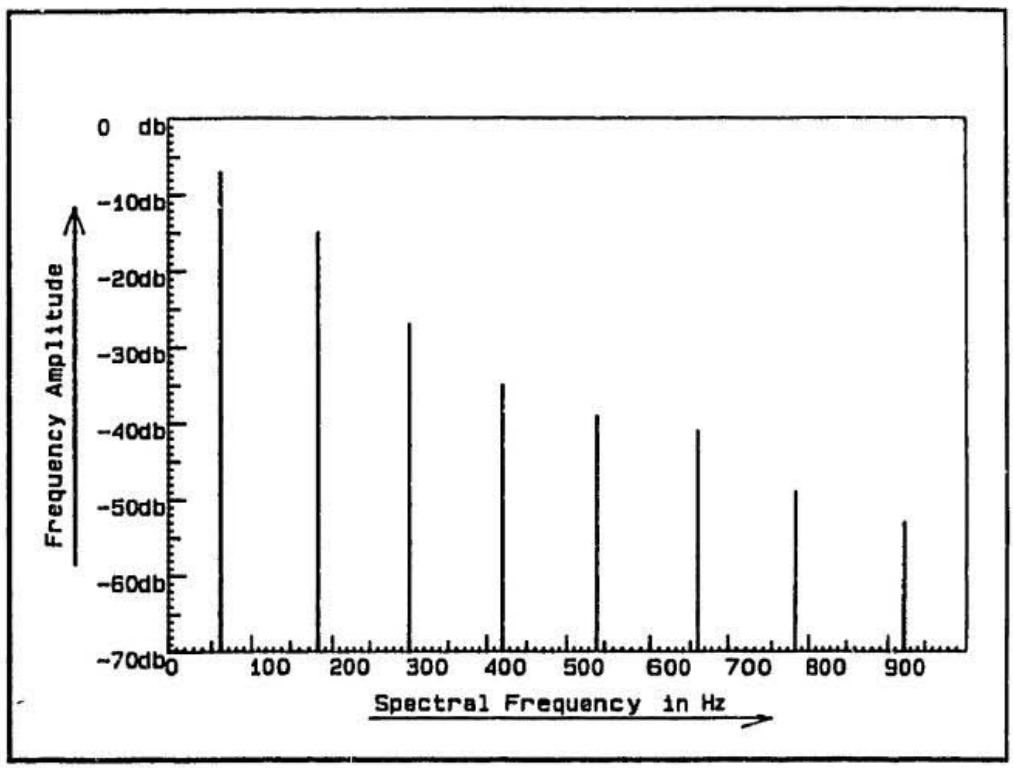

Figure 4.30: Frequency spectra at simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 1.

4.8.24 Simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 2:

To make this happen in the laboratory M2 was disconnected from the close loop of the rotor's circuit and phase 2 of stator was disconnected from the supply. The corresponding frequency spectra is shown below.

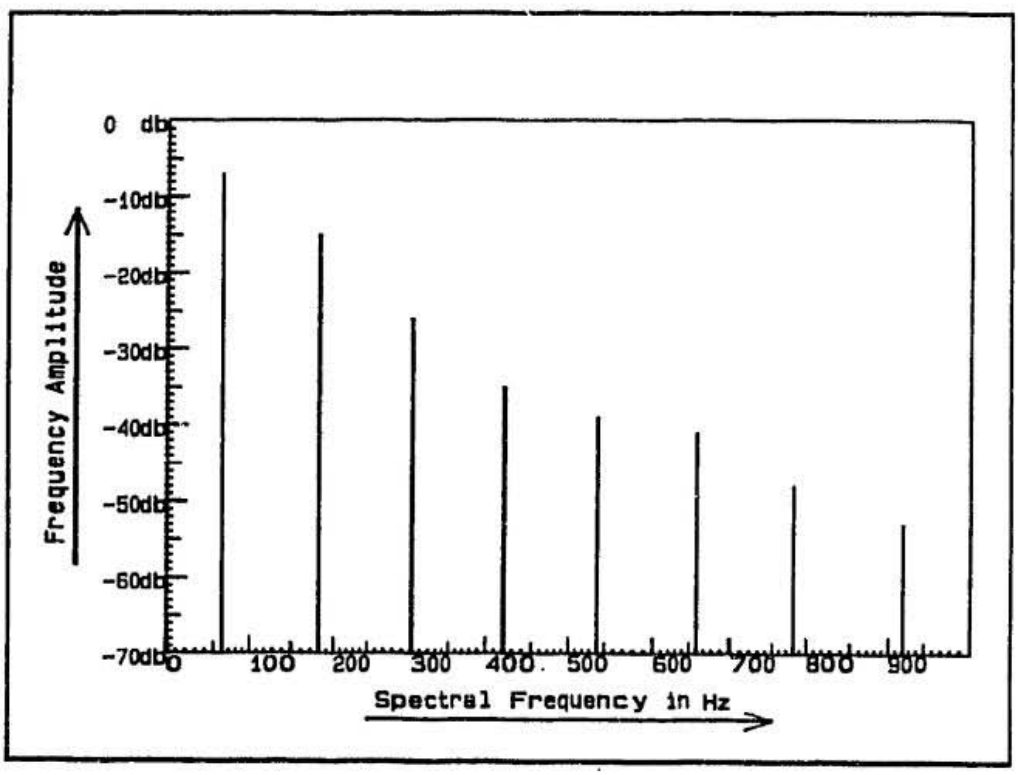

Figure 4.31: Frequency spectra at simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 1.

4.8.25 Simultaneous open circuit fault in Rotor phase 2 as well as Stator phase 3:

At this condition, as stator's phase 3 was open , so there is no stator current in phase 3 .

4.8.26 Remarks on Fault Related Frequenency spectra of the Stator Current

Though a sinusoidal voltage was applied to the motor terminals, the frequency spectra of the stator current carries some higher harmonics in addition to the fundamental due to the non-sinusoidal wave shape of the stator current. It should be noted that in the frequency spectra at different fault conditions, frequency components beyond 5th harmonis are not perfectly divisable by the fundamental; it is due to the slightly non periodic nature of the stator current.

The frequency spectra of the stator current in case of stator's open circuit fault in phase 1 as well as phase 2 are almost identical, though they are different from those of other types of faults. So, it is difficult to differentiate stator open circuit fault in phase 1 from that in phase 2. The same situation is also prevailing in case of rotor open circuit fault and short circuit faults. Moreover, in case of rotor's phase to neutral short circuit fault, the spectra of the stator current is very close to that in case of no fault condition. But the spectra of the major types of faults are significantly different, which opens up the door to use neural network based fault diagnosis scheme .

Chapter 5

Fault Recognition by ART2 Neural Network

5.1 Introduction

The performance of ART2 in pattern recognition both in noise free as well as noisy conditions has been reported in chapter 3. This neural network is not only capable of classifying patterns accurately but also can report if it cannot do so. Moreover, it is suitable in dynamic environment like incipient fault diagnosis of induction motor, as it does not require retraining of already trained patterns to adapt itself to new faults. Frequency spectra of stator current at different fault conditions have been reported in chapter 4. As patterns of those spectra are different for different types of faults, pattern recognition technique can be applied on these spectra to diagonise incipient faults of the induction motor. Under this situation, ART2 neural network can be used to diagonise incipient faults of induction motors based on pattern recognition scheme of frequency spectra of the stator current.

Table 5.1: Table of Faults with unique number.

Serial No.	Pattern No.	Incipient Faults.
0	0	No Fault
1	11	Open circuit fault in Stator's phase 1
2	12	Open circuit fault in Stator's phase 2
3	13	Open circuit fault in Stator's phase 3
4	21	Short circuit fault through resistance in Stator's Phase 1
5	22	Short circuit fault through resistance in Stator's Phase 2
6	23	Short circuit fault through resistance in Stator's Phase 3
7	31	Short circuit fault in Stator's phase 1
8	32	Short circuit fault in Stator's phase 2
9	33	Short circuit fault in Stator's phase 3
10	41	Open circuit fault in Rotor's phase 1
11	42	Open circuit fault in Rotor's phase 2
12	43	Open circuit fault in Rotor's phase 3
13	51	Rotor's Phase 1 is short to neutral
14	52	Rotor's Phase 2 is short to neutral
15	53	Rotor's Phase 3 is short to neutral
16	61	Short circuit fault through resistance in Rotor's Phase 1
17	62	Short circuit fault through resistance in Rotor's Phase 2
18	63	Short circuit fault through resistance in Rotor's Phase 3
19	71	Rotor's phase 1 unbalanced by an external series resistance
20	72	Rotor's phase 2 unbalanced by an external series resistance
21	73	Rotor's phase 3 unbalanced by an external series resistance
22	81	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1
23	82	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 2
24	83	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 3

5.2 Training Data Set

To formulate the fault related spectra in a two-dimensional training data set it is convenient to give unique number to each fault as shown in Table 5.1. The corresponding values of spectral components have been shown in Table 5.2. In this work eight major classes of faults have been studied. To give unique class numbers to different classes of faults pattern number has been used, where the leftmost character of the pattern number represent the class of faults. No fault condition has been numbered as '0'.

Table 5.2: Fault related spectral components in matrix form .

Frequency in Hz.

Pattern number.

	60	120	180	240	300	319	381	420	444	571	540	660	691	780	921
0	-10		-50		-50				-52						
11	-5	-44	-39		-53				-57						
12	-5	-44	-39		-53				-57						
13															
21	-20		-45		-48				-52				-57		
22	-3	-45	-26	-55					-53						
23	-2	-44	-22	-49	-50										
31	-3	-43	-28	-54	-43				-57				-56		
32	-2	-47	-20	-50	-46				-47						
33	-1	-50	-15	-50	-24		-52		-34	-51					
41	-10		-50		-58										
42	-10		-48		-58										
43	-10		-50		-57										
51	-10		-49		-55				-57						
52	-10		-50		-54				-57						
53	-10		-49			-55			-56						
61	-4	-33	-56	-54	-55		-52		-57						
62	-4	-34	-35	-55	-56		-53		-56						
63	-3	-32	-33	-50	-54		-56		-57						
71	-3	-44	-27	-45	-55										
72	-3	-39	-28	-45	-56										
73	-3	-41	-26	-44	-57										
81	-7		-15		-27			-35			-39	-41		-49	-53
82	-7		-15		-26			-35			-39	-41		-48	-53
83															

5.2.1 Data Reduction:

Now to get a suitable training matrix representing patterns of different faults, the following processing have been performed on the matrix of Table 5.2, and after these processing a new matrix has been obtained as shown in Table 5.3.

1. All the components having negative values have been deducted from 70 to make them positive as minimum value of a component may be -70.

Table 5.3: Training Matrix of fault related current spectra.

