ANALYSIS AND IMPLIMINTATION OF RADIAL BASIS FUNCTION NEURAL NETWORK FOR CONTROLLING NON-LINEAR DYNAMICL SYSTEMS

This Thesis is submitted in Partial fulfilment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the Faculty of Engineering University of
Newcastle Upon Tyne

by

GIUMA MUSBAH FATHALA

5 October 1998

Department of Electrical and Electronic Engineering
University of Newcastle Upon Tyne,
NE1 7RU UK

ACKNOWLEDGEMENT

I am indebted to Dr Mohammad FARSI for his supervision, invaluable encouragement and advice throughout the course of study.

I would like to express my sincere appreciation and a word thanks to my colleagues, friends and all those who helped me in the project and presentation of this thesis. Many thanks to Mr Ian Morns for his help in English. Special thanks to the electrical and electronic department mechanical workshop and electronic workshop technicians, particularly thanks to Mr Graham Ewart for his help in the designing of electronic circuits.

Last but not the least I wish to acknowledge an incalculable debt to my wife and my children. The patience and assistance of my family made the working and the writing of this thesis possible.

Finally my thanks are expanded to the Libyan Ministry of Higher Education for providing assistance enabling me to pursue studies in the United Kingdom.

ABSTRACT

Modelling and control of non-linear systems are not easy, which are now being solved by the application of neural networks. Neural networks have been proved to solve these problems as they are described by adjustable parameters which are readily adaptable on-line. Many types of neural networks have been used and the most common one is the backpropagation algorithm. The algorithm has some disadvantages, such as slow convergence and construction complexity.

An alternative neural networks to overcome the limitations associated with the backpropagation algorithm is the Radial Basis Function Network which has been widely used for solving many complex problems. The Radial Basis Function Network is considered in this theses, along with a new adaptive algorithm which has been developed to overcome the problem of the optimum parameter selection. Use of the new algorithm reduces the trial and error of selecting the minimum required number of centres and guarantees the optimum values of the centres, the widths between the centres and the network weights.

Computer simulation using Simulink/Matlab packages, demonstrated the results of modelling and control of non-linear systems. Moreover, the algorithm is used for selecting the optimum parameters of a non-linear real system 'Brushless DC Motor'. In the laboratory implementation satisfactory results have been achieved, which show that the Radial Basis Function may be used for modelling and on-line control of such real non-linear systems.

Dedication

To my mother 'El-SAKTA' suffering
from my absence and to the memory of my father.
To my wife who has not let her help falter during the studying
agonising years and to my children,
MUSBAH, HAMZA, BADER, REEMA, FARGE AND
TO THE LITTLE GIRL KHOWLA

Thanks for the patience and encouragement

Table of contents

SYMBOLS

U	Input sequence space
Y	Output sequence space
y_d	Desired output
F	Order transformation of the inputs to the GMDH network
X	Input vector
^{T}	Matrix transpose
N	Network training set
h(t)	System impulse response for sample instant t
t	Time of the system input/output
A(z)	System denominator polynomial
B(z)	System nominator polynomial
C(z)	Noise nominator polynomial
z and z^{-1}	Forward and backward shift operator associated with the input/output of the ARMAX model
na	
nb	
nc	
and nk	The orders of the polynomials A, B, and C and the number of delays from system input to system output respectively.
e(t)	White noise sequence; error signal
f(.)	Non-linear system operator relating input data to the output data
f^{-1}(.)	Non-linear system operator relating output data to the input data
r(t)	Reference signal
y_m(t)	Model output
θ	Adjustable parameters
η	Learning factor
W	Network weights vector

ϕ	Non-linear activation function
c	Network centres
σ	Network width
$E_{cluster}$	Local minimum clusters
P	Index of P-nearest neighbours
^	Indicates estimated value
t_s	Sampling time

ABBREVIATIONS

GMDH	Group Method of Data Handing
DF	Describing Function
MSE	Mean Square Error
LS	Least Square
RLS	Recursive Least Square
ARMAX	Auto-Regressive Moving model with Exogenous input
MRAC	Model Reference Adaptive Control
STAC	Self Tuning Adaptive Control
ANN	Artificial Neural Network
RBF	Radial Basis Function
PB	Back Propagation
MLP	Multilayer Perceptron
ART	Adaptive Resonance Theory
SVD	Singular Value Decomposition
SAGO	Summation Square of All the Gaussian Outputs
RBFNm	Radial Basis Function Network model
EAL	Error Accuracy Limit

RBFNc	Radial Basis Function Network controller
DD	Direct Drive motors
DC	Direct Current
ARX	Auto-Regressive model with Exogenous input
BMCC	Brushless dc Motor Control Card
D/A	Digital to Analogue converter
A/D	Analogue to Digital converter
F/V	Frequency to Voltage Converter
TDL	Time Delay Line
PID	Proportional-Integral-Derivative
SSE	Summation Squares Error

CHAPTER 0

INTRODUCTION

CHAPTER 0

INTRODUCTION

A variety of a non-linear complex systems such as robotics and many other electrical systems require modelling and control. An example of the real complex systems dealt with in this thesis is a servo motor. This system is considered to be a black box because of the absence of system information. Therefore, it is desirable to use a method of controller design requiring only partial information about the plant. Artificial Neural Networks (ANN) offer the advantages of performance improvement through network learning. The most widely studied neural network is known as the Multi-Layer Perceptron (MLP) neural network. This type of network consists of fully interconnected layers: one input layer, one or two hidden layers which have an activation function (i.e. sigmoidal) and one output layer. Use of this type of network in system identification and control has been successful in different areas, but still suffers from limitations such as the complexity of construction and slow convergence.

An alternative neural network the Radial Basis Function (RBF) is used in this thesis. This network is unlike the MLP, in its construction. RBF network consists of one input layer, one output layer and only one array of hidden nodes called centres. The RBF network is a simple architecture and moreover, the training of this network is faster than that the MLP network. One reason the RBF is very fast, is that learning is divided into two stages. Learning in the hidden layer for selecting centres and widths and the learning in the output layer for selecting the weights.

One problem still exists in using the RBF network, that is the selection of the optimum centres and widths. This problem has been overcome by incorporating various algorithms i.e. K-means clustering algorithm, P-nearest neighbour, Gaussian activation function and least mean square algorithms in one adaptive algorithm. The new algorithm, can adaptively select the centres and the widths of any system given a knowledge of the system input-output and delays.

Two RBF networks were designed; a static network for modelling and a dynamic network for control of non-linear plants. In this thesis, the neural network control approach includes: the general learning method and the adaptive control method. The closed-loop system was used for modelling and control of simulation examples using the RBF network, where successful results have been obtained.

The main objective of designing the networks, is to model and control the experimental system speed. This system is a three phase high speed Brushless DC Motor. The motor speed is proportional to input voltage generated in a PC and controlled by a control card designed in our laboratory. The motor output has been converted by a simple frequency to voltage converter (F/V) circuit also designed in house. The whole experiment was set-up and interfaced with a PC by hardware; e.g. a PCL-818l data acquisition card and software programs. An open-loop system has been considered for identifying the real-system (plant) using an autoregressive exogenous (ARX) model. A closed-loop system has been considered with the dynamic network used for controlling the *'motor model'* on-line. The simulated control of the motor speed was tested where results were obtained by applying changes to input signal. Finally, the *'motor'* itself is controlled on-line by using of a square input signal and changeable parameters. Then the same input signal was used together with the optimum parameters and the obtained results were compared.

This thesis is outlined as follows. In chapter 1, a brief discussion about the principle of the system identification is given. This is followed by several methods which have been used for a non-linear system identification. The non-linear identification methods provided in this chapter are: the group method of data handling, the function series and the parameter estimation methods. Furthermore, the basic control system and four control methods are given. These are: adaptive control, gain scheduling, model reference adaptive control and the self tuning adaptive control methods. These are discussed briefly and the block diagram for each method is illustrated. Chapter 2 provides background for the neural networks, starting with the basic idea of artificial neural networks. Although, some types of neural network are introduced, the *Radial Basis Function* network is discussed in detail, since the main work in this thesis depends on this particular type of

neural network. In addition, clustering algorithms, distance measurements, activation functions and weight adapting algorithms are presented. In chapter 3, a new algorithm for selecting the centres and widths adaptively is explained and proved by a simple hypothetical example.

Chapter 4, provides an introduction to the modelling of a non-linear dynamic system using artificial neural network with the representation of the general model being provided. The procedure and the block diagram structure for training feedforward neural network models with the equation of a non-linear system are discussed. The analysing of a mathematical equation, which describes the RBF neural based identification model output, the hidden or Gaussian output, the network inputs-outputs and the predictive error equations are presented. The parallel-series method for modelling a non-linear systems is used and two examples in which the input signals and the network parameters respectively are varied are tested and the simulation results are presented. In chapter 5, the work outlined is mainly concerned with configurations suitable for the control of dynamic non-linear systems using the radial basis function. The concept of the controlling procedure for SISO dynamic systems is analysed and the block diagram structure for training the network controller is given. Moreover, two control methods are discussed and simulation results are presented. In chapter 6, the real system is modelled and controlled. The system considered was a Brushless DC Motor control system which is a real example of a system containing non-linearity. First basic motor construction and its general dynamics together with the transfer function are given. Thus, the necessary motor interface hardware cards and software programs are mentioned. In addition, the whole experiment is set-up and the ARX model is used for identifying the real system. The schematic model is illustrated and the results of modelling the system using the RBF network are shown. Then, by using the closed-loop system, the motor speed is controlled on-line. The system diagram is shown and the monitored simulation and real results are explained. Finally, conclusions and recommendations for further work are made in chapter 7.

CHAPTER 1

SYSTEM IDENTIFICATION AND CONTROL

CHAPTER 1

SYSTEM IDENTIFICATION AND CONTROL

1.0 INTRODUCTION

In order to design controllers for a dynamic system it is useful to have a model that will describe the system's behaviour adequately. Because of the complexity of the system and unknown parameters, the designer turns to experimental data and the measured response. The process of constructing models and/or estimating the unknown plant parameters from experimental data is called system identification. System identification can be described as a method for finding the relationship between various signals of a system under observation. This relationship, often called a model of a system, is usually a system of difference equations in discrete-time and differential equations in continuous-time. The identification method may be based on linear or non-linear systems, and its process may be carried out in either an on-line or off-line mode, depending on the application context. In practice most systems are non-linear to some extent, therefore a non-linear system identification method may be needed. Due to the difficulties of analysing non-linear systems, only a few methods have been developed. Each method depends on the type of non-linearity of the identified system. These traditional methods have been based upon Group Method of Data Handling (GMDH), Parameter Estimation and Functional Series Methods [55,56,58]. Each method has specific problems and limitations which are discussed briefly in this chapter, particularly in sections 1.1.1-1.1.3. Once the system is identified, the next step is to design a controller for the same system. The designs have been produced by means of using conventional or classical control methods. Conventional control theory is widely applied in linear systems having constant parameters. This is often a good approximation for systems that are controlled at fixed operating points. The controllers will not always be satisfactory when the operating condition changes. Therefore, the iterative *adaptive control system* has been used in these

cases. The adaptive control system is a type that is able to adapt itself to changing parameters at various operating points of the system. The extensive research on adaptive control was started in the early 1950s. To the present date rapid progress has been achieved and many applications are reported [3,90]. After a brief discussion of basic control system, in section 1.2, the general adaptive control structure including with three adaptive systems are described in sections 1.2.1-1.2.3.

1.1 Non-linear System Identification

System identification can be described as a method of finding the relation between the system's input and output signals. System identification may be carried out to gain a better understanding of the properties of a system and to design a good controller for the system based on the identified model. In order to design a control system it is necessary to have a mathematical model that adequately describes the system's behaviour. There are many identification techniques that can be used. These could be linear or non-linear. Before discussing these techniques, the difference between linear and non-linear systems will be briefly discussed. In control literature if a system *S*, depicted in Figure (1.1), has an input *(u)* and an output *(y)*, it is termed linear if the system output depends linearly on the given input, i.e. the principle of superposition holds.

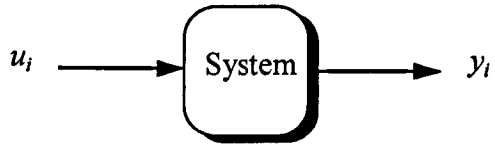


Figure 1.1, Input output model

$$y_i = S(u_1 + u_2 + \ldots u_n) = (y_1, y_2, \ldots, y_n) \qquad i = 1,2,3,\ldots, n \qquad (1.1)$$

Any system that does not satisfy this requirement is defined as non-linear. Linear system identification has been well established and it is found in many applications, but non-linear system identification has received little attention and only a few methods for identifying non-linear systems have been developed. This may be attributed to the difficulty of analysing such systems. In general for both cases, linear or non-linear system, identification can be carried out in one of two ways. The first is the identification when model structure is assumed and an analytic description can be constructed for the system. The second is the identification with an unknown model structure. This latter type of identification is used only if not enough information about the model structure is known to allow analytic equations to be written down. Various techniques have been used for identification of non-linear systems and some of these are discussed in the following sections.