Frequency in Hz. →

Pattern number ↓

	60	120	180	240	300	319	381	420	444	571
0	30.00	0.00	20.00	0.00	20.00	0.00	0.00	0.00	18.00	0.00
11	32.50	26.00	31.00	0.00	17.00	0.00	0.00	0.00	13.00	0.00
12	32.50	26.00	31.00	0.00	17.00	0.00	0.00	0.00	13.00	0.00
13	0.00	2.00	4.00	6.00	8.00	10.00	12.00	14.00	16.00	18.00
21	25.00	0.00	25.00	0.00	22.00	0.00	0.00	13.00	18.00	0.00
22	33.50	25.00	44.00	15.00	0.00	0.00	0.00	0.00	17.00	0.00
23	34.00	26.00	48.00	21.00	20.00	0.00	0.00	0.00	0.00	0.00
31	33.50	27.00	42.00	16.00	27.00	0.00	0.00	14.00	13.00	0.00
32	34.00	23.00	50.00	20.00	24.00	0.00	0.00	0.00	23.00	0.00
33	34.50	20.00	55.00	20.00	46.00	0.00	18.00	0.00	36.00	19.00
41	30.00	0.00	20.00	0.00	12.00	0.00	0.00	0.00	0.00	0.00
42	30.00	0.00	22.00	0.00	12.00	0.00	0.00	0.00	0.00	0.00
43	30.00	0.00	20.00	0.00	13.00	0.00	0.00	0.00	0.00	0.00
51	30.00	0.00	21.00	0.00	15.00	0.00	0.00	0.00	13.00	0.00
52	30.00	0.00	20.00	0.00	16.00	0.00	0.00	0.00	13.00	0.00
53	30.00	0.00	21.00	0.00	0.00	15.00	0.00	0.00	14.00	0.00
61	33.00	37.00	14.00	16.00	15.00	0.00	18.00	0.00	13.00	0.00
62	33.00	36.00	35.00	15.00	14.00	0.00	17.00	0.00	14.00	0.00
63	33.50	38.00	37.00	20.00	16.00	0.00	14.00	0.00	13.00	0.00
71	33.50	26.00	43.00	25.00	15.00	0.00	0.00	0.00	0.00	0.00
72	33.50	31.00	42.00	25.00	14.00	0.00	0.00	0.00	0.00	0.00
73	33.50	29.00	44.00	26.00	13.00	0.00	0.00	0.00	0.00	0.00
81	31.50	0.00	55.00	0.00	43.00	31.00	29.00	35.00	21.00	17.00
82	31.50	0.00	55.00	0.00	44.00	31.00	29.00	35.00	22.00	17.00
83	0.00	2.00	4.00	6.00	8.00	10.00	12.00	14.00	16.00	18.00

2. All blanks cells have been filled by zeros to make them sensible to neural network.

3. Spectral components of higher frequencies have been shifted to lower empty spaces as there are only few components at that region.

4. As fundamental component is dominant, so it has been reduced by fifty percent to give importance to other components which really carry fault related information.

Table 5.4: Faults mapping of the trained network in high precision domain

Serial No.	Pattern No.	Incipient Faults.
0	0	No Fault
1	11	Open circuit fault in Stator's phase 1 or 2
2	12	Open circuit fault in Stator's phase 1 or 2
3	13	Open circuit fault in Stator's phase 3
4	21	Short circuit fault through resistance in Stator's Phase 1
5	22	Short circuit fault through resistance in Stator's Phase 2
6	23	Short circuit fault through resistance in Stator's Phase 3
7	31	Short circuit fault in Stator's phase 1
8	32	Short circuit fault in Stator's phase 2
9	33	Short circuit fault in Stator's phase 3
10	41	Open circuit fault in Rotor's any phase
11	42	Open circuit fault in Rotor's any phase
12	43	Open circuit fault in Rotor's any phase
13	51	Rotor's Phase 1or 2 is short to neutral
14	52	Rotor's Phase 1 or 2 is short to neutral
15	53	Rotor's Phase 3 is short to neutral
16	61	Short circuit fault through resistance in Rotor's Phase 1
17	62	Short circuit fault through resistance in Rotor's Phase 2
18	63	Short circuit fault through resistance in Rotor's Phase 3
19	71	Rotor's phase 1 or 2 unbalanced by an external series resistance
20	72	Rotor's phase 1 or 2 unbalanced by an external series resistance
21	73	Rotor's phase 3 unbalanced by an external series resistance
22	81	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
23	82	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
24	83	Open circuit fault in Stator's Phase 3

5.3 Structure of the Network

Now the matrix as shown in Table 5.3 represents the training matrix. As this is a 25x10 dimension matrix, so the components on F1 layer of ART2 should be 10. Now there are 25 different patterns to be recorded in the neural network, but a number of them are similar, so the number of nodes on F2 layer should be less than 25. For this reason it has been choosen to be 20.

5.4 Training of the Network

Training of the network has been performed in high precision domain. It should be noted that training time with 25 patterns of dimension 10 as shown in Table 5.3 is very low and it is less than one minute. At the end of the training the knowledge of fault diagnosis is now embedded in the weight vector and the network has classified different patterns as the signature of corresponding faults as shown in Table 5.4. The bottom-up and top-down weight matrices are shown in Table 5.5 and Table 5.6, respectively. From Table 5.4 some important remarks regarding the fault classification of ART2 based on frequency spectra of the stator current, can be noted in the following points:

1. Faults having pattern numbers 11 and 12 have been classified in the same group. As their frequency spectra of the stator current are similar, so the network is unable to differentiate between them.

2. In case of pattern numbers 51 and 52 as frequency spectra are similar, so network has grouped them in the same class. It should be noted that the spectra of pattern numbers 51,52 and 53 have similar shape to that of pattern number 0. So, there is a chance that network may recognize rotor's short-circuit-to-neutral fault as no fault condition.

3. The same situation is prevailing in case of pattern numbers 71 and 72 as well as 81 and 82. In fact in such cases, as the spectra are close to each other, it is difficult for the neural network to differentiate between them.

Table 5.5: Top-down weight matrix of the trained network.

8.34	0.00	5.56	0.00	5.56	0.00	0.00	0.00	5.00	0.00
7.24	5.79	6.90	0.00	3.79	0.00	0.00	0.00	2.89	0.00
0.00	0.00	0.00	0.00	3.04	3.80	4.56	5.32	6.07	6.83
6.62	0.00	6.62	0.00	5.83	0.00	0.00	3.44	4.77	0.00
6.46	4.82	8.49	2.89	0.00	0.00	0.00	0.00	3.28	0.00
6.02	4.61	8.50	3.72	3.54	0.00	0.00	0.00	0.00	0.00
6.17	4.98	7.74	2.95	4.98	0.00	0.00	0.00	0.00	0.00
5.63	3.81	8.29	3.31	3.98	0.00	0.00	0.00	3.81	0.00
4.70	2.72	7.49	2.72	6.26	0.00	0.00	0.00	4.90	0.00
9.87	0.00	6.58	0.00	3.95	0.00	0.00	0.00	0.00	0.00
9.00	0.00	6.30	0.00	4.50	0.00	0.00	0.00	3.90	0.00
8.93	0.00	6.25	0.00	0.00	4.47	0.00	0.00	4.17	0.00
6.85	7.68	2.91	3.32	3.11	0.00	3.74	0.00	2.70	0.00
6.14	6.70	6.51	2.79	2.60	0.00	3.16	0.00	2.60	0.00
6.18	7.01	6.83	3.69	2.95	0.00	0.00	0.00	0.00	0.00
6.13	5.67	7.66	4.57	2.56	0.00	0.00	0.00	0.00	0.00
6.19	5.36	8.13	4.80	0.00	0.00	0.00	0.00	0.00	0.00
4.08	0.00	7.12	0.00	5.56	4.01	3.75	4.53	2.72	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5.6: Bottom-up weight matrix of the trained network.

8.34	7.24	0.00	6.62	6.46	6.02	6.17	5.63	4.70	9.87	9.00	8.93	6.85	6.14	6.18	6.13	6.19	4.08	0.00	0.00
0.00	5.79	0.00	0.00	4.82	4.61	4.98	3.81	2.72	0.00	0.00	0.00	7.68	6.70	7.01	5.67	5.36	0.00	0.00	0.00
5.56	6.90	0.00	6.62	8.49	8.50	7.74	8.29	7.49	6.58	6.30	6.25	2.91	6.51	6.83	7.68	8.13	7.12	0.00	0.00
0.00	0.00	0.00	0.00	2.89	3.72	2.95	3.31	2.72	0.00	0.00	0.00	3.32	2.79	3.69	4.57	4.80	0.00	0.00	0.00
5.56	3.79	3.04	5.83	0.00	3.54	4.98	3.98	6.26	3.95	4.50	0.00	3.11	2.60	2.95	2.56	0.00	5.56	0.00	0.00
0.00	0.00	3.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.47	0.00	0.00	0.00	0.00	0.00	4.01	0.00	0.00
0.00	0.00	4.56	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.74	3.16	0.00	0.00	0.00	3.75	0.00	0.00
0.00	0.00	5.32	3.44	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.53	0.00	0.00
5.00	2.89	6.07	4.77	3.28	0.00	0.00	3.81	4.90	0.00	3.90	4.17	2.70	2.60	0.00	0.00	0.00	2.72	0.00	0.00
0.00	0.00	6.83	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

5.5 Fault Recognition by the trained network.

To test the diagnostic performance of the trained network both in noise free as well as noisy conditions, the training data set was taken as first test pattern and after that noisy data set. The performance of the network to diagnose the fault in noise free condition is identical to that of Table 5.4. To simulate a noisy situation, random noise was added to spectral components as shown in Table 5.8. Every bordered cells including shaded cells are corrupted, but shaded ones are corrupted heavily. Now the trained network has been asked to diagnose the faults related to these noisy spectra and the diagnostic performance is reported in Table 5.7. From this table it should be noted that the trained network is unable to detect three faults in this noisy condition. This operational scenario can be improved to handle higher noise margin, but that will loose accuracy by training the network in low precision mode. Under this situation the pattern mapping in training stage and fault diagnosis performance in previous noisy situation are shown in Table 5.9 and Table 5.10, respectively. Table 5.10 aparently shows better performance with only one detection failure but Table 5.9 shows that the network has mapped the rotor short-circuit-to-neutral fault as no fault condition. Thus there exists a need for optimization of accuracy and noise margin. It is beyond the scope of this thesis.

5.6 Model of ART2 neural network based Incipient Fault Detection System

Based on the performance of ART2 neural network in fault diagnosis of induction motor using the frequency spectra of the stator current, it is evident that the basic requirements for incipient fault detection of induction motors can be met with ART2 neural network based system. Fig. 5.1 gives the schematic of such a fault detection system. Here the controller gives the frequency spectra of the stator current to the trained network which detects the present internal condition of the machine. If it can diagnose a fault, the controller gets specific fault related information from the corresponding database and can report to the user. If the neural network is unable to detect a fault, the controller asks the user for information related to the specific situation and simultaneously updates its knowledge-base as well as the underlying neural network.

Table 5.7: Fault related noisy current spectra.