Before discussing identification techniques, a brief description of classical non-linear controllers is given. Assume the system has been identified. The next step is to control the same system. Many control systems contain non-linear elements. Whilst designing a non-linear control system, an analysis phase includes examination of a fixed structure to determine such properties as signal size, stability and dynamic response is needed. Since general non-linear systems have proved so difficult to study, many types of approximate methods have been proposed i.e., the linearization method.

Another approximate function technique was proposed by R. J. Kochenburger in 1950, [68] called Describing Functions (DF). This technique has been widely used and many non-linear problems have been solved. The DF analysis concerns basically with the frequency response of the system and treated only in terms of sinusoidal input signals. The basic idea is that a sinusoidal input signal *(u)* to a non-linear device *f(u)* produces an output signal *(y)* that has frequency as the input, with different shape and possibly shifted in phase. Describing Function analysis assumes that only the functional component of the output is important. Thus, the output can then be expressed by a Fourier series as the sum of an infinite number of frequency and phase shifts. It then assumed that the fundamental

component of the output adequately describes the system response and the higher harmonics are damped out of the system. The describing function is thus the ratio of the fundamental component of the output to the input [30,68,83].

1.1.1 The Group Method of Data Handling (GMDH)

The group method of data handling technique is a multilayer self organising algorithm based on a non-linear mathematical model of data. This method was first introduced by Ivakhnenko in the 1960's [26]. Ivakhnenko developed the method using the principles of heuristic self organisation to solve complex problems with large dimensionality and short data sequences. The method has received much attention by many other researchers and has been used for solving many problems, such as identification of static and dynamic non-linear models, pattern recognition, optimal control etc. The schematic of the method is illustrated in Figure (1.2). To make the method clear, suppose $x_i = [x_1, x_2, x_3, \ldots, x_m]$ is the input and the output $y_i = F[x_i]$ is a non-linear function of x. The procedure for the method is summarised in the following steps :

1. The original data is divided into training set and testing set.
2. Quadratic polynomials are formed for all possible combinations of x_i variables, taking pairs each time.
3. For each polynomial a system of normal Gaussian equations is constructed using all data points in the training set.
4. The models are used to predict the system response in the training set data region. The predictions are passed through a mean squares error (MSE) selection criteria, such as $MSE = \frac{1}{n}\sum_{i}^{n}(y(t) - \hat{y}(t))^2$, where $\hat{y}(t)$ is the predicted value, and n is the number of data points in the testing set.

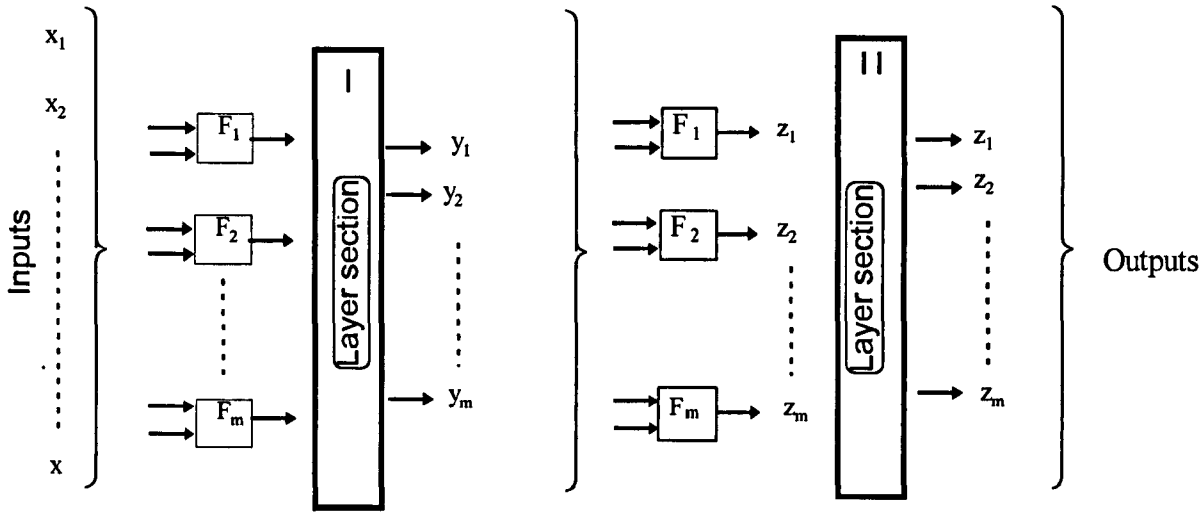


Figure 1.2, The schematic for the Grouping Method of Data Handling (GMDH)

5. The outputs $y,...,y_m$ are ordered with respect to the smallest MSE. The model is allowed to pass to the next level of GMDH if its MSE is less than a specified threshold.
6. At the next level the independent variables for the new training and testing sets are found by mapping the original training and testing sets of data through the single layer which has been formed.
7. According to step 2 new polynomials are formed, and for each layer steps 2-6 are repeated until the smallest MSE is reached.

For obtaining satisfactory results the GMDH, or Perceptron, must have four layers or more. Too many layers may give poor results. The Perceptron results are indicated by the mean squares error method. To avoid unsatisfactory results, the best solution must be chosen based on data from layers of the Perceptron not on the results of the last layer. As an example, assume that the input vectors in the training set are N, each composed of p property values, $X_n = (X_{n1},...,X_{np})$, $n = 1,2,3,...,N$ and the desired outputs are $y_d(n)$.

The function implemented by an element in one of the layers is

$$y = F_2(X) = f_0 + f_1 x_1 + f_2 x_2 + f_3 x_1^2 + f_4 x_2^2 + f_5 x_1 x_2 \tag{1.2}$$

where the subscript in F_2 denotes a second order transformation of the inputs.

Considering one element in the first layer the coefficients of this element can be determined in such that the mean square error between the outputs y_n and the desired output $y_d(n)$ is minimised. Thus, the coefficients are obtained from the Gaussian equations as

$$\left.\begin{aligned} y_d(1) &= f_0 + f_1 x_{1i} + f_2 x_{1j} + f_3 x_{1i}^2 + f_4 x_{1j}^2 + f_5 x_{1i} x_{1j} \\ &\vdots \\ y_d(N) &= f_0 + f_1 x_{Ni} + f_2 x_{Nj} + f_3 x_{Ni}^2 + f_4 x_{Nj}^2 + f_5 x_{Ni} x_{Nj} \end{aligned}\right\} i = j = 1,2,3,\ldots,N \tag{1.3}$$

rewrite equation (1.3) in the matrix form as

$$Y_d = XF \tag{1.4}$$

where matrix Y_d, X and F are of order $N\times1$, $N\times6$ and 6×1, respectively. Multiply both sides of equation (1.4), by the transpose of X, gives

$$X^T Y_d = (X^T X)F \tag{1.5}$$

where matrix $X^T X$ is 6×6 and the solution is found by inverting the matrix in equation (1.5)

$$F = (X^T X)^{-1} X^T Y_d \tag{1.6}$$

Vector F contains the set of the coefficients which enables this element to approximate the actual outputs value with minimum squares error. The procedure is repeated for each element in the first layer with the components in matrix X changing each time dependent on the identity of the pairs of the input to the particular element. The same procedure and technique are used to find the six coefficients of each element in the other layers. It is recommended that, as mentioned earlier in this section the, experimental data is divided into a training and testing set. If the training data which were used to estimate the coefficients are used for the network testing, unsuccessful results are expected, since small changes in the training data will lead to large changes in the coefficient values.

The advantage of GMDH, is its ability to construct differential equations for the system without *a priori* information on the relationship between input and output variables. On the other hand using this algorithm requires very heavy computing power [10,46,58].

1.1.2 Function Series Methods

The functional series method of Volterra and Wiener is based on the representation of a system by the Volterra series [1,10]. It is well known that, for a linear time invariant system, the output response $y(t)$, to an input $x(t)$, may be computed from a knowledge of the system-impulse response $h(t)$, by using the convolution integral:

$$y(t) = \int_{-\infty}^{\infty} h(t)x(t-\tau)d_\tau \tag{1.7}$$

where $u(t)$ and $y(t)$ are the system input and output respectively. A study of non-linear function (1.7) was made by Volterra [80] in 1887. He studied analytical functions and introduced the following representation

$$y_{(t)} = \int_{-\infty}^{\infty} g_1(\tau_1)x(t-\tau_1)d\tau_1 + \int_{-\infty}^{\infty}\int g_2(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)d\tau_1 d\tau_2 +$$
$$\int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty} g_1(\tau_1,\tau_2,\ldots,\tau_n)x(t-\tau_1)x(t-\tau_2)\ldots \qquad (1.8)$$
$$\ldots x(t-\tau_n)d\tau_1 d\tau_2 \ldots d\tau_n$$

where, $x(t)$ and $y(t)$ are the input and the output of the system at time t, and the function $g_n(\tau_1,\tau_2,\ldots,\tau_n)$ is termed the Volterra kernel of order n. The kernels in (1.8) are bounded and continuous in each τ_n, and symmetric functions of their arguments [10].

Unfortunately, some difficulties arise in the use of Volterra series in system modelling (e.g., the problem of practical measurements of the Volterra kernels). These problems have not prevented research and in 1942, Weiner overcame the Volterra limitations. Weiner was one of the first researchers to develop and apply Volterra series to the identification of non-linear systems. Thus, the Weiner model structure is shown in Figure (1.3), with the dynamic linear system first, followed by a static non-linear element [7,10,95].

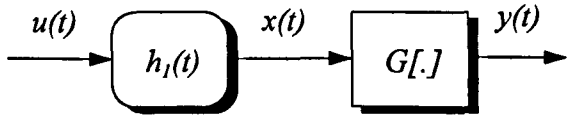


Figure 1.3, The Weiner model

$u(t)$ and $y(t)$ are the input and the output respectively.

Although, the functional series of non-linear systems are now well established, very few attempts have been made at using such series for practical identification of real systems. This might be attributed to the difficulties associated with the system kernels and the excessive computational requirements necessary to characterise systems. Moreover, when

using Volterra kernels, only two kernels are considered for identification of non-linear systems, for any system involving higher than second order kernels, this offers considerable difficulties. Furthermore, Weiner's formulation is impractical and difficult to use because of the high number of coefficients required [10].

Other researchers have tried to develop a solution to overcome the problems of non-linear system identification. In 1966 Narendra and Gallman [65] produced a technique known as the Hammerstein model shown in Figure (1.4). This model consists of a non-linear element followed by a linear element. These authors have examined this model for identification of various non-linear systems and conclude that the experimental results of computer simulations indicate the method could be quite effective for many non-linear systems for which the Weiner model could be a poor choice.

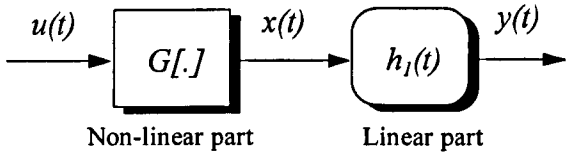


Figure 1.4, Hammerstein model

Thus, if the Hammerstein model is considered to be the system identification method then, from the separability theory [6], each block can be identified on its own. Based on this theory, if the linear dynamic part of this model is known or can be estimated recursively, using linear control theory techniques, then the non-linear part can be identified by a non-linear identification method such as correlation functions or Newton Raphson. The results obtained for the Hammerstein model are readily valid for the Wiener model [1,6,8]. The comparison between these two methods and the summary of the identification of each techniques have been given in [45].

Consequently, various researchers have turned their attention to a restricted class of non-linear systems. This system is known as the block oriented method, cascaded system or

general model. The model as shown in Figure (1.5), is composed of a linear system followed by a non-linear system followed by another linear system.

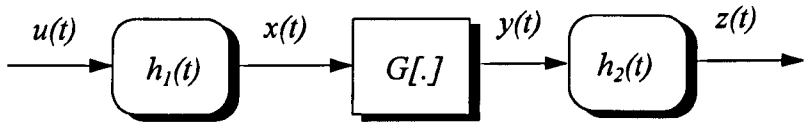


Figure 1.5, The general non-linear model

where $h_1(t)$ and $h_2(t)$ are the linear elements, $y(t)$ is the non-linear element's output, $u(t)$ is the system input and $z(t)$ is the system output. This method is typically an *off-line* technique. It has been studied by many researchers and they have found that it requires extensive experimental action to identify the system accurately [7,8,10]. In conclusion, the identification of a non-linear systems is very difficult and none of the stated techniques can be recommended as providing an acceptable solution and each must be judged according to the problem under investigation and their merits.

1.1.3 Parameter Estimation Method

The parameter estimation methods for identification of non-linear systems has been considered by many researchers. This method has been very successful in many applications, but is limited in how good an approximation it may give [9]. One of the most widely used structures is a linear difference equation, assumed to be in the form of an autoregressive moving average exogenous (ARMAX) model, given in the following equation

$$A(z)y(t) = B(z)\,u(t-nk) + C(z)\,e(t) \tag{1.9}$$

in which the polynomials A, B and C are defined as:

$$\left.\begin{aligned} A(z) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{na} z^{-na} \\ B(z) &= b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{nb} z^{-nb} \\ C(z) &= 1 + c_1 z^{-1} + c_2 z^{-2} + \ldots + c_{nc} z^{-nc} \end{aligned}\right\} \quad (1.10)$$

where z^{-1} is the backward shift operator (delay operator), *nk* is the number of delays from system input to system output and *na, nb* and *nc* are the orders of the polynomials *A,B* and *C*. Also *y(t)* and *u(t)* are the system output and input sequences respectively and *e(t)* is a disturbance affecting the system.