	60	120	180	240	300	319	381	420	444	571
0	30.00	0.00	20.00	0.00	20.00	0.00	0.00	0.00	18.00	0.00
11	32.50	26.00	32.00	0.00	17.00	2.00	0.00	0.00	13.00	0.00
12	32.50	26.00	30.00	0.00	18.00	0.00	0.00	0.00	13.00	0.00
13	0.00	2.00	4.00	6.00	8.00	10.00	12.00	14.00	16.00	18.00
21	25.00	2.00	25.00	0.00	22.00	0.00	0.00	13.00	19.00	0.00
22	33.50	25.00	50.00	15.00	0.00	0.00	0.00	0.00	19.00	0.00
23	34.00	26.00	48.00	[illegible]	20.00	0.00	0.00	0.00	0.00	0.00
31	33.50	27.00	42.00	18.00	27.00	0.00	0.00	14.00	13.00	0.00
32	34.00	23.00	52.00	21.00	24.00	0.00	0.00	0.00	23.00	0.00
33	34.50	20.00	55.00	20.00	46.00	0.00	18.00	0.00	36.00	19.00
41	30.00	0.00	20.00	0.00	12.00	0.00	0.00	0.00	0.00	0.00
42	30.00	0.00	[illegible]	0.00	[illegible]	0.00	0.00	0.00	0.00	0.00
43	30.00	0.00	20.00	0.00	13.00	0.00	0.00	0.00	0.00	0.00
51	30.00	0.00	22.00	0.00	15.00	0.00	0.00	0.00	13.00	0.00
52	30.00	0.00	20.00	0.00	16.00	0.00	0.00	0.00	13.00	0.00
53	30.00	0.00	21.00	0.00	0.00	15.00	0.00	0.00	14.00	0.00
61	33.00	37.00	14.00	16.00	15.00	0.00	18.00	0.00	13.00	0.00
62	33.00	40.00	35.00	15.00	14.00	0.00	19.00	0.00	14.00	0.00
63	33.50	38.00	[illegible]	20.00	16.00	0.00	14.00	0.00	13.00	0.00
71	33.50	26.00	43.00	25.00	15.00	0.00	0.00	0.00	0.00	0.00
72	33.50	31.00	42.00	25.00	15.00	0.00	0.00	0.00	0.00	0.00
73	33.50	29.00	54.00	26.00	13.00	0.00	0.00	0.00	0.00	0.00
81	31.50	0.00	55.00	0.00	43.00	31.00	29.00	35.00	21.00	17.00
82	31.50	0.00	55.00	0.00	44.00	31.00	29.00	35.00	22.00	17.00
83	0.00	2.00	4.00	6.00	8.00	10.00	12.00	14.00	16.00	18.00

Table 5.8: Diagnostic test result of the trained network in noisy situation.

Serial No.	Pattern No.	Incipient Faults.
0	0	No Fault
1	11	Open circuit fault in Stator's phase 1 or 2
2	12	Open circuit fault in Stator's phase 1 or 2
3	13	Open circuit fault in Stator's phase 3
4	21	Short circuit fault through resistance in Stator's Phase 1
5	22	Short circuit fault through resistance in Stator's Phase 2
6	23	Unable to diagonise the fault
7	31	Short circuit fault in Stator's phase 1
8	32	Short circuit fault in Stator's phase 2
9	33	Short circuit fault in Stator's phase 3
10	41	Open circuit fault in Rotor's any phase
11	42	Unable to diagonise the fault
12	43	Open circuit fault in Rotor's any phase
13	51	Rotor's Phase 1 or 2 is short to neutral
14	52	Rotor's Phase 1 or 2 is short to neutral
15	53	Rotor's Phase 3 is short to neutral
16	61	Short circuit fault through resistance in Rotor's Phase 1
17	62	Short circuit fault through resistance in Rotor's Phase 2
18	63	Unable to diagonise the fault
19	71	Rotor's phase 1 or 2 unbalanced by an external series resistance
20	72	Rotor's phase 1 or 2 unbalanced by an external series resistance
21	73	Rotor's phase 3 unbalanced by an external series resistance
22	81	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
23	82	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
24	83	Open circuit fault in Stator's phase 3

Table 5.9: Faults mapping in training phase in low precision domain.

Serial No.	Pattern No.	Incipient Faults.
0	0	No Fault
1	11	Open circuit fault in Stator's phase 1 or 2
2	12	Open circuit fault in Stator's phase 1 or 2
3	13	Open circuit fault in Stator's phase 3
4	21	Short circuit fault through resistance in Stator's Phase 1
5	22	Short circuit fault through resistance in Stator's Phase 2
6	23	Short circuit fault through resistance in Stator's Phase 3
7	31	Short circuit fault through resistance in Stator's Phase 3
8	32	Short circuit fault in Stator's phase 2
9	33	Short circuit fault in Stator's phase 3
10	41	Open circuit fault in Rotor's any phase
11	42	Open circuit fault in Rotor's any phase
12	43	Open circuit fault in Rotor's any phase
13	51	No Fault
14	52	No Fault
15	53	Rotor's Phase 3 is short to neutral
16	61	Short circuit fault through resistance in Rotor's Phase 1
17	62	Short circuit fault through resistance in Rotor's Phase 2
18	63	Rotor's phase 1 or 2 unbalanced by an external series resistance
19	71	Rotor's phase 1 or 2 unbalanced by an external series resistance
20	72	Rotor's phase 1 or 2 unbalanced by an external series resistance
21	73	Rotor's phase 3 unbalanced by an external series resistance
22	81	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
23	82	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
24	83	Open circuit fault in Stator's phase 3

Table 5.10: Fault diagnostic performance of the network trained in low precision domain in noisy situation .

Serial No.	Pattern No.	Incipient Faults.
0	0	No Fault
1	11	Open circuit fault in Stator's phase 1 or 2
2	12	Open circuit fault in Stator's phase 1 or 2
3	13	Open circuit fault in Stator's phase 3
4	21	Short circuit fault through resistance in Stator's Phase 1
5	22	Short circuit fault through resistance in Stator's Phase 2
6	23	Unable to diagonise the fault
7	31	Short circuit fault through resistance in Stator's Phase 3
8	32	Short circuit fault in Stator's phase 2
9	33	Short circuit fault in Stator's phase 3
10	41	Open circuit fault in Rotor's any phase
11	42	Open circuit fault in Rotor's any phase
12	43	Open circuit fault in Rotor's any phase
13	51	No Fault
14	52	No Fault
15	53	Rotor's Phase 3 is short to neutral
16	61	Short circuit fault through resistance in Rotor's Phase 1
17	62	Short circuit fault through resistance in Rotor's Phase 2
18	63	Rotor's phase 1 or 2 unbalanced by an external series resistance
19	71	Rotor's phase 1 or 2 unbalanced by an external series resistance
20	72	Rotor's phase 1 or 2 unbalanced by an external series resistance
21	73	Rotor's phase 3 unbalanced by an external series resistance
22	81	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
23	82	Simultaneous open circuit fault in Rotor's phase 2 as well as Stator's phase 1 or 2
24	83	Open circuit fault in Stator's phase 3

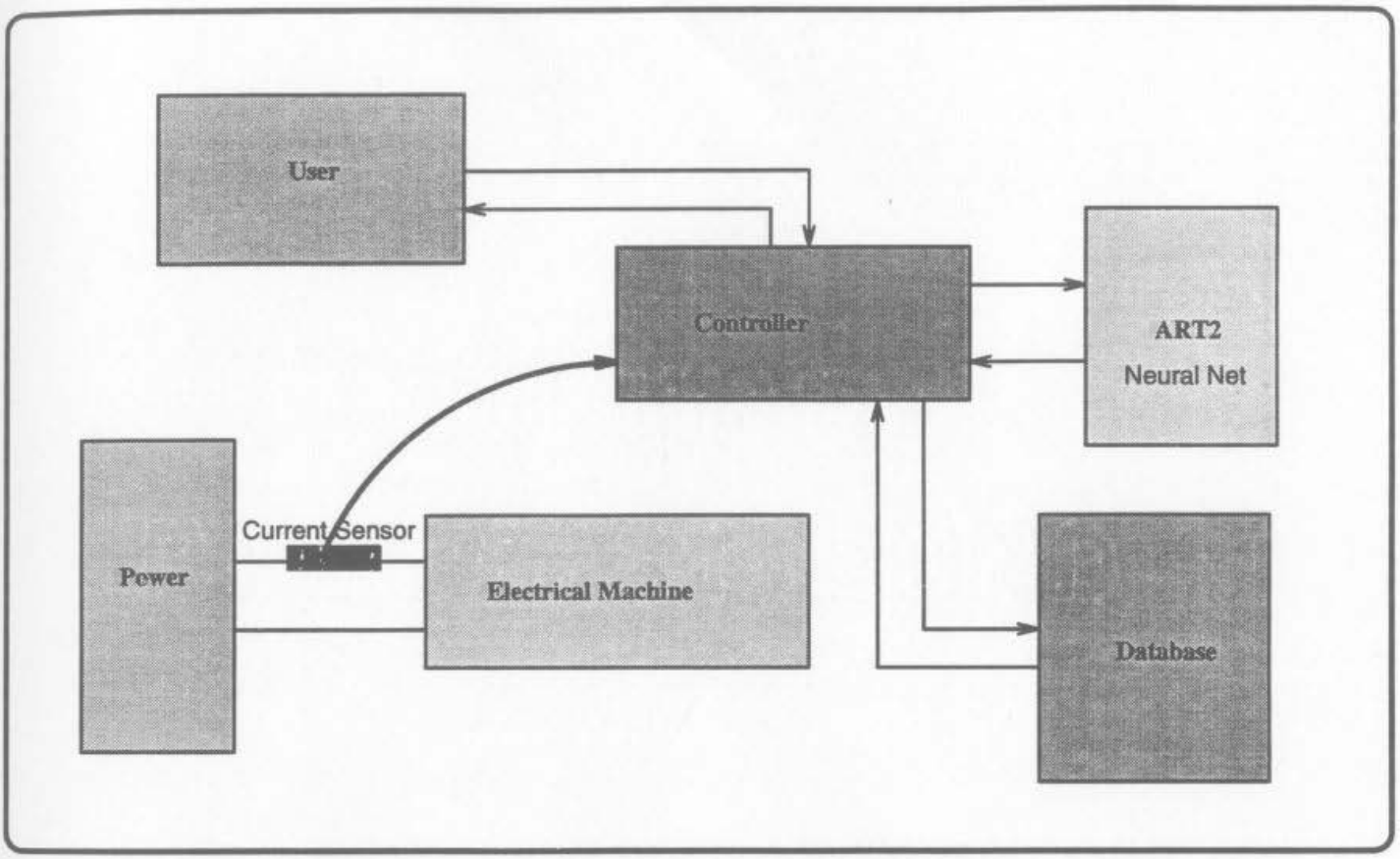

Figure 5.1: Model of ART2 neural network based on-line incipient fault diagnosis system for induction motors.

Chapter 6

Conclusions and Recommendations for Future Work

6.1 Conclusions

In this thesis an incipient fault detection scheme of induction motors based on ART2 neural network has been developed. This fault diagnosis scheme is not only capable of detecting a fault but also can report if it cannot diagnose a particular fault, so that preventive steps can be taken to update the underlying neural network to cope with this undetected fault while retaining the already acquired knowledge without retraining of the trained patterns. The accuracy of this fault diagnosis scheme is susceptable to noise margins. For lower noise level, it gives high accuracy in its performance.

A laboratory experimental set-up based on DSP techniques to collect the fault related frequency spectra in real-time of the stator current of induction

motor has been developed. This is a general purpose frequency spectra collection system, which can be used for other purposes. Frequency spectra of the stator current of a wound-rotor induction motor at different fault conditions making unbalance in the stator as well as rotor circuits have been collected. From the patterns of the spectra it is evident that the major faults can be detected from these spectra through a pattern recognition scheme. But it is difficult to differentiate similar faults as they have identical frequency spectra.

ART2 neural network has been implemented using object oriented software methodology. Its training, recalling and dynamic updating performance have been studied. It's training time is very low in comparison to the popular Feedforward Neural Network. It can update its knowledge to cope with the new pattern while retaining the acquired knowledge without the need of retraining.

In this thesis work, only stator current has been considered as fault related information carrier. But in order to develop a comprehensive fault diagnosis system, multi-sensors based scheme may give better result. This is beyond the scope of this thesis. Through this research work the present state-of-the-art of incipient fault detection of an induction motor has been improved to find better neural network and to collect fault related frequency spectra of the stator current, which can be used as a basis for the development of robust and comprehensive fault diagnosis system for electrical machines .

6.2 Recommendations for Future Work

From previous work [25] it is evident that information from a single sensor is not good enough to identify all possible faults that may occur in induction machines. As for example, stator current usually does not carry the signature of the bearing faults. Therefore it is necessary to pay attention in multiple sensors based scheme. Performance of a machine fault diagnosis system should be independent of operating conditions of the machine as well as the design parameter variations. Moreover, the system should report not only the type of faults but also the extent of the detected fault. To make it cost-effective and user-friendly for the maintenance engineer in the case of a large manufacturing plant, a single system should be capable to diagonise faults of multiple machines.

The future work should also consider both the theoretical as well as the experimental study to select the optimum number of sensors necessary to collect all major faults related information. For example, a vibration sensor may be necessary for bearing faults.

Data from multiple sensors should be reduced through neural network based filter to avoid redundant information.

Usually faults related information are contaminated by the interference of operating conditions as well as the design parameter variation of the machine. It is thus necessary to develop suitable neural network based filter to get noise free fault related information.

As the extent of a particular fault is fuzzy in nature, therefore, in such a

case the application of fuzzy logic should be explored. A neuro-fuzzy computing model should be developed for the incipient fault diagnosis of electrical machines.

Bibliography

[1] T.S. Sankar," Diagnosis and Monitoring for AC Drives," *IAS Conf. record,* vol. 1, 1992, pp. 370-377.

[2] Peter J. Tavner, James Penman, *Condition Monitoring of Electrical Machines,*Research Press. Ltd. John Wiley & Sons Inc.

[3] Mo-Yuen Chow and Sui-Oi Yee, "Methodology for On-line Incipient Fault Detection in Single Phase Squirrel-case Induction Motors using Artificial Neural Networks," *IEEE Transaction on Energy Conversion,* vol. 6,no. 3, Sept. 1991, pp. 536-545.

[4] Mo-yuen Chow, Robert N. Sharpe and James C. Hung,"On the Application and Design Consideration of Artificial Neural Network Fault Detectors-Part II," *IEEE Transaction on Industrial Electronics,* vol. 40, no.2, April, 1993,pp. 189-196.

[5] Mo-Yuen Chow, Griff Bilbro and Sui Oi Yee,"Application of Learning Theory to an Artificial Neural Network that Detects Incipient Faults in Single Phase Induction Motors," *International Journal of Neural System,* vol.2, No.1&2, 1991, pp. 91-100.

[6] Chin-Teng and C.S. George Lee, "Neural-network based Fuzzy Logic Control and Decision System," *IEEE Trans. on Computers*, vol. 40, No. 12, March 1991, pp. 1320-1335.

[7] Timo Sorsa and Heikki N.Koivo,"Neural Network in Process Fault Diagnosis," *IEEE Trans. on System, Man, and Cybernetics*, vol. 21, no. 4, March 1991, pp. 815-825.

[8] Petri A. Jokinen, "Comparison of Neural Network Models for Process Fault Detection and Diagnosis Problems," *IJCNN Seatle'91*, vol. 1, pp. 239-244.

[9] Paul V. Goode and Mo-yuen Chow, "Neural/Fuzzy Systems for Incipient Fault Detection in Induction Motors," *IECON'93* vol.1,pp.332-337.

[10] Mo-Yuen Chow, Peter M. Mangum and Sui Ou Yee ,"A Neural Network approach to Real-time Condition Monitoring of Induction Motors," *IEEE Transaction on Industrial Electronics*,vol. 38, no. 6, Dec. 1991, pp. 448-453.

[11] Mo-yuen Chow, Robert N. Sharpe and James C. Hung,"On the Application and Design Consideration of Artificial Neural Network Fault Detectors-Part I," *IEEE Transaction on Industrial Electronics*, vol. 40, no.2, April, 1993, pp. 181-188.

[12] M.F. Abdel Mageed, A.F. Sakr, A.Bahgat,"Fault Detection and Identification using Hierarchical Neural Network-based System," *IECON'93* vol. 1, pp. 338-342.

[13] F. Filippetti and M.Martelli,"Neural Networks Aided On-line Diagnostics of Induction Motor Rotor Faults," *IAS Conf. record*, 1993, pp. 316-323.

[14] F. Filippetti and M.Martelli, "Development of Expert System Knowledge-base to On-line Diagnosis of Rotor Electrical Faults of Induction Motors," *IAS Conf. record*, 1992, pp. 92-99.

[15] R. Natarajan,"Failure Identification of Induction Motors by Sensing Unbalanched Stator Currents ," *IEEE Transaction on Energy Conversion,* vol. 4, no.4, December, 1989, pp. 585-590.

[16] C.A. Protopapas, S.D. Kaminaris, AV. Machias and B.C. Papadias,"An Expert System for Fault Repairing and Maintenance of Electraical Machines. ," *IEEE Transaction on Energy Conversion,* vol. 5, no.1, March, 1990, pp. 79-83.

[17] Simon Haykin,*Neural Networks : A Comprehensive Foundation,* Macmillan College Publishing Company, USA, 1994.

[18] James A.Freeman and David M.Skapura,*Neural Networks Algorithm, Application and Training Techniques,* Addition-Welsy Publishing Company, 1991.

[19] Gail A.Carpenter and Stephen Grossberg,"Invarient pattern recognition and recall by an attentive self-organizing ART architecture in a nonstationary world," *Proceedings of the IEEE First International Conference on Neural Networks,* vol. II, June 1987, pp. 737-745.

[20] Gail A.Carpenter and Stephen Grossberg,"ART2: Self-organization of stable category recognition codes for analog input patterns," *Proceedings of the IEEE First International Conference on Neural Networks*, vol. II, June 1987, pp. 727-735.

[21] Alan V.Oppenhein and Ronald W.Schafer,*Discrete-Time Signal Processing*, Prentice-Hall,Inc,New Jersey 07632,1989.

[22] *DSP56000/DSP56001-Dsigital signal Processor : User's Manual*, Motorola, 1990.

[23] *Operating Manual for the PC-56 : DSP Co-processor Board*, Ariel Corporation, 1989.

[24] *DSP Developer's Toolkit for the Motorola DSP56000/DSP56001*, Ariel Corporation, 1989.

[25] Toshio Fukuda, Koji Shimojima,"Multi-Sensor Integration System with Fuzzy Interference and Neural Networks," *International Joint Conference on Neural networks*, vol. 11, 1992, pp. 757-762.

APPENDIX 'A' Program listing for ART2 Neural Network.

This is the program for implementation of ART2 neural network. Art2 has been implemented using object oriented software development methodology. So it can be used as a class library to add its feature to application program. Dynamic memory operation has been used to handle all matrices, so virtually the size of input pattern is determined by the memory of the system. It also includes an user interface routine to train and test a neural network. It has been written in Borland C++ under O/S MS-DOS for IBM PC environment.