Equation (1.9) can be rearranged in a linear difference equation format as:

$$y(t) = -\sum_{i=1}^{na} a_i y(t-i) + \sum_{i=0}^{nb} b_i u(t-nk-i) + (1 + \sum_{i=1}^{nc} c_i e(t-i)) \quad (1.11)$$

where *nk* is the system time delay (nk ≥ 1).

If the structural of model (1.11) is known, then the model is simply a linear combination of unknown parameters. Thus, the unknown parameters are estimated directly by using the available input-output data with one of several algorithms (e.g., Least Square, Recursive Least Square, etc.). In practice most systems in industry are non-linear to some extent and in many applications, non-linear models are required to provide acceptable representations. Moreover, in some models a little *a priori* information is available and the process is treated as a black-box. In this case, the usual approach is to expand the input-output data using a suitable model representation, which is usually selected to be non-linear in the input and output variables.

When the system is non-linear however, the traditional system descriptions are based on group method of data handling and functional series such as described in sections (1.1.1 and 1.1.2). Whilst, these provide an adequate representation for a wide class of non-linear

systems, they have some difficulties and several parameters are required to characterise the non-linear systems making the identification of such systems limited. Thus, Billings and his group have realised that the wide application of linear difference equations make it natural to search for non-linear difference equation models. Therefore, they have proved that the linear difference equation (1.11), which relates sampled input signals to sampled outputs can be generalised for the non-linear time invariant process.

The linear different equation (1.11), can be realised as

$$y_i(t+p) = \begin{bmatrix} a_{i1}\, a_{i2} \cdots a_{in} \end{bmatrix} \begin{bmatrix} y_1(t+n_1-1) \\ \vdots \\ y_1(t) \\ \vdots \\ y_m(t+n_m-1) \\ \vdots \\ y_m(t) \end{bmatrix} + \begin{bmatrix} b_{i1}\, b_{i2} \cdots b_{ik} \end{bmatrix} \begin{bmatrix} u_1(t+p) \\ \vdots \\ u_1(t) \\ \vdots \\ u_r(t+p) \\ \vdots \\ u_r(t) \end{bmatrix} \tag{1.12}$$

where $i = 1,2, \ldots, m$, r and m are the dimensions of the input-output vectors u and y respectively, and $p = max(n_1, n_2, \ldots, n_m)$. The integers indices $n_1, n_2, \ldots, n_m$ are the observability indices of the system and the summation of these integers $(n = n_1 + n_2 + \ldots + n_m)$ is the model order. Every index n_i corresponds to the specific output y_i. The model (1.12) can be regarded as m interconnected single output models.

The multi-structural input output linear time invariant relationship of equation (1.12) was generalised to the non-linear equation [55,56] as

$$y_i(t+p) = f_i\{y_1(t+n_1-1), \cdots, y_1(t), \cdots, y_m(t+n_m-1), \cdots, y_m(t), u_1(t+p), \cdots, u_1(t), \cdots, u_r(t+p), \cdots, u_r(t)\} \tag{1.13}$$

where $f(.)$ is a non-linear function.

For the case of single input single output (SISO) systems, equation (1.13) can be realised into the non-linear stochastic form as [55,56,86]:

$$y(t+1) = f[y(t),. \quad . \quad .,y(t-n+1),u(t+1),. \quad . \quad .,u(t-n+1)+ e(t),. \quad . \quad .,e(t-n+1)]+e(t+1) \tag{1.14}$$

This model is called the NARMAX model (non-linear autoregressive moving average model with exogenous input) for the standard ARMAX model *(equation (1.9))*. A special case of the general NARMAX model (1.14) is the non-linear autoregressive with exogenous (NARX) model

$$y(t+1) = f[y(t),. \quad . \quad .,y(t-n+1),u(t+1),. \quad . \quad .,u(t-n+1)+e(t) \tag{1.15}$$

The non-linear functional form of $f(.)$ for a real world system can be very complicated and is rarely known *a priori*. Therefore a model must be constructed based on some known simpler functions [15,17,18,55,56].

1.2 Control Systems

In the recent years control systems have assumed an increasingly important role in the development and advancement of modern civilisation and technology. Control systems are found in abundance in the domestic domain and in the industries, such as in air-conditioning, transportation systems, robotics and many other systems .

The basis system is shown in Figure (1.6), where $u(t)$ and $y(t)$ are the system input and output respectively, and the intermediate block is the system components.

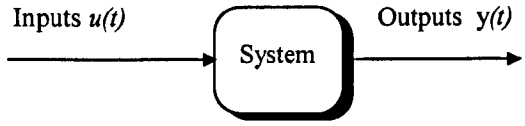


Figure 1.6, The basic system

Regardless of what type of control systems are used the systems can be controlled by either open-loop or close-loop methods. However, this depends on the nature of the system to be controlled.

Open loop systems: an example of open loop systems is an electrical washing machine. The controlling of such systems should be open loop, this is because the amount of machine wash time, soap powder quantity etc. are determined by the user. The reason being that the machine cannot continuously detect and check the cleanness of clothes being washed i.e. the machine cannot make decision whether to stop or start washing. As shown in Figure (1.7), the open loop system consists of a controller followed by the controlled process system.

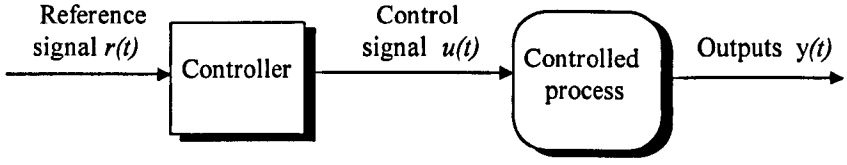


Figure 1.7, Open loop control system block diagram

In the above diagram an input signal $r(t)$ is applied to the controller, the controller signal $u(t)$, controls the process to produce the desired output signal $y(t)$. This type of controller is very simple and economical but would not satisfactorily fulfil the desired performance requirements for many cases. In these cases the closed loop system can be used.

Closed loop systems: the closed loop system block diagram is depicted in Figure (1.8). This type of control system has the same structure as the previous one, except for one difference, the output signal is returned to be compared with the reference signal.

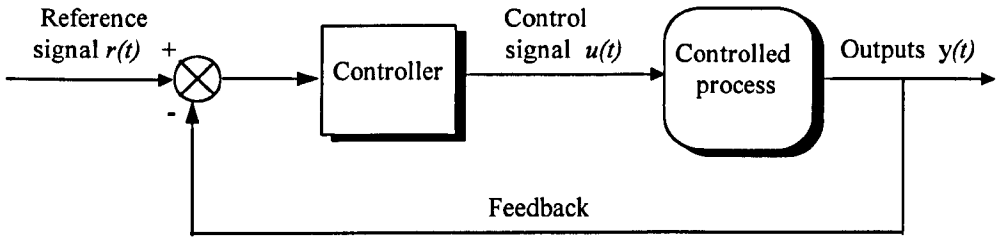


Figure 1.8, Feedback control system

The control signal *u(t),* is a function of the difference of the reference and output signals, must be sent through the controlled process to correct the error. This procedure is called feedback control. The classical linear controllers have been found not always to give satisfactory results, since the parameters of the process may change for some reason, e.g. ageing, operating point changes, mild non-linearities, etc . Hence there seemed to be a need for a more sophisticated controller which could automatically adapt itself to changing characteristics of the controlled process. Therefore, researchers have focused their attention towards adaptive control methods.

1.2.1 Adaptive Control

Adaptive control is used to alleviate the problem of varying plant dynamics. The main idea of using an adaptive control strategy is to adjust the controller parameters automatically, based on the measured input-output of the plant. So adaptive control is the problem of controlling the output of the process (plant) with a known structure but unknown parameters. The general structure of an adaptive controller consists of three

elements as described in Figure (1.9). The elements are the process, adjustable mechanism and the controller.

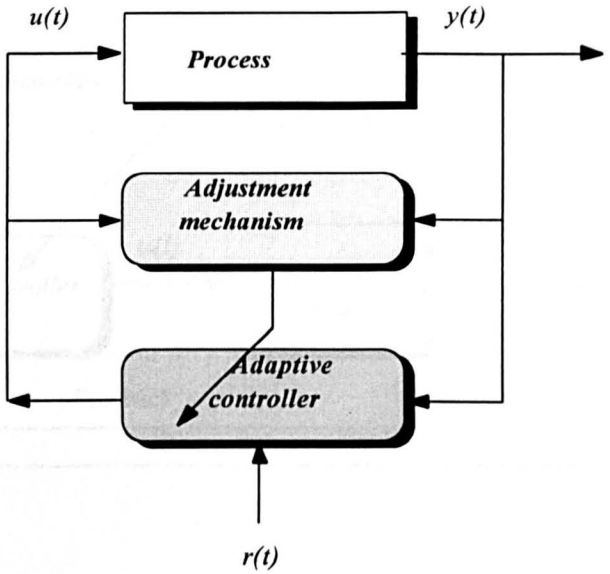


Figure 1.9, General adaptive control structure

Where $r(t)$ and $y(t)$ are the desired and actual outputs respectively and $u(t)$ is the controlled signal. In the following sections, some adaptive systems are presented to provide the reader with some examples. The review of the implementation of adaptive controllers and more information about this task may be found in [3,40].

1.2.2 Gain Scheduling Method

An adaptive controller is a controller that can modify its behaviour in response to changes in the dynamics of the system. In many situations, however, the dynamics of a system change with its operating conditions. In this way, the controller parameters can be adapted directly as a function of the system operating conditions. This idea works in

feedforward mode and is referred to as Gain Scheduling. It is depicted in Figure (1.10), where *r(t)*, *y(t)* and *u(t)* are the reference input, output and control signals respectively.

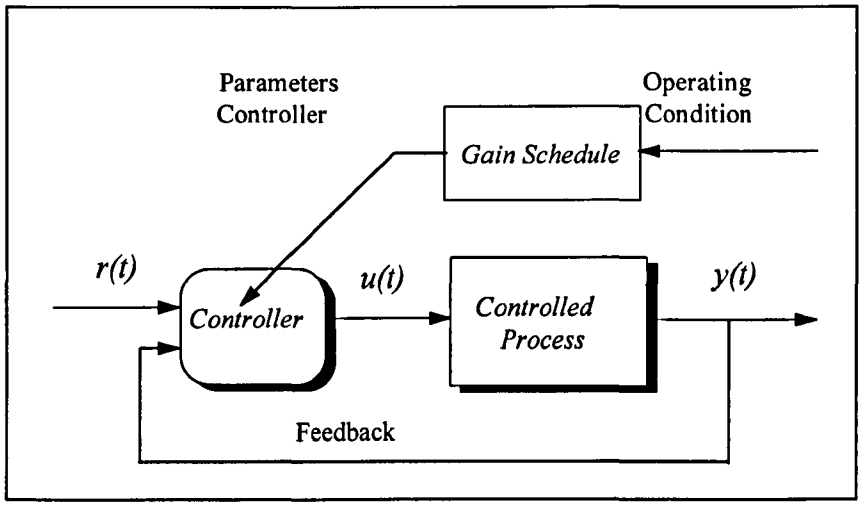


Figure 1.10, The gain scheduling block diagram

The gain schedules are designed off-line and adjusted in an open-loop mode i.e. there is no feedback to compensate for an incorrect schedules. The main problem in the design of systems with gain scheduling is to find suitable scheduling variables. This is normally done based on knowledge of the physics of a system. This method was widely used in process industries where the process dynamics are non-linear functions of one or more parameters. The method has the advantages that the controller parameters can be changed very quickly in response to process changes. These parameters must be determined for many operating conditions and the performance ideally be checked by simulations. On the other hand, in this approach, the controller parameters are changed in an open-loop process without feedback from the performance of the closed-loop system. This makes the method impossible to use if the dynamics of the process or the disturbance are not known accurately. Also this type of adaptive control cannot be generalised, and so has been used only for special cases, such as in auto-pilots for high-performance aircraft [30,40,90].

1.2.3 Model Reference Adaptive Control (MRAC)

Model reference adaptive control is one of the most important and successful forms of adaptive control strategy. The original MRAC was first proposed by Whitaker in the late 1950's. As shown in Figure (1.11), the reference model is set in parallel with the system to be controlled (controlled process or plant). In this model, the measured input-output data are used to monitor the system performance. These are then combined with the reference model output $y_m(t)$ according to an adaptation rule and the result is used to adjust the controller.

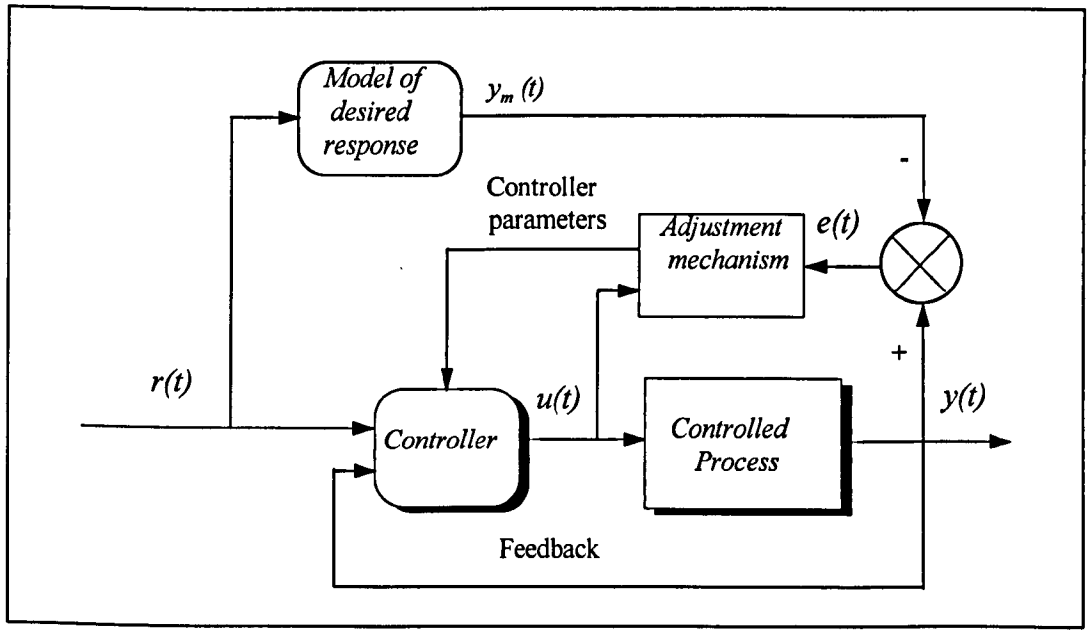


Figure 1.11, Model reference adaptive control block diagram

The controller consists of two loops: the inner loop, an ordinary feedback loop composed of the process and the controller element and the outer loop, which is used for adjusting

the controller parameters. These parameters originally are adjusted by the Massachusetts Institute of Technology (MIT) rule,

$$\frac{d\theta}{dt} = -\eta e \frac{de}{d\theta} \tag{1.16}$$

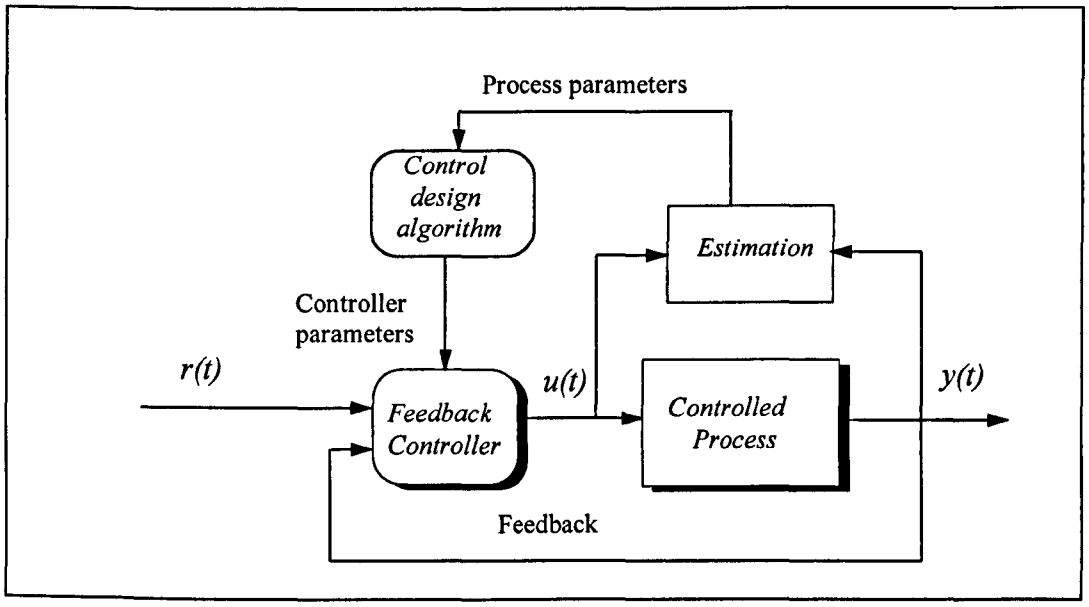
where

$\frac{de}{d\theta}$	are the sensitivity derivatives of the error
η	is the adaptation rate
e	is the error and
θ	is the adjustable parameter.

The MIT rule is the adjustment mechanism which keeps the system stable while driving the error $e(t)$, between the controlled process output $y(t)$ and the model output $y_m(t)$, to small values. This procedure is called adjustment of controller parameters. The MRAC has been studied and applied in many applications [3,48,90].

1.2.4 Self-Tuning Adaptive Control (STAC)

Self-tuning adaptive controller, illustrated in Figure (1.12), is based on the idea of estimating some parameters of the process. The STAC is composed of three main elements. The first is the standard feedback controller in the form of a difference equation, which acts upon a set of values such as the measured output and reference signal. This controller then produces a new control signal $u(t)$ to be the input signal to the process. In this approach the feedback controller element and the process are called the inner loop. The second element is the parameter estimator, which computes the process dynamics by using knowledge of the process input-output.



Figure, 1.12, The self-tuning adaptive control block diagram

The parameters are estimated on-line using one of the recursive algorithms such as least-squares and extended and generalised least squares. The third element is the control design algorithm, which receives the estimated parameters and then provides a new set of coefficients (controller parameters) for the feedback controller. The control design element represents an on-line solution to the design problem for a system with known parameters. Some design methods that can be used are pole placement or minimum variance. The control design algorithm and the recursive parameter estimator are the outer loop, which update the controller parameters at each sampling period.

In conclusion, the STAC is designed to obtain an automatic adjustment mechanism. It must identify the system (controlled process) using measured input-output data to form an appropriate controller. The adaptive control can be identified as a control technique in which controller parameters are continuously and automatically adjusted, in response to measured variables, in order to approach optimum performance [3,90].

As discussed above, it is clear that the STAC is a form of MARC. The distinction between these two methods has mainly been based on the different design approaches. From Figures (1.11) and (1.12) we can see that both control schemes are similar to each other. The difference between the methods is only the updating of the control parameters. This difference however, is not fundamental, because STAC may be modified so that the control parameters are updated directly, the same as the MARC [3,48].

Conventional and adaptive control methods have been widely used and satisfactory results have been obtained when they are applied in linear systems. However, very little attention has been given to non-linear control systems. This is because of the complexity of such systems. One approach called bilinear, has been widely introduced by many researchers [21,37]. Models having a bilinear structure have been shown to be applicable to many non-linear systems. The bilinear approach was used for controlling of a non-linear systems and an acceptable simulation and experimental results were obtained. In conclusion, if the system is a non-linear, then all these methods will suffer greatly and hence more recently *Artificial Neural Networks (ANN)*, have been studied extensively and are discussed briefly in chapter two. A special type of neural network, known as the *Radial Basis Function (RBF)*, is also discussed in detail in this chapter. The RBF method is developed and applied to model and control of non-linear simulated examples, as presented in chapters three, four and five. Also the RBF is used for modelling and control of a real laboratory servo system. The implementation and the obtained results are discussed in chapter six.

CHAPTER 2

ARTIFICIAL NEURAL NETWORKS (ANNs)

CHAPTER 2

ARTIFICIAL NEURAL NETWORKS (ANNs)

2.0 INTRODUCTION

The concept of a neural network was originally conceived as an attempt to model the biophysiology of the brain. The early work in neural networks was carried out by biophysicists and scientific psychologist groups. At the same time, research engineers were concerned with how to use *Artificial neural networks (ANNs)* to form controllers from neurons with interesting and powerful computational capabilities.

ANNs offer a potential solution for problems which require complex data analysis and promise to form the future basis of an improved alternative to current engineering practice. Therefore, this area has received a considerable amount of attention from many researchers.

The first idea of neural networks was conceived by McCulloch and Pitts (1940s) [78] as a means of mimicking human brain activity. In 1943 they published the first systematic study of artificial neural networks. McCulloch and Pitts carried on working in this field and much of their work involved the simple neuron network *(have been called perceptrons).* These neurons are still the major building block of virtually every neural network being developed [42].

In the 1960s Minsky and Papert proved that there are several restrictions in the tasks the simple single layer perceptron can perform; e.g. it can not implement the simple exclusive-OR logic problem. Therefore, neural network research was extinguished until the early 1980s, when John Hopfield studied an *autoassocitive* network that has some similarities with the perceptron [31,82,88].

A learning algorithm called the generalised *delta rule* or *backpropagation rule* was developed and reported by Rumelhart, Hinton and Williams in 1986. In the same year, *parallel distributed processing* by Rumelhart and McClelland were published in two volumes. After their publications, the field exploded with research publications [4,42]. Many researchers found that neural networks have many applications in various fields of study including signal processing, modelling and control of linear and non-linear systems. However, one of the great potentials areas for the application of neural networks is in control.

Neural networks have been developed in different ways, where various algorithms and methods have been applied. In this chapter a brief discussion of the simple *perceptron* and the well known *Back-Propagation (BP)* rule, are given. Later, the *Radial Basis Function (RBF)* is discussed in detail; this ANN architecture is used in our research.

2.1 Perceptron

The simple perceptron shown in Figure (2.1), is a single processing unit with an input vector $X=(x_0,x_1,x_2,...,xn)$. This vector has n elements and so is called an n-dimensional vector. Each element has its own weights usually represented by the n-dimensional weight vector W, e.g. $W=(w_0,w_1,w_2,...,w_n)$.

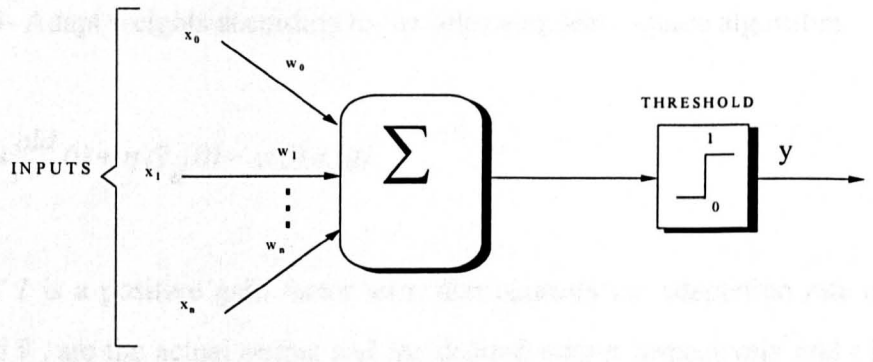


Figure 2.1, Perceptron diagram

The output y is a weighted sum of the input X. Mathematically, this is described by the following equation:

$$y = \sum_{i=0}^{n} w_i x_i \tag{2.1}$$

Equation (2.1), is thresholded to give a binary output, that is

$$y = \begin{cases} 0 & \text{if } \sum_{i=0}^{n} w_i x_i < 0 \\ 1 & \text{if } \sum_{i=0}^{n} w_i x_i \geq 0 \end{cases} \tag{2.2}$$

The output is at either the state 1 or 0; the neuron is either *on* or *off*. The training of the perceptron can be done by changing the weight vector. The perceptron can also be trained using a set of input-output pairs called training data, and the weights are changed by using a learning law. This learning law can be summarised as following

1- Randomise all network weights.

2- Present input training vector X and desired output y_d.

3- Calculate the actual output using equation (2.1).

4- Adapt weights according to the following least square algorithm.

$$w_i^{new}(t+1) = w_i^{old}(t) + \eta\,(\bar{y}_d(t) - y(t))\,x_i(t) \tag{2.3}$$

where $0 < \eta \leq 1$ is a positive gain factor term that controls the adaptation rate of the algorithm, y and $\bar{y}_d$ are the actual output and the desired output respectively and t is the current time.

5- Steps 2- 5 should be repeated until the network converges.

This algorithm adjusts the weights to reduce the error at each iteration until ideally $(\bar{y}_d(t) - y(t)) = 0$ which means no modification to the weight would be necessary. The perceptron has a major limitation in that as mentioned before it can not solve the simple Exclusive-OR example. This problem has been overcome by the work of Widrow and Hoff [4,41,88] in which they proposed a learning rule known as the Widrow-Hoff delta rule (Least Mean Squares). Using this rule the square of the difference between the weighted sum and the required output (which they called the error, E) is calculated. That is

$$E(t) = \sum_{i=1}^{N} (y_d(t) - y(t))^2 \tag{2.4}$$

$$\Delta = (y_d(t) - y(t)) \tag{2.5}$$

where N is the length of the output data. This error is minimised with respect to the weights .

Substitute (2.5) into (2.3) yields

$$w_i^{new}(t+1) = w_i^{old}(t) + \eta \Delta x_i(t) \tag{2.6}$$

It is worth noting that the equation (2.6) is written exactly as equation (2.3), however, in equation (2.3) the output variable has only two states (0,1). The Δ in equation (2.6) is varied depending on the difference between the actual output and the desired output.