Mohd. Rokonuzzaman.

```
#include <stdlib.h>
#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <alloc.h>
#include <math.h>

#define TRAIN 10
#define RECALL 11
#define DEFINE 12
#define QUIT 13

const features=5;
const type=4;
const DBSIZE=10;

int train_net(char *fname,int fea,int type,char *fname1);
int recall_net(char *fname,char *result);
int menu();
```

```
/*****************************************************************************/
/************************** Decleration of art2 class ************************/
/*****************************************************************************/

class art2{
        private:
        float avf,bvf,cvf,tvf,dvf,evf,rovf;
        float *uwpf,*dwpf;
        int *f2n;
        int Nvi,Mvi;

        public:
        art2(float aavf=10.0,float abvf=10.0,float acvf=.1,float atvf=.2,float advf=.9,float
        aevf=0.0,
          int aMvi=features,int aNvi=type,float arovf=1.0);
        ~art2();
        int train(float *I);
        int recall(float *I);
        int show_uw(FILE *fp);
        int show_dw(FILE *fp);
        int add_neuron();
        int define(int fea,int type);
        int parameters(float e,float ro);
        int give_fea_type(int *M,int *N);
        };

art2  oart2;

/*****************************************************************************/
/********************** An interface to train and test the Network ***********/
/*****************************************************************************/

main()
  {
  int N,M,Uo,c;
  float e,ro;
  char fname[10],fname1[10];

  Uo=100;
  while(Uo !=QUIT)
        {
        menu();
        c=getch();
```

```
        if(c=='t' || c=='T' || c=='1')Uo=TRAIN;
        if(c=='r' || c=='R' || c=='2')Uo=RECALL;
        if(c=='d' || c=='D' || c=='3')Uo=DEFINE;
        if(c=='q' || c=='Q' || c=='4')Uo=QUIT;
        switch(Uo)
                {
                case TRAIN:
                        printf("\n\n\n");
                        printf("\n          Pattern' features   =");scanf("%d",&M);
                        printf("\n          Classification Types=");scanf("%d",&N);
                        printf("\n          Pattern File name   =");scanf("%s",fname);
                        printf("\n          Weight File name    =");scanf("%s",fname1);
                        train_net(fname,M,N,fname1);
                        break;

                case RECALL:
                        printf("\n\n\n\n\n\n\n" );
                        printf("\n          Recall Pattern File name   =");scanf("%s",fname);
                        printf("\n          Recall result File name    =");scanf("%s",fname1);
                        recall_net(fname,fname1);
                        break;
                case DEFINE:
                        printf("\n          Please Enter value of 'e' =");scanf("%f",&e);
                        printf("\n          Please Enter value of 'ro'=");scanf("%f",&ro);
                        oart2.parameters(e,ro);
                        break;
                case QUIT:
                        Uo=QUIT;
                        printf("\n\n\n\          **** Thanks a lot **** ");
                        break;
                }
        }

 return 1;
 }

/*****************************************************************************/
/************************** Constructor ************************************/
/*****************************************************************************/

art2::art2(float aavf,float abvf,float acvf,float atvf,float advf,float aevf,int aMvi,int aNvi,float
        arovf)
        {
        avf=aavf;
        bvf=abvf;
        cvf=acvf;
        tvf=atvf;
        dvf=advf;
        evf=aevf;
```

```
        Mvi=aMvi;
        Nvi=aNvi;
        rovf=arovf;

        uwpf=(float *)farmalloc(sizeof(float)*Mvi*Nvi);
        dwpf=(float *)farmalloc(sizeof(float)*Mvi*Nvi);
        f2n=(int *)farmalloc(sizeof(int)*Nvi);

        float uiwvf,diwvf;
        uiwvf=2.236;
        diwvf=0;
        for(int i=0;i<Nvi;++i)
                {
                for(int j=0;j<Mvi;++j)
                        *(uwpf+i*Mvi+j)=uiwvf;
                f2n[i]=0;
                }
        for(i=0;i<Mvi;++i)
                {
                for(int j=0;j<Nvi;++j)
                        *(dwpf+i*Nvi+j)=diwvf;
                }
        }

/*****************************************************************************/
/********************************** Destructor *******************************/
/*****************************************************************************/
art2::~art2()
                {
                farfree(uwpf);
                farfree(dwpf);
                }

/*****************************************************************************/
/******************** Display and store bottom up the Weight *****************/
/*****************************************************************************/

int art2::show_uw(FILE *fp)
                {
                printf("\n");
                for(int i=0;i<Nvi,++i)
                        {
                        printf("\n");
                        for(int j=0;j<Mvi;++j)
                                printf( "%f ",*(uwpf+i*Mvi+j) );
                        }
                fprintf(fp,"\n \n Bottom up weight:\n");
                for( i=0;i<Nvi;++i)
                        {
```

```
                fprintf(fp,"\n");
                for(int j=0;j<Mvi;++j)
                        fprintf(fp, "%f ",*(uwpf+i*Mvi+j) );
                }
        return 1;
        }

/*************************************************************************/
/******************** Display and store the top down  Weight *************/
/*************************************************************************/

int art2::show_dw(FILE *fp)
        {
        printf("\n");
        for(int i=0;i<Nvi;++i)
                {
                printf("\n");
                for(int j=0;j<Mvi;++j)
                        printf( "%f ",*(dwpf+i*Mvi+j) );
                }
        fprintf(fp,"\n \n Top_down weight:\n");
        for(i=0;i<Nvi;++i)
                {
                fprintf(fp,"\n");
                for(int j=0;j<Mvi;++j)
                        fprintf(fp, "%f ",*(dwpf+i*Mvi+j) );
                }
        return 1;
        }

/*************************************************************************/
/********************** Train the Net with define  Pattern ***************/
/*************************************************************************/

int art2::train(float *I)
        {
        int counter,J,*f2,i13,i14=0;
        float *u,*v,*w,*x;
        float *q,*qb,*r,*T;
        float *p,*g;
        float *fx,*fq;

        float nv,nvp,test1,test2;                       //normalized value
        int eq,reso;

        J=0;
        f2=(int* )farmalloc(sizeof(int)*Nvi);

        u=(float *)farmalloc(sizeof(float)*Mvi);
```

```
        v=(float *)farmalloc(sizeof(float)*Mvi);
        w=(float *)farmalloc(sizeof(float)*Mvi);
        x=(float *)farmalloc(sizeof(float)*Mvi);
        q=(float *)farmalloc(sizeof(float)*Mvi);
        qb=(float *)farmalloc(sizeof(float)*Mvi);
        r=(float *)farmalloc(sizeof(float)*Mvi);
        T=(float *)farmalloc(sizeof(float)*Mvi);
        p=(float *)farmalloc(sizeof(float)*Mvi);
        g=(float *)farmalloc(sizeof(float)*Mvi);
        fx=(float *)farmalloc(sizeof(float)*Mvi);
        fq=(float *)farmalloc(sizeof(float)*Mvi);
        //1.
        counter=1;
 for(int i=0;i<Mvi;++i)
        {
        u[i]=0.0; v[i]=0.0; w[i]=0.0;x[i]=0.0;
        q[i]=0.0;qb[i]=0.0;r[i]=0.0;T[i]=0.0;
        p[i]=0.0;g[i]=0.0;
        }
  for(i=0;i<Nvi;++i)
         f2[i]=1; i13=0;
for(;;)
  {do
        {
        do{
         if(i13==0)
          {
          for(i=0;i<Mvi;++i)
                w[i]=I[i]+avf*u[i];
          nv=0.0;
          for(i=0;i<Mvi;++i)
                nv=nv+w[i]*w[i];
          nv=sqrt(nv);
          for(i=0;i<Mvi;++i)
                x[i]=w[i]/(evf+nv);
          for(i=0;i<Mvi;++i)
                 {
                 if(x[i] >tvf)fx[i]=x[i]; else fx[i]=0.0;
                 fq[i]=q[i];
                 v[i]=fx[i]+bvf*fq[i];
                 }
          nv=0;
          for(i=0;i<Mvi;++i)
                nv=nv+v[i]*v[i];
          nv=sqrt(nv);
          for(i=0;i<Mvi;++i)
                u[i]=v[i]/(evf+nv);
         }
        for(i=0;i<Mvi;++i)
```

```
        p[i]=u[i]+dvf* *(dwpf+J*Mvi+i);

       nv=0;
       for(i=0;i<Mvi;++i)
        nv=nv+p[i]*p[i];
       nv=sqrt(nv);
       for(i=0;i<Mvi;++i)
        q[i]=p[i]/(evf+nv);
      eq=1;
      for(int i=0;i<Mvi;++i)
        {
        if(qb[i] !=q[i])eq=0;
         qb[i]=q[i];
        }
      }
while(eq==0);

      nv=0;nvp=0;
      for(i=0;i<Mvi;++i)
       nv=nv+u[i]*u[i];
      nv=sqrt(nv);

      for(i=0;i<Mvi;++i)
       nvp=nvp+p[i]*p[i]*cvf*cvf;
      nvp=sqrt(nvp);

      for(i=0;i<Mvi;++i)
     r[i]=(u[i]+cvf*p[i])/(evf+nv+nvp);

      nvp=0;
      for(i=0;i<Mvi;++i)
       nvp=nvp+r[i]*r[i];
      nvp=sqrt(nvp);
      test1=rovf/(evf+nvp);//printf("\n %f",test1);
      if(test1 >1)
       {
       f2[J]=0;
       counter=1;
       ++J;
       if(J==Nvi)return -1;
       }
      }
      while(test1 >1);

     if(counter >1)i14=1;

     if(counter==1)
      {
      ++counter;
```

```
        for(int j=0;j<Nvi;++j)
                        {
                        T[j]=0.0;
                         for(i=0;i<Mvi;++i)
                                T[j]+=p[i]* *(uwpf+j*Mvi+i);
                        T[j]=T[j]*f2[j];
                        }

                       j=0;
                       for(j=0;j<Nvi;++j)
                         if(T[J] <T[j])J=j;
                       for(j=0;j<Nvi;++j)
                         {
                         if(j==J)
                                T[j]=dvf*T[j];
                         else T[j]=0.0;
                         }
         }
        i13=1;
        if(i14==1)break;
        }

       for(i=0;i<Mvi;++i)
         *(uwpf+J*Mvi+i)=u[i]/(1-dvf);

       for(i=0;i<Mvi;++i)
         *(dwpf+J*Mvi+i)=u[i]/(1-dvf);
        f2n[J]=1;
        farfree(f2);farfree(u);farfree(v);
        farfree(w);farfree(x);farfree(q);
        farfree(qb);farfree(r);farfree(T);
        farfree(p);farfree(g);farfree(fx);
        farfree(fq);
        return J;
        }

/*****************************************************************************/
/****************************** Pattern Matching *****************************/
/*****************************************************************************/

int art2::recall(float *I)
        {
        int counter,*f2,J,i13,i14=0;

        float *u,*v,*w,*x;
        float *q,*qb,*r,*T;
        float *p,*g;
        float *fx,*fq;
        float nv,nvp,test1,test2;                    //normalized value
```

```
        int eq,reso;

        J=0;
        f2=(int *)farmalloc(sizeof(int)*Nvi);
        u=(float *)farmalloc(sizeof(float)*Mvi);
        v=(float *)farmalloc(sizeof(float)*Mvi);
        w=(float *)farmalloc(sizeof(float)*Mvi);
        x=(float *)farmalloc(sizeof(float)*Mvi);
        q=(float *)farmalloc(sizeof(float)*Mvi);
        qb=(float *)farmalloc(sizeof(float)*Mvi);
        r=(float *)farmalloc(sizeof(float)*Mvi);
        T=(float *)farmalloc(sizeof(float)*Mvi);
        p=(float *)farmalloc(sizeof(float)*Mvi);