2.2 Multi-Layer Perceptron

If we recall equation (2.2) in the previous section, the two state neuron being either *on* or *off*, gives no indication of the scale by which we need to adjust the weights. In other words the *hard-limiting* threshold function removes the information that is needed if the network is to learn successfully. This difficulty is known as the *credit assignment* problem since it means that the network is unable to determine which of the input weights should be increased or decreased and which ones should not. Therefore, the network is unable to work out what changes should be made to produce a better solution next time [4,41]. This difficulty can however be resolved by replacing the hard-limiting by a non-linear *(sigmoid)* function as the thresholding process. By using this function, the network would be able to determine when the relevant weights need to be strengthened or weakened. Many problems may be solved by using an important class of neural networks known as *multilayer feedforward* networks which are commonly referred to as *multilayer perceptron* (MLPs).

Multilayer perceptrons, are constructed from multiple layers of elements, neurons or nodes, which are quite similar to the simple perceptron discussed in section (2.2). This type of neural networks shown in Figure (2.2), consists of units that constitute the input layer, an output layer and a number of intermediate layers *(hidden layers)*.

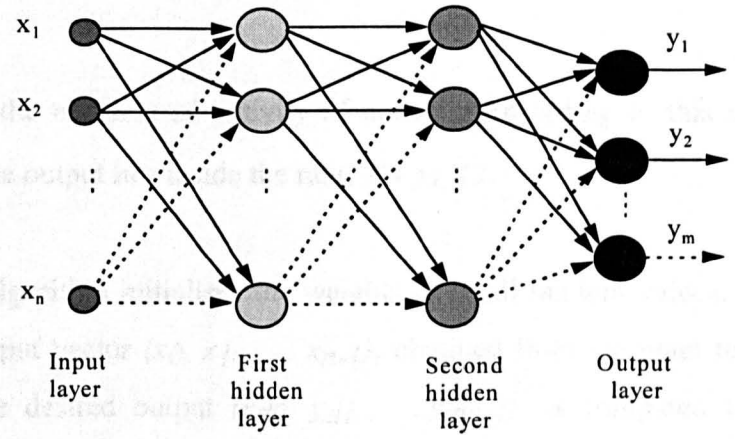


Figure 2.2, Multilayer perceptron diagram

The network requires a set of data as inputs to the input layer. The outputs of the input layer are then fed as weighted inputs to the first hidden layer. The outputs from the first hidden layer are fed as weighted inputs to the second hidden layer and so on. This process continues until the output layer is reached [22,42]. Multilayer perceptrons have been applied successfully to solve many problems such as motor speed and position control, speech and pattern recognition.

The most popular training technique used to find a set of weights is the *backpropagation (BP)* algorithm [1,22,32,42]. The backpropagation algorithm is a set of learning laws in which a training signal is presented to the input layer, passed through the hidden layers and then to the output layer to produce the actual response of the network. The actual output is then compared with the desired output to produce an error signal. Finally the weights of the network are adjusted to minimise the error signal. This procedure is carried out repeatedly adjusting the weights to make the actual response of the network closer to the desired response.

To examine the algorithm let us consider the network illustrated in Figure (2.2) with the particular output defined for node j by the function

$$y_j(t) = \frac{1}{1+\exp(-a_j(t))} \qquad -\infty < a_j(t) < \infty \tag{2.7}$$

where $a_j(t)$ is the net internal activity of node j . According to this non-linearity, the amplitude of the output lies inside the rang $0 \leq y_i \leq 1$.

To begin the algorithm initialises the weights to small random values, and presents the net with the input vector $(x_0, x_1, \ldots, x_{n-1})$, obtained from the plant to be modelled or controlled. The desired output $(y_{d0}, y_{d1}, \ldots, y_{dm-1})$, is compared to the calculated network outputs $(y_0, y_1, \ldots, y_{m-1})$, where n and m are the number of input and output

nodes, respectively . Then, a recursive algorithm is used which starts at the output nodes and works back to the first hidden layer, adjusting weights by the following adaptive expression:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \Delta_j x_i \tag{2.8}$$

where $w_{ij}(t)$ represents the weights from node i to node j at time t, η is the learning rate, Δ_j is an error term for unit j, and x_i is the actual input of unit i. If unit j is an output unit, then Δ_j can be computed as [53]:

$$\Delta_j = y_j(1-y_j)(y_{dj} - y_j) \tag{2.9}$$

where y_{dj} is the desired output of unit j and y_j is the actual output. If unit j is an internal hidden unit then

$$\Delta_j = x_j(1-x_j)\sum_k \Delta_k w_{jk} \tag{2.10}$$

where the sum is over the k nodes in the layer above j. Internal unit thresholds are adapted in a similar manner by treating them as connection weights on links from auxiliary constant valued inputs. The convergence can be improved if a momentum term is added and weight changes are smoothed [4,32,41]. This is shown in the equation below.

$$w_{ij}(t+1) = w_{ij}(t) + \eta\ \Delta_j x_i + \alpha(w_{ij}(t) - w_{ij}(t-1)) \tag{2.11}$$

where α is usually a positive number ($0 < \alpha < 1$) called the momentum constant. This factor controls the feedback loop acting around the weight changes $\eta \Delta_j x_i$ [22,32,78].

The process is repeated by re-presenting the *actual* and *desired* outputs, and is carried on till the error is reduced to a pre-defined limit.

In most cases the structure of the MLPs is carried out in a fairly heuristic way, so for a certain problem a reasonable number of layers and neurons in each layer are initially selected based on experience. However, if incorrect number of nodes are selected, then adjustments can be made on a trial and error basis. Moreover, the backpropagation algorithm suffers from several deficiencies, such as slow convergence and construction complexity [23,52,54].

An alternative approach to overcome the limitations associated with the BP algorithm is to use the Radial Basis Function (RBF) network which is discussed in detail in the following section.

2.3 Radial Basis Function Network

Feedforward layered neural networks have increasingly been used in many areas such as modelling and control of non-linear systems. One example of a feedforward neural network is the *BP* neural network, discussed briefly in the previous section. This form of neural networks has been applied in different fields of research and satisfactory performance has been obtained [69,85,92]. In practice, however, the BP algorithm has been found to perform poorly (e.g. slow convergence of weights in the non-linear updating procedure and difficulty in modelling differential response [86]). A viable alternative neural network is the *RBF* network. The RBF is surveyed by Powell (1985) [73], Broomhead and Lowe (1988) [13], and Moody and Darken (1989) [63]. Many other researchers are also exploiting the use of RBF in the design of neural network controllers. The first neural network controller was used by Broomhead and Lowe (1988).

The RBF network can be regarded as a special three layer network including input, hidden and output layers. Full explanations of the connections of these layers together with the activation function are given in the next sections. The performance of the RBF depends on the proper selection of three important parameters, *centres, widths and the weights.* The K-means clustering algorithm for selecting the centres and P-nearest neighbour for the width selection are discussed in detail in Section (2.5), whilst Singular Value Decomposition *(SVD)*, used for selecting the weights *off-line* is discussed in subsections 2.5.3.1. The least mean squares method *(LMS)* is used for adapting the weights *on-line* and is discussed in subsection (2.5.3.2). The conclusions to this chapter are outlined in section (2.6).

2.4 Radial Basis Function Network Structure

The radial basis function has been shown to be able to solve many problems in different fields; one example is the modelling and controlling of non-linear systems [23,39,44,86]. The RBF neural network has a feedforward structure consisting of three layers as shown in Figure (2.3).

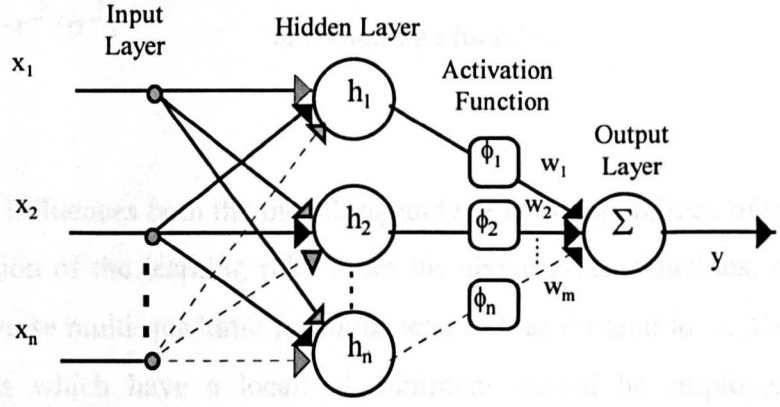


Figure 2.3, The radial basis function structure

The input layer passes the input data to the hidden layer. The hidden layer consists of an array of nodes and each node contains a parameter vector, called a centre. The output

layer is only a set of weighted linear combinations of the activation functions. The only adjustable parameters in the output layer are the weights. The output vector y is given by

$$y = \sum_{j=1}^{N} w_j \phi_j \tag{2.12}$$

where (w_j) is the weight of the (j^{th}) node and ϕ_j is the activation function. The transformation of the data from the input space to the hidden unit space is non-linear. The hidden node calculates the distance between the centre and the RBF network input vector, and then passes the result through the non linear activation function (ϕ) to the output layer. The RBF make use of various activation functions [19,20,23,60] and some typical choices are given below:

$\phi\ ((r) = r^2 log(r))$ the thin plate spline function

$\phi\ (r) = [r^2 + \sigma^2]^{1/2}$ the multi-quadratic function

$\phi\ (r) = \frac{1}{[r^2+\sigma^2]^{1/2}}$ the inverse multiquadratic function

$\phi\ (r) = exp\left(-r^2/\sigma^2\right)$ and Gaussian function

The chosen function influences both the modelling and the learning abilities of network, as well as the selection of the learning rule. From the above given functions, only the Gaussian and the inverse multi-quadratic functions tend to 0 as (r) tend to ∞. Therefore, only those functions which have a localised minimum should be employed when instantaneous adaptation rules *(e.g. LMS)* are used to train the weight vector [14]. The Gaussian activation function is *unique,* in a sense that it is the only radial function which can be written as a product of univariate functions

$$\phi_j(x) = exp\left(-\frac{\|x-cj\|}{\sigma_j^2}\right) = exp-\frac{\sum_{i=1}^{n}(x_i-c_j)^2}{\sigma_j^2} \qquad i=1,2,3,\ldots,n,\ j=1,2,3,\ldots,m \qquad (2.13)$$

where (ϕ_j) is the output of the (j^{th}) unit in the hidden layer, $r=|x-c|$ in which (x_i) is the input data to the network, (c_j) is the centre of the j^{th} unit in the input space and $\|\|$ is the Euclidean norm. Value (σ_j) is the width of the Gaussian function, (m) is the number of centres and (n) the dimension of the input space.

Equation (2.13) is easier to implement for use in Gaussian RBF networks with high-dimensional input spaces where only a small number of inputs are relevant. This factorisation also makes it possible to incrementally construct Gaussian RBF networks where each unit may depend on different inputs. Weighting functions (non exponential functions) shown above, can be used but do not guarantee a good approximation. As we will explain later, σ is an important variable and functions not using such a parameter are not preferred.

The major requirement is that the function must tend to zero quite rapidly as the distance increases between the input x and centre c. This can be assured by using the Gaussian exponential function. It can be shown that the exponential function possesses good approximation properties. Moreover, the Gaussian function is the most common basis and has been used by many researchers [14,16,35,50,89,93].

The operation of the network may be seen clearly by referring to Figure (2.4), which shows the simplified single neuron for RBF case, when only one input to the hidden layer is considered.

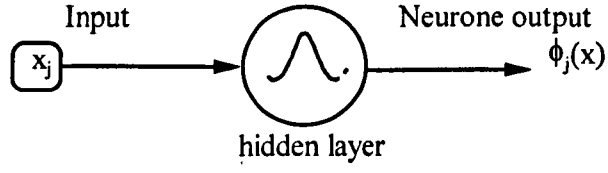


Figure 2.4, One input to single RBF

The Gaussian function (2.13) is applied to the given input x_j and the result showing the hidden output varying with respect to x is depicted in Figure (2.5).

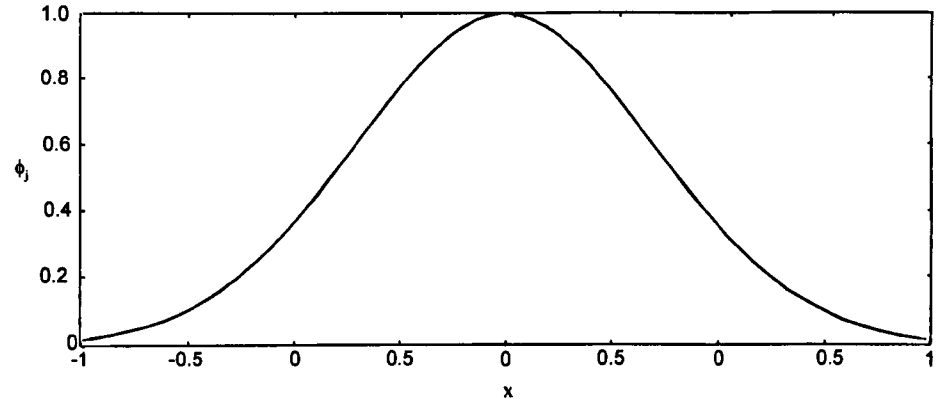


Figure 2.5, Hidden layer response function

At any instant, when the input value is equal to the centre value, the function output is 1.0, regardless of the value of σ. Thus, the centre values determine the value of the input which produces the maximum output from the neuron (hidden output). As shown in Figure (2.5) above, the response of the neuron at other values of x drops quickly as x deviates from c. If the value of x is far from the value of c, then the hidden layer output becomes negligible i.e. zero.

The Gaussian function can be extended to any number of inputs as shown in Figure (2.3) and equation (2.13). Multiple outputs can be added to the network, but this is not considered in this work, where only one output is needed [23,28,94].

2.5 Learning In The Radial Basis Function Network

The Radial Basis Function Network consists of three important parameters, centres *(c)* ,widths *(σ)* and weights *(w)*. The value of these parameters are generally unknown and may be found during the learning process of the network. There are a variety of methods to allow the RBF network to learn. These processes are generally divided into two stages,

as each layers of the RBF perform a different task [23,42,44]. The first learning stage involves selecting the centres and the widths in the hidden layer. The second stage is to adjust the weights in the output layer.

2.5.1 Selection of the Centres *(c).*

In general, the theoretical analysis of RBF networks assumes that the basis functions are distributed on an n-dimensional lattice, with the centre of a basis function occurring at every point on the lattice. Hence, in theory, the number of basis functions depends exponentially on n, in equation (2.13). In the first practical applications of RBF it was assumed there were as many basis functions as data points, and the centres of the basis functions were chosen to be the input vectors data points. In this case, if the amount of input data is large, then a correspondingly large number of basis functions are required [11,14]. This leads to lengthy calculations and may result in redundancy. One solution is to choose randomly selected training vectors as the centres. This idea may give unacceptable results, especially for small training sets which are not representative of the whole data. Therefore, the random method is inconsistent if small numbers of centres are employed [76]. The gradient descent method has been used for learning the centres. That is the centres are moved consecutively so as to reduce the error by the greatest amount. This procedure needs repeated training of the network output and causes slow learning in the output layer [2,42], therefore, selecting the centres in this manner is unsatisfactory and sometime gives unsatisfactory models. The major problem which therefore remains is one of how to select an appropriate set of RBF centres. To overcome this problem, the network requires some strategy for selecting the adequate set of centres, hence clustering algorithms have been used extensively. The objective of clustering algorithms is to categorise or cluster the data. The classes must be found from the correlation of an input data set. So, the clustering is a way of grouping similar patterns and separating dissimilar (different) ones. Assume there is a set of data to be used as input to an RBF network and no information is known about the number of classes that may be present in this set.

Clustering in such a case involves identifying the number of classes and assigning individual datum membership of these classes. The vectors in the same cluster are similar which means that they are close to each other in the input space. There are many clustering algorithms which have been used: such as Kohonen feature map [33] and adaptive resonance theory [41] which are discussed briefly in the following sections. The importance of using each algorithm may vary from application to application. Moreover, it is difficult to compare the variety of different algorithms because the comparison depends on the criteria used to evaluate the final clustering. However it seems that the well known K-means clustering is a good algorithm. Therefore this clustering technique is presented in this theses.

2.5.1.1 Kohonen Feature Map

The kohonen feature map has been developed by Teuvo Kohonen in 1982 [33] and has been applied to a large variety of problems i.e. biological modelling. The algorithm is summarised as in Table (2.1). For each training set $\{x_1, x_2,\ldots,x_N\}$ it is possible to find the closest output **y** and move the weights of **y** and the weights of its neighbours closer to **x**. The weights are adapting according to the equation in Table (2.1) step 3.

Table 2.1, Summary of Kohonen algorithm

-Specify the number of iterations K
-Define the step size for each iteration by $\alpha(k) = \left(1 - \frac{k-1}{K}\right)$ where k is a single iteration
1- Initialise the weights to random value
2- Select a random training set X at the starting of iteration k
3- By using of the Euclidean distance, find the output y with weights W closest to X.
4- Adjust the weights W at each iteration; $W = W + \alpha(k)\{X - W\}$
5-Go back to step3 for the next iteration and increase k by one.

At each iteration weights are adjusted using a decreasing step size α, where α is a small constant selected heuristically, usually between 0.1-0.7. The Kohonen feature map has a limitation when it has been demonstrated to produce an acceptable result only for the simplest problems [33,96].

2.5.1.2 Adaptive Resonance Theory

Adaptive resonance theory (ART), is one of the important artificial neural network clustering algorithms. This algorithm was proposed by Grossberg in the 1976 [33]. ART has been developed into a series of different algorithms e.g. ART1, ART2 and ART3. However, ART1 network that applies to problems of learning and clustering is considered. This approach learns clusters in an unsupervised mode and it can accommodate new clusters without affecting the storage or recall capabilities for clusters already learned. The network produces clusters by itself if such clusters are identified in input data and stores the clustering information about patterns without *a priori* information about the number of clusters. The training procedure of ART1 is explained briefly as follows. Every training iteration consists of taking a training example U and examining existing prototypes weight vector W that are sufficiently similar to U. Then, according to the following conditions the decision will be taken about the clusters. Thus, if a prototype W is found to match the training example U, then U is added to clusters represented by W and W is adjusted to make it better match U. Otherwise U becomes the prototype for a new cluster. This approach nicely integrates clustering but on the other hand by using it problems caused by noise might also be amplified. ART1 networks are restricted to binary values $\{0,1\}$ of the input values U and to the networks weights W. For more details about ART algorithms see [41,96].

2.5.1.3 K-means Algorithm

The choice of a subsets of data as centres for the radial basis function is a very important task, since network performance relies upon good generalisation. The K-means clustering algorithm is used for selecting the values of the centres, the number of which must be decided in advance [59,62,63]. This algorithm has been shown in the literature, because of its simplicity and ability, to produce good results [23,50,54,64]. The K-means finds a set of cluster centres and partitions the training data into subsets. Each cluster centre is associated with one of the H hidden units in the RBF network. The data is partitioned such that the training points are assigned to the cluster with the nearest centre. The algorithm finds a local minimum in terms of the total squared Euclidean distances between the training points assigned to each cluster and the cluster centres $\hat{c}_h$. This minimum is known as $E_{cluster}$ defined as

$$E_{cluster} = \sum_{h=1}^{H} \sum_{k=1}^{K} B_{hk} \left\| \hat{c}_h - c_k \right\| \qquad h=1,2,3,..,H, \quad k=1,2,3,...,K \tag{2.14}$$

where B_{hk} is the cluster partition or membership function which is a $H \times K$ matrix, H is the maximum number of hidden layer and K is the number of centres in each hidden layer. The centre of each cluster is initialised to a different randomly chosen training point. Then each training example is assigned to the unit nearest to it. When all training points have been assigned, the average position of the training points for each cluster are found and the cluster centre is moved to that point. The procedure is repeated until the data sets converge, or in other words there is no further change in the grouping of the data points. Once this has happened, each cluster is associated with one radial basis. The revised cluster centres become the unit centres $\hat{c}$ of the RBF units [23,33]. To show the clustering results of this algorithm a hypothetical exampleis considered, with one input node, three centres and one output node. The input data clustered as depicted in Figure (2.6).

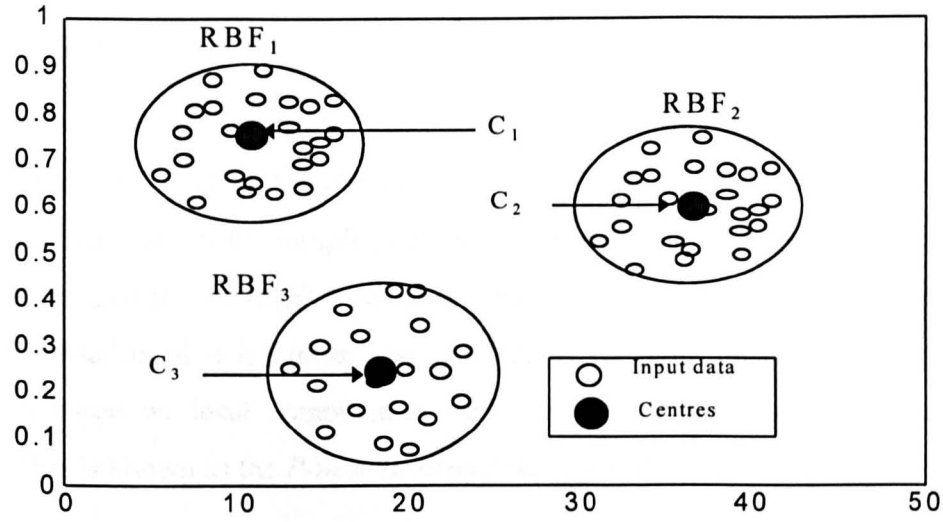


Figure 2.6, Clusters data using K-means algorithm

If some input data is far away from the selected centres, then one possible solution is to increase the number of centres.

2.5.2 Width selection *(σ)*

The next task would be the proper selection of the width between the centres. This parameter controls the amount of overlapping of Gaussian Function as well as the RBF network generalisation, i.e., the degree of the network approximation. The width of the Gauassian Function is a positive constant that represents the standard deviation of the function. RBF neural networks with the same σ in each hidden unit have the capability of universal approximation [64]. This suggests that one may simply use a single global fixed value σ for all the activation functions of the hidden layer. Moreover, using centers suggested that the width of Gaussian RBF may be fixed at $\sigma = \frac{d}{\sqrt{2M}}$, where *(d)* is the maximum distance between the centers and *(M)* is the maximum number of centres [42]. Empirical results (Moody and Darken, 1989) suggest that a good estimate for the global

width parameter is the average width $\sigma=\|c_i - c_j\|$. This represents a global average over all Euclidean distances, between the centre of each unit *i* and that of its nearest neighbour *j*. In general, the choice of σ is crucial in the estimation process. If σ is too small then, it will give sharp peaks at the sample point and so yield a rapidly decreasing function . If σ is too large, then it will result in a more gently varying function. So, the selection of standard deviation of σ is not an easy procedure. To avoid these two extremes, other heuristics based on local computations may be used which give individually tuned widths. This is known as the *P-nearest neighbour algorithm* [38,42,76].

2.5.2.1 P-nearest neighbour

The P-nearest neighbour algorithm is used for training the widths of the hidden nodes of an RBF network. Note that the width adjustment does not depend on the inputs directly, but depends on the selected centres. The P-nearest neighbour algorithm attempts to set the width of each node to the root mean square value of the Euclidean distances between each node and its P-nearest neighbours. It is given by

$$\sigma_i = \left(\frac{1}{P} \sum_{f=1}^{P} \|\hat{c}_i - c_f\| \right)^{1/2} \qquad f=1,2,\ldots P. \tag{2.15}$$

where $\hat{c}_i$ are the P-nearest neighbours of (c_f), σ is the Gaussian width and P is the index of the P-nearest neighbours. The most important point in selecting the width of the Gaussian function is to find a reliable way of measuring the distance between the centres. Various methods have been used for this purpose and some of them are discussed below.

Hamming distance measure:

The Hamming distance is widely used for measuring the distance between two vectors :

$$X_f = [x_1, x_2, \ldots x_n] \tag{2.16}$$

$$C_f = [c_1, c_2, \ldots c_n] \tag{2.17}$$

This distance is found by evaluating the difference between each corresponding component of the two vectors, and summing these differences to provide an absolute value for the variation between the two vectors. The measure is defined by

$$H = \sum_{f=1}^{n} \left(\left| c_f - x_f \right| \right) \qquad f = 1, 2, 3, \ldots, n \tag{2.18}$$

where n is the dimensionality of the vectors C and X and H is the Hamming distance. This distance is often used to compare binary data, i.e., the data used by the exclusive-OR function.

The square and city block distance:

These two methods are similar to Euclidean distance, but they perform the Euclidean measure without calculating the square root functions making them much faster but less reliable. Thus, the city block distance is given as:

$$D_{cb} = \sum_{f=1}^{n} \left| c_f - x_f \right| \tag{2.19}$$

and the square distance equation is

$$D_{sq} = MAX \left| c_f - x_f \right| \tag{2.20}$$

where MAX is defined as the maximum of the differences between each measured element [4]. So, equation (2.20) measures the largest distance between the two vectors (C and X).

Euclidean distance measure

The most common method used is the Euclidean distance measure. This method is widely used because it is simple to calculate and more reliable as compared to the above methods [54,86]. The use of this method can be explained by an example in a rectangular co-ordinate system. Consider Figure (2.7), the measured distance is between vectors X and C. The shortest distance between these vectors is the Euclidean distance defined as:

$$E_{dist}=\left[\sum_{i=1}^{n}\left(x_i-c_i\right)^2\right]^{1/2} \tag{2.21}$$

where n is the vector dimension and E_{dist} is the Euclidean distance. For the n-dimensional example, equation (2.21) gives :

$$E_{dist}=\sqrt{\left(x_1-c_1\right)^2+,\ldots,\left(x_n-c_n\right)^2} \tag{2.22}$$

In the same manner, the measure has been used in neural network learning algorithms. Thus, assume x_i and c_j denote an N×1 input and centre vectors, respectively

$$x_i=\left[x_{i1},x_{i2},\ldots,x_{iN}\right]^T \tag{2.23}$$

$$c_j=\left[c_{j1},c_{j2},\ldots,c_{jN}\right]^T \tag{2.24}$$

where the superscript T denotes transpose and N is the input dimension. The Euclidean distance between a pair of N×1 vectors x_i and c_j is defined by

$$E_{dist}(ij)=\left\|x_i-c_j\right\|=\left[\sum_{n=1}^{N}\left(x_{in}-c_{jn}\right)^2\right]^{1/2} \tag{2.25}$$

where x_{in} is the n^{th} element of the input vector x_i and c_{jn} is the n^{th} element of the centre vector c_j. In conclusion, the similar points (x_i, c_j), should produce a similar representation in a neural network and would therefore be classified as belonging to the same category [4,42].