        g=(float *)farmalloc(sizeof(float)*Mvi);
        fx=(float *)farmalloc(sizeof(float)*Mvi);
        fq=(float *)farmalloc(sizeof(float)*Mvi);
        counter=1;
 for(int i=0;i<Mvi;++i)
        {
        u[i]=0.0;v[i]=0.0;w[i]=0.0;
        x[i]=0.0;q[i]=0.0;qb[i]=0.0;
        r[i]=0.0;T[i]=0.0;p[i]=0.0;
        g[i]=0.0;
        }
 for(i=0;i<Nvi;++i)
        f2[i]=1;
i13=0;
for(;;)
 {
 do{
    do{
       if(i13==0)
         {
         for(i=0;i<Mvi;++i)
            w[i]=I[i]+avf*u[i];
         nv=0.0;
         for(i=0;i<Mvi;++i)
               nv=nv+w[i]*w[i];
         nv=sqrt(nv);
         for(i=0;i<Mvi;++i)
               x[i]=w[i]/(evf+nv);
         for(i=0;i<Mvi;++i)
               {
               if(x[i] >tvf)fx[i]=x[i]; else fx[i]=0.0;
               fq[i]=q[i];
               v[i]=fx[i]+bvf*fq[i];
               }
         nv=0;
         for(i=0;i<Mvi;++i)
```

```
                nv=nv+v[i]*v[i];
            nv=sqrt(nv);
            for(i=0;i<Mvi;++i)
                u[i]=v[i]/(evf+nv);
           } //i13=0;
           for(i=0;i<Mvi;++i)
            p[i]=u[i]+dvf* *(dwpf+J*Mvi+i);
           nv=0;
           for(i=0;i<Mvi;++i)
            nv=nv+p[i]*p[i];
           nv=sqrt(nv);
           for(i=0;i<Mvi;++i)
            q[i]=p[i]/(evf+nv);
          eq=1;
          for(int i=0;i<Mvi;++i)
          {
          if(qb[i] !=q[i])eq=0;
          qb[i]=q[i];
          }
          } //8.
while(eq==0);

          nv=0;nvp=0;
          for(i=0;i<Mvi;++i)
            nv=nv+u[i]*u[i];
          nv=sqrt(nv);

          for(i=0;i<Mvi;++i)
            nvp=nvp+p[i]*p[i]*cvf*cvf;
          nvp=sqrt(nvp);

          for(i=0;i<Mvi;++i
         r[i]=(u[i]+cvf*p[i])/(evf+nv+nvp);

          //10.
           nvp=0;
          for(i=0;i<Mvi;++i)
            nvp=nvp+r[i]*r[i];
          nvp=sqrt(nvp);
          test1=rovf/(evf+nvp);//printf("\n %f",test1);
          if(test1 >1)
            {
            f2[J]=0;
            counter=1;
            ++J;
            if(J==Nvi)return -1;
            }
          }
          while(test1 >1);
```

```
        if(counter >1)i14=1;

        if(counter==1)
         {
                ++counter;
               //11.
                for(int j=0;j<Nvi;++j)
                        {
                        T[j]=0.0;
                         for(i=0;i<Mvi;++i)
                                T[j]+=p[i]* *(uwpf+j*Mvi+i);
                        T[j]=T[j]*f2[j];
                        }
               //12.
                       j=0;
                       for(j=0;j<Nvi;++j)
                         if(T[J] <T[j])J=j;
                       for(j=0;j<Nvi;++j)
                         {
                         if(j==J)
                                T[j]=dvf*T[j];
                         else T[j]=0.0;
                         }
         }
        i13=1;
        if(i14==1)break;
        }
        if(f2n[J] ==0)J= -1;
        farfree(f2);farfree(u);farfree(v);
        farfree(w);farfree(x);farfree(q);
        farfree(qb);farfree(r);farfree(T);
        farfree(p);farfree(g);farfree(fx);
        farfree(fq);
        return J;
        }

/*****************************************************************************/
/******************** Add a New Neuron to classify more classes ******************/
/*****************************************************************************/

int art2::add_neuron()
         {
         float *nuwpf,*ndwpf;
         int *nf2n;

        ++Nvi;
        nuwpf=(float *)farmalloc(sizeof(float)*Mvi*Nvi);
        ndwpf=(float *)farmalloc(sizeof(float)*Mvi*Nvi);
```

```
        nf2n=(int *)farmalloc(sizeof(int)*Nvi);


        float uiwvf,diwvf;
        uiwvf=2.236;
        diwvf=0;
        for(int i=0;i<Nvi-1;++i)
                {
                for(int j=0;j<Mvi;++j)
                        *(nuwpf+i*Mvi+j)=*(uwpf+i*Mvi+j);
                nf2n[i]=f2n[i];
                }
         nf2n[i]=0;
         farfree(f2n);
         f2n=nf2n;

         for(int j=0;j<Mvi;++j)
                 *(nuwpf+i*Mvi+j)=uiwvf;
         farfree(uwpf);
         uwpf=nuwpf;


         for(i=0;i<Nvi-1;++i)
                        {
                        for(int j=0;j<Mvi;++j)
                                *(ndwpf+i*Mvi+j)=*(dwpf+i*Mvi+j);
                        }
         for( j=0;j<Mvi;++j)
                 *(ndwpf+i*Mvi+j)=diwvf;

         farfree(dwpf);
         dwpf=ndwpf;

        return 1;
        }

/*****************************************************************************/
/************** Define the Net by features and type of classes ***************/
/*****************************************************************************/

int art2::define(int fea,int type)
        {
         oart2.Mvi=fea;
         oart2.Nvi=type;

        farfree(uwpf);
        farfree(dwpf);
        farfree(f2n);
```

```
        uwpf=(float *)farmalloc(sizeof(float)*Mvi*Nvi);
        dwpf=(float *)farmalloc(sizeof(float)*Mvi*Nvi);
        f2n=(int *)farmalloc(sizeof(int)*Nvi);

        float uiwvf,diwvf;
        uiwvf=2.236;
        diwvf=0;
        for(int i=0;i<Nvi;++i)
                {
                for(int j=0;j<Mvi;++j)
                        *(uwpf+i*Mvi+j)=uiwvf;
                f2n[i]=0;
                }
        for(i=0;i<Mvi;++i)
                {
                for(int j=0;j<Nvi;++j)
                        *(dwpf+i*Nvi+j)=diwvf;
                }
         return 1;
         }

/*****************************************************************************/
/************************** Define generalization parameters *****************/
/*****************************************************************************/

int art2::parameters(float e,float ro)
        {
        evf=e;
        rovf=ro;
        return 1;
        }

/*****************************************************************************/
/****************** Give the values of feature and Type **********************/
/*****************************************************************************/

int art2::give_fea_type(int *M,int *N)
        {
        *M=Mvi;
        *N=Nvi;
        return 1;
        }
```

```
/*****************************************************************************/
/*********************** User interface to train the Net *********************/
/*****************************************************************************/

int train_net(char *fname,int fea,int type,char *fname1)
        {
        FILE *fp;
        float data;
        long int handle,fsize;
        int i,j,f;
        char *buff,vbuff[DBSIZE];
        float *pattern;
        int M,N;

        oart2.give_fea_type(&M,&N);
        handle=open(fname,0);
        fsize=filelength(handle);
        buff=(char *)farmalloc(fsize);
        pattern=(float *)farmalloc(sizeof(float)*fea);
        close(handle);
        fp=fopen(fname,"rb+");if(fp==NULL)return -1;
        fread(buff,1,fsize,fp);

        for(i=0;i<fsize;++i)printf("%c",buff[i]);printf("\n\n");

        oart2.define(fea,type);

        f=0;
        for(i=0;i<fsize;++i)
                {
                while(buff[i]==32)++i;
                if(i==fsize)break;
                if(buff[i] !=13)
                        {
                        j=0;
                        for(;;)
                        {
                        vbuff[j]=buff[i];
                        ++j;++i;
                        if(buff[i]==13 || buff[i]==32)break;
                        if(i==fsize)break;
                        if(j==DBSIZE)break;
                        }
                        vbuff[j]='\0';
                        data=atof(vbuff);

                        if(f <M)
                        {
```

```
                        pattern[f]=data;
                        ++f;
                        }
                         if(buff[i]==13)
                        {
                        while(f !=M){pattern[f]=0.0;++f;}
                        ++i;f=0;
                        oart2.train(pattern);
                        }
                         }
                else {++i; printf("\n ");
                }
                 }
          farfree(buff);
          farfree(pattern);
          fclose(fp);

         fp=fopen(fname1,"wb+");
         oart2.show_uw(fp);
         oart2.show_dw(fp);
         fclose(fp);
         return 1;
         }

/*************************************************************************************/
/******************** User interface for pattern classification **********************/
/*************************************************************************************/

int recall_net(char *fname,char *result)
         {
         FILE *fp,*fpr;
         float data;
         long int handle,fsize;
         int i,j,f,rtype,patno;
         char *buff,vbuff[DBSIZE];
         float *pattern;
         int M,N;

         oart2.give_fea_type(&M,&N);
         handle=open(fname,0);
         fsize=filelength(handle);
         buff=(char *)farmalloc(fsize);
         pattern=(float *)farmalloc(sizeof(float)*M);
         close(handle);
         fp=fopen(fname,"rb+");if(fp==NULL)return -1;
         fread(buff,1,fsize,fp);


        // for(i=0;i<fsize;++i)printf("%c",buff[i]);printf("\n\n");
```

```
fpr=fopen(result,"wb+");
fprintf(fpr," Pattern Matching Result Of ART2\n\n");
f=0;patno=0;

for(i=0;i<fsize;++i)
        {
        while(buff[i]==32)++i;
        if(i==fsize)break;
        if(buff[i] !=13)
                {
                j=0;
                for(;;)
                {
                vbuff[j]=buff[i];
                ++j;++i;
                if(buff[i]==13 || buff[i]==32)break;
                if(i==fsize)break;
                if(j==DBSIZE)break;
                }
                vbuff[j]='\0';
                data=atof(vbuff);

                if(f <M)
                {
                pattern[f]=data;
                ++f;
                }
                if(buff[i]==13)
                {
                ++patno;
                while(f !=M){pattern[f]=0.0;++f;}
                ++i;f=0;
                rtype=oart2.recall(pattern);
                fprintf(fpr,"\n");
                if(rtype==-1)
                fprintf(fpr,  "No Matched pattern of pattern No= %d  ",patno);
                else fprintf(fpr,"%d is the Matched  pattern of pattern No=
                        %d",rtype+1,patno);
                }
                }
        else {++i; printf("\n ");
        }
        }
farfree(buff);
farfree(pattern);
fclose(fp);
fclose(fpr);
return 1;
}
```

```
/*****************************************************************************/
/****************************** User interface main menu *********************/
/*****************************************************************************/

int menu()
        {
        clrscr();
        printf("\n\n\n\n\n\n\n          **** TEST  OF  ART2 ****");

        printf("\n\n\n                    1. Training");
        printf("\n                  2. Recall");
        printf("\n                  3. Define Parameters");
        printf("\n                  4. Quit");
        return 1;
        }

/*****************************************************************************/
/****************************************** END ******************************/
/*****************************************************************************/
```