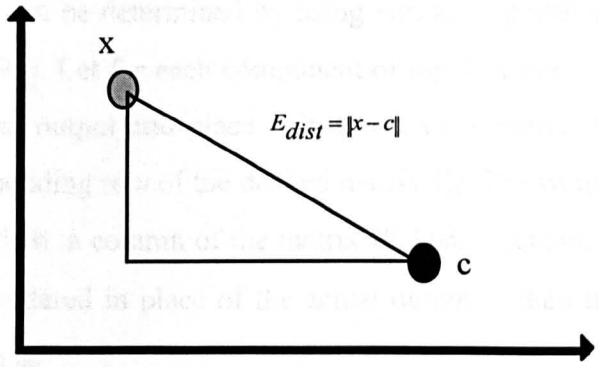


Figure 2.7, Euclidean distance

2.5.3 Output layer Learning

After the centres and widths of Gaussian function are found, the next step would be the estimation of the weights of the linear combination at the output layer. Thus, the output layer weight matrix *(W)* can be optimised by using a supervised training algorithm. A training data set must be available that is composed of pairs of vectors called input and

target *(desired)* vectors. The target vector indicate the desired output for the network when the associated input vector are applied. The training process consists of the following steps:

1- Apply the input vector *(x)* from the training data set to the input layer.
2- Calculate the outputs of the hidden layer using equation (2.13), which are then the inputs to the output layer.
3- Compute the RBF network output vector *(y)*. Compare this to the desired vector (y_d), then adjust the vector W so as to reduce the difference.
4- Repeat steps 1 to 3 for each vector in the training set.
5- Repeat steps 1 to 4 until $(Y\text{-}Y_d)$ tends to zero or other terminating conditions occur.

The weight parameters can be determined by using various algorithms e.g., LMS, RLS, SVD etc. [32,47,48,54,91]. Let for each component of input vector x in the training data set, calculate a Gaussian output and place it in a row of a matrix *(G)*. Also place the vector y_d in the corresponding row of the desired matrix Y_d. The weights associated with the Gaussian output is then a column of the matrix W. Using equation (2.12) and letting $G = \phi$ and if y_d is considered in place of the actual output y, then the equation can be rewritten in matrix form as

$$Y_d = GW \tag{2.26}$$

or

$$W = G^{-1}Y_d \tag{2.27}$$

where G^{-1} is the inverse of G. However, the matrix G can not be guaranteed to be square so it is not invertible and therefore, only its pseudoinverse can be found (if it exists). Finding the pseudoinverse involves inverting a matrix which may be ill-

conditioned *(singular or nearly so)* and cannot be accurately inverted. However, *Singular Value Decomposition (svd)* can be used to approximate it. This will be briefly explained in the following section.

2.5.3.1 Singular Value Decomposition (svd).

In many cases some methods for calculating the pseudoinverse of a matrix such as the lower and upper triangular matrices (LU) fail to give satisfactory results [36,42,91]. Singular value decomposition is a very powerful technique and has the ability to deal with a set of data or matrices e.g., Gaussian Outputs. The *svd* algorithm is an efficient *off-line* procedure and will be used to obtain the weights *off-line*. The *svd* matrices is given by

$$\begin{bmatrix} \\ \phi \\ \\ \end{bmatrix} = \begin{bmatrix} \\ U \\ \\ \end{bmatrix} \cdot \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \cdot \begin{bmatrix} \\ V^T \\ \\ \end{bmatrix} \tag{2.28}$$

If $\phi = m \times n$, then $U = m \times n$ left singular matrix *(orthogonal)*, V is an $n \times n$ right singular matrix and λ_i are the singular values of the matrix ϕ.

With respect to RBF networks, ϕ_j are the activation functions outputs as shown in Figure (2.3) and equation (2.13). The singular value decomposition algorithm coded in Matlab, is used to compute the *svd* matrix as *[U,S,V] = svd (rbf,0)* which produces the economy size decomposition.

where $U = \{u_{ij}, \ldots, u_{nm}\}$, $i = 1,2,3,\ldots, n$, $j = 1,2,3,\ldots, m$

$V = \{v_{ii}, \ldots, V_{nn}\}$

and λ is $n \times n$ with nonnegative diagonal elements in decreasing order; $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n \geq 0$. Where m is the length of the input vector of the RBF network and n is the size of the hidden layer.

Now from equation (2.28), the inverse of matrix ϕ is given by

$$\phi^{-1} = V[diag(1/\lambda_i)]U^T \tag{2.29}$$

Assuming $W = \phi^{-1} y_d$ using equation (2.27), the equation (2.29) may be rewritten as

$$\begin{pmatrix} \\ W \\ \\ \end{pmatrix} = \begin{pmatrix} \\ V \\ \\ \end{pmatrix} \cdot \begin{pmatrix} 1/\lambda_1 & 0 & \cdots & 0 \\ 0 & 1/\lambda_2 & \cdots & 0 \\ 0 & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 1/\lambda_n \end{pmatrix} \cdot \begin{pmatrix} \\ U^T \\ \\ \end{pmatrix} \cdot \begin{pmatrix} \\ y_d \\ \\ \end{pmatrix} \tag{2.30}$$

To clarify the process, equation (2.30), can be rewritten to express any matrix W, as a sum of outer products of columns of V, rows of U^T and columns of $y_d{}^T$ with the weighting factors being the singular values λ_i. The vector $y_d = n \times 1$ and the vector $W = n \times 1$. Assume the RBF network has only one output. Therefore, the weights are calculated as

$$W^{new} = W^{old} + V \frac{1}{\lambda} U^T y_d T \tag{2.31}$$

The objective being that of finding a set of weights that minimise the squares of the errors between the desired and the actual RBF network outputs, i.e.

$$E(t) = \frac{1}{2} \sum_{j=1}^{N} (y_d(t) - y(t))^2 \tag{2.32}$$

where $y_d(t)$ and $y(t)$ are the desired and the actual outputs of neural network, respectively, t is the varying time and N is the number of input data set.

2.5.3.2 Least Mean Squares Method (LMS)

In our work the *feedforward* neural network was considered. In this network, the proposed learning procedure involves the presentation of a set of system's *input-output* pairs. The network uses the input vector to produce the estimated *(actual)* output value compared with the true *(desired)* value to generate an error vector. If there is no difference, no learning takes place, otherwise, the weights are adjusted to reduce the error. The *LMS* rule for adapting the weights can be written as:

$$\Delta w_j = \eta\,(y_d(n) - y(n))\,g_j(n) = \eta\, e_j g_j(n) \qquad (2.33)$$

$$j = 1,2,\ldots C, \text{ and } \quad n = 1,2,\ldots, N.$$

where C is the number of centres, N is the number of inputs to the network, η is the learning rate, and y_d, y and g are the desired output, actual output and the hidden output *(Gaussian output)*, respectively. The $e_j = (y_{d_i} - y_i)$ is the error signal at the output and Δw_j is the change to the network weights.

The LMS method was originally introduced by Widrow and Hoff (1960's) [42] for use in adaptive switching circuits and is therefore known as the Widrow-Hoff rule or Delta rule. This method is similar to the well known *gradient descent* method [32,34,78]. It is considered to be an *on-line* method for adapting the weights of RBF networks.

The recursive process permits the weights to change *on-line* in accordance with LMS, that is

$$\hat{w}_j(n+1) = \hat{w}_j(n) + \eta\,(y_d(n) - y(n))\, g_j(n) \tag{2.34}$$

where $y(n)$ is the output of the network, g_j are the Gaussian outputs, $\hat{w}_j(n)$ are the previous weights originally set to zero and $\hat{w}_j(n+1)$ are the updated weights.

The adaptation algorithm described by equation (2.34) is illustrated in Figure (2.8).

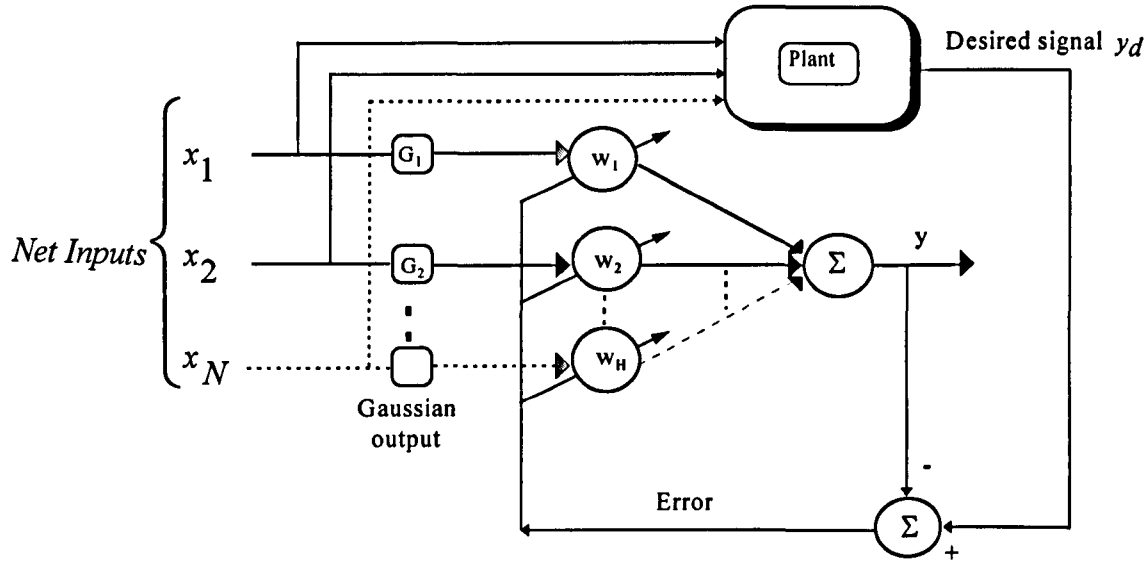


Figure 2.8, The adaptive LMS algorithm

Finally, a summary of this algorithm is presented in Table (2.2), which clearly illustrates the simplicity of the algorithm.

Table 2.2, Summary of the LMS algorithm

1- Initialisation of the weight vector.

$$\hat{w}_j(1) = 0 \quad for\ j=1,2,\ldots, H$$

2- Calculate the network output. *For time steps* $n = 1,2,\ldots,N.$

$$y(n) = \sum_{j=1}^{N} w_j(n) G_j(n)$$

$$e(n) = (y_d(n) - y(n))$$

$$\hat{w}_j(n+1) = \hat{w}_j(n) + \eta\,(y_d(n) - y(n))\, g_j(n)$$

2.6 Conclusion

The simple pereceptron neural network was discussed briefly. The learning rule was explained and the main limitation of perceptron was also mentioned. It was shown that this limitation can be solved by the introduction of the *Backpropagation algorithm.* The later was discussed and its disadvantages were stated. The *Radial basis Function (RBF)* neural network structure was shown and the parameter selection algorithms were discussed. Three clustering algorithms were introduced of which the K-means clustering algorithm was briefly explained along with an example of data clustering showing the ability of this algorithm. The selection of the widths together with different ways of finding the distance have been illustrated. The method of adjusting the weights of the RBF network off-line was also given and the on-line method discussed and summarised.

CHAPTER 3

DESIGN OF A NEW RADIAL BASIS FUNCTION

CHAPTER 3

DESIGN OF A NEW RADIAL BASIS FUNCTION

3.0 INTRODUCTION

There are different learning strategies that can be considered in the design of radial basis function networks. The original RBF method requires as many RBF centres as there are data points. This method is rarely practical in modelling and controlling of non-linear systems, as the number of data points is usually very large. The other approach is to select and fix the centres randomly from the training data set. In this case the Gaussian function has to be employed with its standard deviation fixed. An alternative method may be the self organised selection of centres, that is the RBF are permitted to move the locations of their centres in a self organised fashion regardless of the centre widths. The self organised component of the learning process serves to allocate network resources in a meaningful space where significant data are present. Note, the linear weights of the output layer are computed using a supervised learning rule. Hence, the selection of the centres affects both the adjustment of the widths and the learning of the output layer weights [42]. Improper selection of these parameters may give unsatisfactory results. Therefore, the choice of the correct number of the centres and the adjustment of the widths are not easy tasks. However, a *new algorithm* is presented in this chapter, which is capable of selecting these parameters adaptively. The optimisation of the proposed new algorithm and its summary are given in sections 3.1 and 3.2, respectively. In section 3.3 the signal flow diagram of the algorithm is shown with an explanation of its mechanism. An example and concluding remarks are given in sections 3.4 and 3.5.