APPENDIX "B" Program listing for real-time frequency spectra acquisition..

This program is for real-time frequency spectra collection of the stator current of an induction motor using digital signal processing(DSP) technique.

```
#define MAX_COM 80

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <graph.h>
#include <math.h>

#include "pc56.h"        /* header fie for global pc56 declarations */
#include "pc56ext.h"     /* external variable definitions           */
                         /* defined in PC-56 library (init_56.c)    */
#include "fft_56.h"      /* fft_56 definitions   */
#include "demogrph.h"    /* graphics definitions */
#include "printer.h"
#include "plotter.h"

extern unsigned long int DATA;          /* data array          */
extern unsigned long int COEF;          /* sin/cos table       */
extern unsigned long int WINDCOF;       /* window coefficients */
extern long int maxN;                   /* maximum data size   */

/*****************************************************************************/
/************************** Function prototypes ******************************/
/*****************************************************************************/

void initgrph(long int);
void demoplot(int [], int [], int, int, int);
void pause(void);

/*****************************************************************************/
/* ************************Global data arrays for transfers, etc ************************/
/*****************************************************************************/

float fp;
int arrays1[2048], arrays2[2048],in;
char c,xstep=1;
int parray[514],diff,counter=0,parrays2[514],dis_counter=0,signal,avcycle=1;
int fre[100],amp[100];
char tbuff[50];
```

```
main()
{
  struct dostime_t  start, finish;

  char key,tbuff[128];
  int errcode,j,max;
  int points, endloop, count, dspopt, plotopt;
  long int n, l, fsamp, space, presample;
  int bpfilt, gain;
  register int i;
  int option, timopt, comp_flag, word_size, usr_flag;
  unsigned long int retcode;

/* Initialize variables, etc */

   long int x_memflag = 0L;
   long int y_memflag = 1L;

/* Initialize Essential graphics text mode */


   _setvideomode(_ERESCOLOR);

n = 1024L;
l = 513L;
fsamp = 800L;

bpfilt = 1;
gain = 4L;
timopt = 0;
plotopt = 0;
 dspopt = 1;

/* Initialize fft_56 program */

  errcode = init_fft(&n, &l);
  if(errcode != no_error){
        printf("\aFATAL ERROR %d - cannot initialize\n",errcode);
        exit(1);
       }

/* Initialize AIC parameters */

  errcode = init_aic(&fsamp, bpfilt, gain);
  if(errcode != no_error){
        printf("\aFATAL ERROR %d - cannot initialize\n",errcode);
        exit(1);
        }
```

```
/* Initialize AIC converter */
  errcode = cmd_56(adinit);
  if(errcode == err_hostcmd)
        printf("\aERROR %d Unable to initialize AIC\n",errcode);
/*  Load user command fft routine */

  errcode = load_usr("demofft.lod",usr);
  if(errcode != no_error)
        printf("ERROR %d in loading user command\n",errcode);

  points = l;             /* comvert to integer */

_reset:

  if(!dspopt){
        errcode = cmd_56(sample);
        for(i=0; i<n; i++)
          arrays2[i] = 0;
        }
  else{
        errcode = cmd_56(usr);        /* Execute on time to fill array*/
        for(i=0; i<n; i++)
          arrays2[i] = graph_ulx;
     }

/*  Sample and display until keyboard is hit */

  count = 0;                  /* initialize count */
  if(timopt){
        endloop = 10;             /* initialize loop count if timing */
        _dos_gettime(&start);
  }

  do{

        errcode = dwnarras(arrays1, l, X_SPACE, DATA);

/*  Start process, display arrays1, erase arrays2 */

        if(dspopt)
          errcode = hostcmd(usr);
        else
          errcode = hostcmd(sample);        /* start process */

/*!      demoplot(arrays1,arrays2,points,yellow,plotopt); */

/*  When complete, download into arrays1 */

        while(!dsp_done()){    /* wait for 56000 code to be finished */
```

```
        }
        rcv_hp(&retcode);
        errcode = retcode;          /* conversion to int */

        if(errcode != no_error)
          exit(1);

        errcode = dwnarras(arrays2, 1, X_SPACE, DATA);

/*  Start process for next array, display arrays2, erase arrays1 */

        if(dspopt)
          errcode = hostcmd(usr);
        else
          errcode = hostcmd(sample);      /* start process */

      for(i=0;i<11;++i)
          {
          _settextcolor(1);
          sprintf(tbuff,"%d ",i);
          _settextposition(25,8+i*5+i/4+i/5);
          _outtext(tbuff);
          _setcolor(15);
          _moveto(48+i*45,349);
          _lineto(48+i*45,340);
          _moveto(48+i*45+22,349);
          _lineto(48+i*45+22,345);
          }
        _setcolor(6);
        _rectangle(2,0,90,639,349);
        _setcolor(7);
        _moveto(48,95);
        _lineto(48,335);
        _lineto(638,335);
        _setcolor(14);
        _settextcolor(14);


       for(i=0;i<points;++i)
        {
        if(arrays2[i] >310)arrays2[i]=310;
        arrays2[i]=310-arrays2[i];
        }
       for(i=0;i<5;++i)arrays2[i]=0;

      max=0;
    for(j=0;j<points;++j)
          {
          if(arrays2[max] < arrays2[j])
```

```
            max=j;
          }
for(j=0;j<points;++j)
          {
          if(arrays2[j] < (arrays2[max])/5 )
          arrays2[j]=0;
          }

    for(i=0;i<points;++i)
     {
     if(arrays2[i] !=0)
      {
      max=i;
      for(j=i;j<i+25;++j)
       {
       if(arrays2[max] < arrays2[j])
            max=j;
       }
      for(j=i;j<i+25;++j)
       {
       if(j!=max)
            arrays2[j]=0;
       }
      i+=24;
      }
     }

                                                    /* Draw the Spectra on the video Screen */
    for(i=0;i<points;++i)
      {
       _setcolor(0);
       _moveto(49+i*xstep,334);
        _lineto(49+i*xstep,92);
    _setcolor(14);
       _moveto(48+i*xstep,334);
       if(arrays2[i] !=0)
       _lineto(48+i*xstep,334-arrays2[i]);
       }


_settextcolor(1);
j=7;
for(i=0;i<8;++i)
     {
     sprintf(tbuff,"-%d0db",j);
     _settextposition(24-i*2-i/4,2);
     _outtext(tbuff);
  _setcolor(5);
     _moveto(47,335-i*31);
```

```
        _lineto(37,335-i*31);
        _moveto(47,335-i*31-15);
        _lineto(42,335-i*31-15);
        --j;
        }
      _settextcolor(14);
_setcolor(5);
_rectangle(3,0,0,639,88);
j=0;
for(i=0;i<points;++i)
  {
  if(arrays2[i] !=0)
       {
       fp=i;
       fp=fp*2.4;
       in=fp;
       if(in%60 <17)
         in=in-in%60;
       if(in%60 >50)
         in=in+(60-in%60);
       sprintf(tbuff,"%d=%d",in,arrays2[i]);
       _settextposition(4+j/7,5+(j%7)*8);
   _outtext(tbuff);
       ++j;
       }
  }


  if(kbhit() !=0){
                c=getch();
                if(c=='p')
                gettext();
                }

/*
      diff=0;signal=0;

      for(i=0;i<points;++i)
         {
          if(dis_counter >0)
           {
               _moveto(10+i*xstep,339);
               _lineto(10+i*xstep,arrays2[i]);
           }
          diff+=abs(parray[i]-arrays2[i]);
          signal+=310-arrays2[i];
          parray[i]=arrays2[i];
          sprintf(tbuff,"%d ",310-arrays2[i]);
          if(i<15)
```

```
            _settextposition(7,0+i*3);
            else if(i >=15 && i<30) _settextposition(7+1,0+(i-15)*5);
            else if(i >=30 && i<45) _settextposition(7+2,0+(i-30)*5);
            else _settextposition(7+3,0+(i-45)*5);
            _outtext(tbuff);
            }
           signal+=1.0;
           sprintf(tbuff,"Error=%d  Signal=%d  per. of Error=%d
.....",diff,signal,((diff*100)/signal));

           _settextcolor(14);
           _setcolor(0);
           _rectangle(3,0,0,100,20);
           _settextposition(0,0);
           if(dis_counter >1)
           _outtext(tbuff);

    if(kbhit() !=0){
                    c=getch();
                    if(c=='q'){_setvideomode(_DEFAULTMODE); exit(1)}
                    if(c=='p')
                    gettext();
                    getch();
                    }

  counter=0;
  ++dis_counter;
  for(i=0;i<points;++i)
       parrays2[i]=0;
  }  */


/*  When complete, download into arrays1 (at top of loop) */

       while(!dsp_done()){    /* wait for 56000 code to be finished */
       }
       rcv_hp(&retcode);
       errcode = retcode;          /* conversion to int */

       if(errcode != no_error)
         exit(1);

       count++;                  /* increment counter */

       if (timopt){
         endloop = endloop-1;
       }
       else
         endloop = !kbhit();
```

```
    }
    while(endloop);
    if(timopt)
        _dos_gettime(&finish);
    else{
        key = getch();                  /* get character */
        switch(key){
          case('t'):                    /* toggle display */
          case('T'):
                dspopt = !dspopt;
                goto _reset;
                break;
          case('f'                      /* freeze display */
          case('F'):
                while(!kbhit()){
                }
                key = getch();
                goto _reset;
                break;
          case('p'):                    /* graph with points */
          case('P'):
                /* plotopt = 0; */
                gettext();
                key=10;
                goto _reset;
                break;
          case('l'):                    /* graph with lines */
          case('L'):
                plotopt = 1;
                goto _reset;
                break;
          case('Q'):                    /* Quit display */
          case('q'):
          case('\x1B'):
                break;
          default:                      /* Ignore all other keys */
                goto _reset;
                break;
        }
    }

    if(timopt){
        printf("Start %d.%2.2d Finish %d.%2.2d\n",start.second,
                start.hsecond,finish.second,finish.hsecond);
        printf("Executed loop %d times, %d spectra displayed\n",
                count, 2*count);
    }
_setvideomode(_DEFAULTMODE);
}
```

```
/*****************************************************************************/
/***************************** Get User Option *******************************/
/*****************************************************************************/
int gettext()
   {
   float fp;
   int i,j,in;

    _settextposition(0,0);
   printf("Frequency spectrun of l of induction motor at  Condition: \n");
   scanf("%s",tbuff);


  init_serial();
  init_plotter();

  for(i=0;i<100;++i)
    fre[i]=-1;
  i=0;j=0;
  for(i=0;i<500;++i)
   {
   if(arrays2[i] >0)
        {
        fp=i;
        fp=fp*2.4; in=fp;
        if(in%60 <17)
          in=in-in%60;
        if(in%60 >45)
           in=in+(60-in%60);
        if(in <1000)
          {
          fre[j]=in;   fp=arrays2[i];
          fp=(fp/217)*70; amp[j]=fp;
          ++j;
          if(j>1)
            {
            if(fre[j-1]==fre[j-2])
              {
              if(amp[j-1] >amp[j-2]) amp[j-2]=amp[j-1];
              fre[j-1]=-1;  --j;
              }
            }
          }
        }
   if(j==MAX_COM-1)break;
   }
                                          /* Plot the spectra */
  plot_graph(fre,amp,"Rotor's phase   replaced by external resistor of 6 ohms");
  return 1;}
```

APPENDIX "C" Program listing of HP-plotter driver.

This is the program for Six pen HP-plotter driver to get real-time plot of frequency spectra of the stator current of an induction motor..

```
#define  RED   1
#define  GREEN  2
#define  BLACK  3
#define  PURPLE 4

#define RIGHT_MOST_X  10000
#define UP_MOST_Y     8500
#define DVELOCITY  7
#define MVELOCITY  38
#define DEVELOCITY   10

#include <stdio.h>
#include <dos.h>
#include<stdlib.h>


/*****************************************************************************/
/******************************** Plot The spectra ***************************/
/*****************************************************************************/

int plot_graph(int fre[],int amp[], char title[])
    {
    float famp;
    int i,j,value,x,y;
    int x1,y1,x2,y2,xdstep,xhdstep,xsstep,ydstep,yhdstep,ysstep,sdh,shdh,ssh,xdim,ydim;
    x1=1250; y1=2000; x2=7250; y2=6000;
    xdim=1000;ydim=70;
    xdstep=100;xhdstep=50;
    xsstep=10;ydstep=10;
    yhdstep=5;ysstep=1;
    sdh=120;shdh=80;
    ssh=30;

    edge_rect(x1,y1,x2,y2);
    plot_absolute(x1,y1);
    for(i=0;i<(xdim/xdstep);++i)
         {
         plot_lineto(0,sdh);
         x=(x2-x1)/(xdim/xdstep);
         y=-sdh;
         plot_relative(x,y);
         }
```

```
 plot_absolute(x1,y1);
x=x1;
for(i=0;i<(xdim/xhdstep);++i)
      {
      plot_lineto(0,shdh);
      x+=(x2-x1)/(xdim/xhdstep);
      y=-shdh;
      plot_absolute(x,y1);
      }
plot_absolute(x1,y1);
for(i=0;i<(xdim/xsstep);++i)
      {
      plot_lineto(0,ssh);
      x=(x2-x1)/(xdim/xsstep);
      y=-ssh;
      plot_relative(x,y);
      }

plot_absolute(x1,y1);
for(i=0;i<(ydim/ydstep);++i)
      {
      plot_lineto(sdh,0);
      x=-sdh;
      y=(y2-y1)/(ydim/ydstep);
      plot_relative(x,y);
      }
plot_absolute(x1,y1);
for(i=0;i<(ydim/yhdstep);++i)
      {
      plot_lineto(shdh,0);
      x=-shdh;
      y=(y2-y1)/(ydim/yhdstep);
      plot_relative(x,y);
      }
plot_absolute(x1,y1);
for(i=0;i<(ydim/ysstep);++i)
      {
      plot_lineto(ssh,0);
      x=-ssh;
      y=(y2-y1)/(ydim/ysstep);
      plot_relative(x,y);
      }
/*    select_pen(GREEN); */
plot_absolute(x1,y1);
plot_line(x1+(x2-x1)/4,y1-500,x1+(3*(x2-x1))/4,y1-500);
plot_line(x1+(x2-x1)/4,y1+10-500,x1+(3*(x2-x1))/4,y1+10-500);
plot_line(x1+(3*(x2-x1))/4,y1-500,x1+(3*(x2-x1))/4-200,y1-70-500);
plot_line(x1+(3*(x2-x1))/4,y1-500,x1+(3*(x2-x1))/4-200,y1+70-500);
```

```
    plot_line(x1-700,y1+(y2-y1)/6,x1-700,y1+(5*(y2-y1))/6);
    plot_line(x1-710,y1+(y2-y1)/6,x1-710,y1+(5*(y2-y1))/6);
    plot_line(x1-705,y1+(5*(y2-y1))/6,x1-705-70,y1+(5*(y2-y1))/6-200);
    plot_line(x1-705,y1+(5*(y2-y1))/6,x1-705+70,y1+(5*(y2-y1))/6-200);

/*   select_pen(RED); */
    i=0;
    while(fre[i] !=-1)
          {
          if(amp[i] <1000 && amp[i] >0)
            {
            x=x1+fre[i]*((x2-x1)/xdim);
            y=y1+amp[i]*((y2-y1)/ydim);
            plot_line(x,y1,x,y);
            plot_line(x+10,y1,x+10,y);
            }
         ++i;
         }
/*   select_pen(PURPLE); */
    i=0;
    plot_string(x2+500,y2,   "Frequency  Amplitude");
    plot_string(x2+500,y2-200," in Herz    in db");
/*   select_pen(BLACK); */
    while(fre[i] !=-1)
          {
          if(amp[i] <1000 && amp[i] >0)
            {
            x=x2+750;
            y=y2-i*250-500;
            plot_intval(x,y,fre[i]);
            x=x2+2000;
            famp=amp[i];
            plot_intval(x,y,-(70-amp[i]));
            }
         ++i;
         }
/*   select_pen(GREEN); */
    edge_rect(x2+400,y2+200,x2+2700,y2-500-i*250);
    edge_rect(x2+400,y2+200,x2+2700,y2-350);
/*   select_pen(BLACK); */

/*   select_pen(RED); */
    edge_rect(x1-1200,y1-800,x2+300,y2+800);
    edge_rect(x1-1215,y1-815,x2+310,y2+810);
    plot_absolute(12000,9200);getch();getch();
    relative_direction(0,1);
/*   select_pen(BLACK); */
    plot_string(x1-800,y1+(y2-y1)/6+150,"Frequency Amplitude ");
```

```
/*    select_pen(PURPLE); */
    for(i=0;i<8;++i)
          {
          value=-(7-i)*10;
          x=x1-520;
          y=y1+i*((y2-y1)/(ydim/ydstep))-100;
          plot_intval(x,y,value);
          x=x1-200;
          plot_string(x,y,"db");
          }

    for(i=0;i<(xdim/xdstep);++i)
          plot_intval(x1+i*((x2-x1)/(xdim/xdstep)),y1-150,i*100);
/*    select_pen(BLACK); */
    plot_string(x1+(x2-x1)/4+60,y1+80-500,"Spectral Frequency in Herz");
    plot_string(x1+300,y1-1100,title);
    return 1;
    }
/*****************************************************************************/
/******************************** Plot intiger value *************************/
/*****************************************************************************/

int plot_intval(int x,int y,int value)
          {
    int i;
    char ter=2;
    char buff[10];

    itoa(value,buff,10);
    pen_up();
    plot_absolute(x,y);
    write_serial('D');
    write_serial('T');
    write_serial(ter);
    write_serial(';');

    write_serial('L');
    write_serial('B');
    i=0;
    while(buff[i] !=NULL)   /* Print The String */
          {
          write_serial(buff[i]);
          ++i;
          }
    write_serial(ter);
    init_plotter();
    return 1;

          return 1;
```

```
        }
/*****************************************************************************/
/********************************** Plot absolute to the point x,y ***********/
/*****************************************************************************/

int plot_lineto(int x,int y)
   {
    pen_down();
    select_velocity(DVELOCITY);
    plot_relative(x,y);
    pen_up();
    select_velocity(DVELOCITY);
    return 1;
   }

/*****************************************************************************/
/******************************** Plot the reacangle *************************/
/*****************************************************************************/

int plot_rectangle(int x1,int y1,int x2,int y2)
    {
    plot_absolute(x1,y1);
    plot_lineto(x2-x1,0);
    plot_lineto(0,y2-y1);
    plot_lineto(x1-x2,0);
    plot_lineto(0,y1-y2);
    return 1;
    }

/*****************************************************************************/
/*************************** Plot relative from present position *************/
/*****************************************************************************/

int plot_relative(int x,int y)
    {
  int i;
  char buff[10];

  write_serial('P');
  write_serial('R');
  itoa(x,buff,10);
  i=0;
  while(buff[i] != NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
  write_serial(',');
  itoa(y,buff,10);
```

```
    i=0;
    while(buff[i] !=NULL)
            {
            write_serial(buff[i]);
            ++i;
            }
    write_serial(';');
     return 1;
     }

/*****************************************************************************/
/************************** Plot text string *********************************/
/*****************************************************************************/

int plot_string(int x,int y,char buff[])
    {
    int i;
    char ter=2;
    pen_up();
    plot_absolute(x,y);
    write_serial('D');
    write_serial('T');
    write_serial(ter);
    write_serial(';');

    write_serial('L');
    write_serial('B');
    i=0;
    while(buff[i] !=NULL)  /* Print The String */
            {
            write_serial(buff[i]);
            ++i;
            }
    write_serial(ter);
    default_plotter();
/*    plot_absolute(10,10);  Move to origin */
    return 1;
    }
```

```
/*****************************************************************************/
/****************************** PLOT  LINE ***********************************/
/*****************************************************************************/

int plot_line(int x1, int y1,int x2,int y2)
   {
   pen_up();
   plot_absolute(x1,y1);
   pen_down();
   select_velocity(DVELOCITY);
   plot_absolute(x2,y2);
   pen_up();
   select_velocity(DVELOCITY);
   return 1;
   }


/*****************************************************************************/
/***************************** SELECT VELOCITY *******************************/
/*****************************************************************************/

int select_velocity(int v)
  {
 int i;
  char buff[10];

  write_serial('V');
  write_serial('S');
  itoa(v,buff,10);
  i=0;
  while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
  write_serial(';');
  return 1;
  }
```

```
/*****************************************************************************/
/****************************** PLOT ABSOLUTE ********************************/
/*****************************************************************************/

int plot_absolute(int x, int y)
  {
  int i;
  char buff[10];

  write_serial('P');
  write_serial('A');
  itoa(x,buff,10);
  i=0;
  while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
  write_serial(',');
  itoa(y,buff,10);
  i=0;
  while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
  write_serial(';');
  return 1;
  }


/*****************************************************************************/
/****************************** PLOT CIRCLE **********************************/
/*****************************************************************************/

int plot_circle(int x,int y, int radious)
  {
  int i;
  char buff[10];
  pen_up();
  plot_absolute(x,y);
  write_serial('C');
  write_serial('I');
  itoa(radious,buff,10);
  i=0;
  while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
```

```
        ++i;
        }
  write_serial(';');
  return 1;
  }

/*****************************************************************************/
/***************************** SELECT PEN ************************************/
/*****************************************************************************/

int select_pen(int no)
  {
  int i;
  char buff[10];

  write_serial('S');
  write_serial('P');
  itoa(no,buff,10);
  i=0;
  while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
  write_serial(';');
  return 1;
  }

/*****************************************************************************/
/***************************** INIT PLOTTER **********************************/
/*****************************************************************************/

int init_plotter()
  {
  write_serial('I');
  write_serial('N');
  write_serial(';');
  return 1;
  }
int default_plotter()
  {
  write_serial('D');
  write_serial('F');
  write_serial(';');
  return 1;
  }
```

```
/*****************************************************************************/
/*************************** PEN UP and DOWN *********************************/
/*****************************************************************************/

int pen_up()
   {
   write_serial('P');
   write_serial('U');
   write_serial(';');
   return 1;
   }

int pen_down()
   {
   write_serial('P');
   write_serial('D');
   write_serial(';');
   return 1;
   }


/*****************************************************************************/
/*************************Initialization and Writing to Serial Port ***********/
/*******************************************************************************/

int init_serial()
        {
        union REGS regs;
        regs.h.ah=0;
        regs.h.al=(128+0+32)+(0+0)+(1)+(1+1);
        regs.x.dx=0;
        int86(0x14,&regs,&regs);
        return 1;
        }

int write_serial(char c)
        {
      union  REGS regs;
        regs.h.ah=0x01;
        regs.h.al=c;
        regs.x.dx=0;
        int86(0x14,&regs,&regs);
        return 1;
        }
```

```
/*****************************************************************************/
/************************** Set Relative Direction ***************************/
/*****************************************************************************/

int relative_direction(int run,int rise)
   {
    int i;
   char buff[10];

   write_serial('D');
   write_serial('R');
   itoa(run,buff,10);
   i=0;
   while(buff[i] !=NULL)
         {
         write_serial(buff[i]);
         ++i;
         }
   write_serial(',');
   itoa(rise,buff,10);
   i=0;
   while(buff[i] !=NULL)
         {
         write_serial(buff[i]);
         ++i;
         }
   write_serial(';');

   return 1;
   }

/*****************************************************************************/
/*************** Draw an edged recangle in absolute co-ordinat ***************/
/*****************************************************************************/

int edge_rect_ab(int x,int y)
   {
   int i;
   char buff[10];

   write_serial('E');
   write_serial('A');
   itoa(x,buff,10);
   i=0;
   while(buff[i] !=NULL)
         {
         write_serial(buff[i]);
         ++i;
```

```
        }
    write_serial(',');
    itoa(y,buff,10);
    i=0;
    while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
    write_serial(';');

    return 1;
    }

/*****************************************************************************/
/*************** Draw an edged recangle in relative co-ordinate*****************/
/*****************************************************************************/

int edge_rect(int x1,int y1,int x2,int y2)
    {
    int i;
    char buff[10];
    plot_absolute(x1,y1);
    write_serial('E');
    write_serial('R');
    itoa((x2-x1),buff,10);
    i=0;
    while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
    write_serial(',');
    itoa((y2-y1),buff,10);
    i=0;
    while(buff[i] !=NULL)
        {
        write_serial(buff[i]);
        ++i;
        }
    write_serial(';');

    return 1;
    }
```