3.1 Optimisation

In this section the new adaptive technique for the selection of the RBF parameters is presented. The selection of these parameters is divided in two stages [23,54]. The first

stage deals with the training of the hidden layer, for selection of the centres and the widths. The second stage is the learning process for adapting the weights in the output layer. Each learning problem is treated separately (i.e., there is no direct relationship between these two learning strategies).

The most commonly used methods for selecting the centres and the widths are the k-means clustering algorithm and the P-nearest neighbour technique [23,54,63]. When using k-means, the desired number of centres is usually determined a *priori* using equation (2.14). In this case, a trial and error procedure is used for selecting the optimum number of centres. The RBF width parameter, σ, is usually chosen as an average distance between the neighbouring centres. The number of these centres, however, is dependent on the index P of P-nearest neighbour given in equation (2.15). This should also be known *a priori*. The selection of the index P is dependent on the characteristics of the model to be learnt. Proper selection of these two parameters is not easy and depends upon the complexity of the system to be modelled or controlled. If one could adjust their value adaptively, however then an optimal solution may be found. Accordingly a ***new method*** is introduced to improve parameter selection. The new method is a merger of the two stages in one. It is composed of a number of algorithms *(e.g., k-means, svd, Gaussian function and P-nearest neighbour),* incorporated in a unsupervised adaptive algorithm.

Although heuristic in some sense, the algorithm will adaptively choose the number of centres, the widths and the weights respectively. Consider the following:

$$S_i = \sum_{i=1}^{ni} \sum_{r=1}^{nc-1} \sum_{j=r+1}^{nc} \left(g_{ir} - g_{ij} \right)^2 \tag{3.1}$$

$$CN = \left(\frac{nc.(nc-1)}{2} \right) \tag{3.2}$$

$$SAGO = \left(\frac{S}{CN}\right) \tag{3.3}$$

where, $i = 1,2,3,...,ni$, $r = 1,2,3,...,nc-1$ and $j = 2,3,...,nc$, in which nc is the number of centres and ni is the length of the input data vector. S is the sum of the squares of the distances between the Gaussian output sets at some amplitude level. CN is a calculation number indicating the horizon of the Gaussian outputs over which the value of S is averaged, $G = [g_{11}, g_{21}, g_{31}, \cdots, g_{nc,ni}]^T$ are the Gaussian's outputs and $SAGO$ is the Sum of the Squares of the distance between the Gaussian Outputs over the calculation number. The algorithm starts with an initial number of centres, which should be initially selected equal to or less than the input data vector length. This is not an important issue as the algorithm will adjust it approximately, however it can be selected based on the number of data elements in the system's difference equation. Then the Gaussian outputs are checked for overlapping which means the centres are too close to one another. If so, then the spread between the centres is increased. Otherwise, if the centres are underlapping (i.e., the centres are too far from each other) then the number of centres are increased by one. During the process of training, the number of centres are increased as well as the spread between them until an optimal solution is reached.

This algorithm monitors the Gaussian outputs as shown in Figure (3.1) and makes an intelligent decision as to whether to increase the number of centres or the index P [29]. Using the algorithm, overlapping between the centres may be avoided and the appropriate number of the centres and the widths are selected. The criteria would be in having some conditions on the net output which also insures convergence of the RBF output.

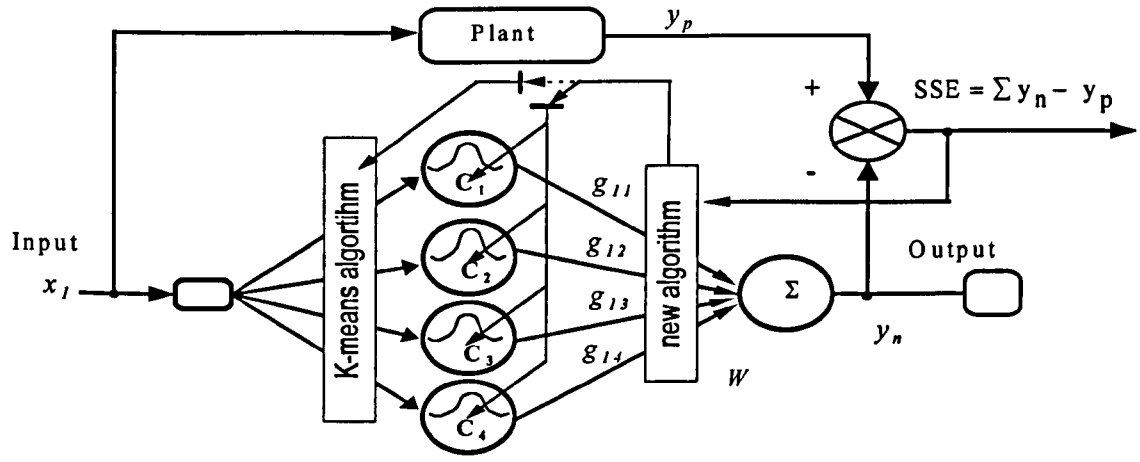


Figure 3.1, RBF network using the new algorithm

3.2 The Algorithm

1. Select the sample number *NS* of the input data for RBF network training.
2. Initialise the number of centres starting with a guess value $nc \leq ni$, and the distance index number, $P \leq nc$. Initialise $SAGO \leq 0$.
3. Compute the new *SAGO*, using equations (3.1-3.3), and compare it with the previous *SAGO*.
4. If new *SAGO* $\leq$ previous *SAGO*, then the centres are overlapping and the index p should be increased, otherwise go to step (5).
5. Using s*vd*, algorithm, find the values of weights W_i, use equation (2.31).
6. Calculate the sum squared error *SSE* between the plant and the network output, within the input samples window. If *SSE* is beyond the pre-set boundary, then increase the number of centres by one.
7. Repeat steps 3 to 6 until both conditions (new *SAGO* > previous *SAGO* and $SSE^{new} \leq SSE^{old}$) are satisfied.

3.3 Description of Algorithm

The algorithm is a routine which searches for the optimum solution of RBF parameters *(centres, width and weight)* automatically. The flow diagram in Figure (3.2) gives an outline of the algorithm.

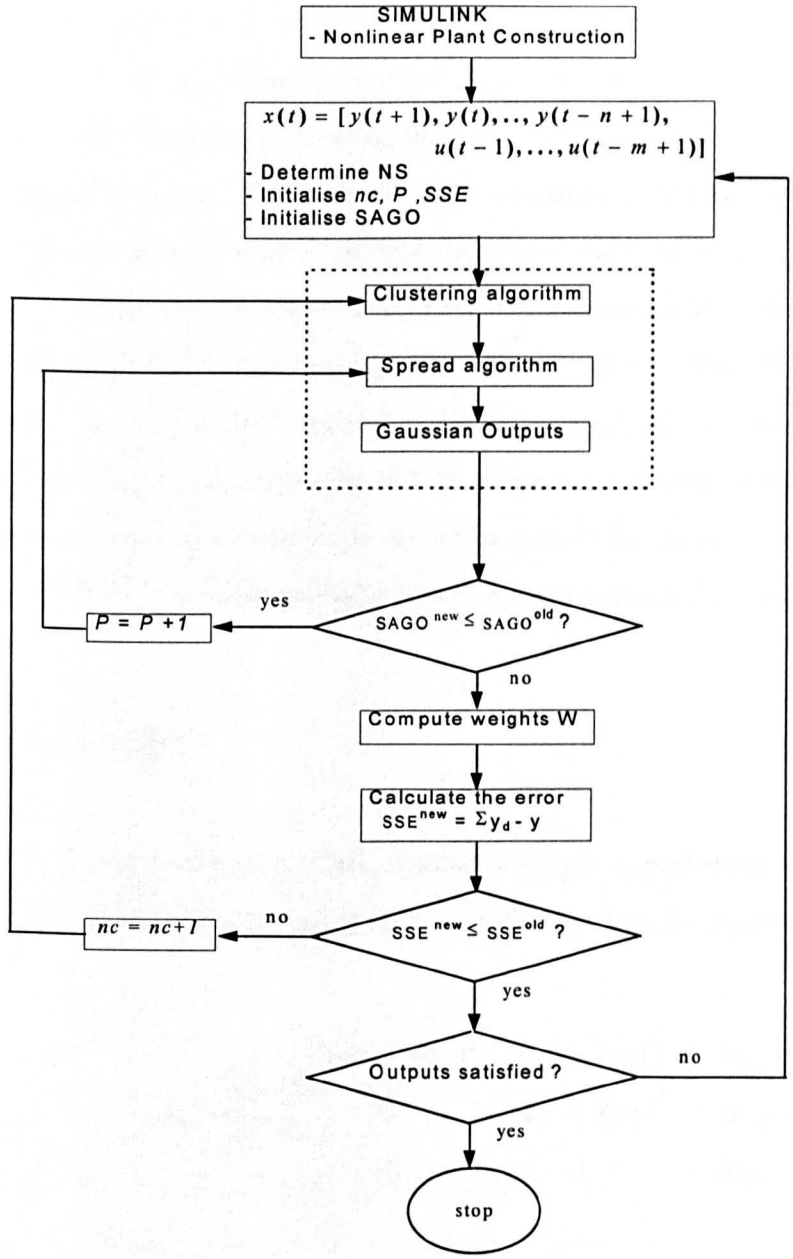


Figure 3.2, New algorithm flow diagram

The Simulink package is used for constructing the non-linear plant. In the Matlab environment, the network inputs, consisting of the plant *input-output*, including the delays $x(t) = [y(t+1), y(t), .., y(t-n+1), u(t-1), ..., u(t-m+1)]$, are determined by a function coded as Matlab programs. The number of network input samples *NS* is determined and *nc* and *P* are initialised to small values. The initial centres are calculated by using the K-means algorithm *equation (2.14)* and the width between the centres is generated using *equation (2.15).* Then, the Gaussian output is calculated using *equation (2.13)* and the algorithm progresses by initialising SAGO to a large value. *Equations (3.1-3.3)* are used for calculating the value of SAGO. If $SAGO^{new} \leq SAGO^{old}$, then this is sufficient to ensure that the centres are overlapping and the index *P* increases by one. If the above condition is not satisfied, then the weights of RBF network are calculated. The process continues and the sum squares errors between the desired and actual signals are calculated and the result is compared with the old one. If $SSE^{new} \leq SSE^{old}$ is true then the conditions are satisfied and the optimum parameters are obtained. Thus, the search is finished and the algorithm stops. Otherwise, *nc* is incremented by one, and the process continues until all possibilities are exhausted. If no possible solution is found, then the only solution is to increase the number of input samples *NS* and start a new search.

3.4 An Example

To demonstrate this algorithm, assume a simple hypothetical example when *nc* and *ni* are 4 and 5 respectively and rewrite equation (3.1) in the matrix form as:

$$S_1 = \sum_{r=1}^{3} \sum_{j=2}^{4} \begin{bmatrix} (g_{11}-g_{12})^2 + (g_{11}-g_{13})^2 & + (g_{11}-g_{14})^2 \\ 0 & + (g_{12}-g_{13})^2 & + (g_{12}-g_{14})^2 \\ 0 & 0 & + (g_{13}-g_{14})^2 \end{bmatrix}, \quad i=1 \qquad (3.4)$$

$$i=5 \qquad S_1 = \sum_{r=1}^{3} \sum_{j=2}^{4} \begin{bmatrix} (g_{51}-g_{52})^2 + (g_{51}-g_{53})^2 & + (g_{51}-g_{54})^2 \\ 0 & + (g_{52}-g_{53})^2 + (g_{52}-g_{54})^2 \\ 0 & 0 + (g_{53}-g_{54})^2 \end{bmatrix}, \tag{3.5}$$

and in general

$$S_i = \sum_{i=1}^{ni} \sum_{r=1}^{nc-1} \sum_{j=2}^{nc} \begin{bmatrix} (g_{ni,r}-g_{ni,r+1})^2 + (g_{ni,r}-g_{ni,r+2})^2 & + \cdots + (g_{ni,r}-g_{ni,n})^2 \\ \vdots & \vdots \quad \ddots \quad \vdots \\ 0 & 0 \quad \cdots + (g_{ni,nc-1}-g_{ni,nc})^2 \end{bmatrix} \tag{3.6}$$

Equations (3.4-3.6) are equivalent to equation (3.1) where $S(ni) = s_1 + s_2 + \ldots + s_{ni}$. Equation (3.2) gives the distances between the Gaussian outputs, and for the example given above, $\mathrm{CN} = \frac{4\times(4-1)}{2} = 6$, as can be verified by equation (3.4) and (3.5). Using the same procedure, if *nc* is 5, then $CN = \frac{5\times(5-1)}{2} = 10$ and so on. Note that this number depends only on the number of centres *nc* regardless of the length of input data vector *ni*.

To illustrate the outcome of the algorithm for the above example, consider the outputs illustrated in Figures (3.3) and (3.4). These have been obtained when training a Single Input Single Output *(SISO)* system. The RBF network consisted of one input node, 4 centres, one output node and 10 input samples. Figure (3.3) shows no overlapping between the centres. However, to achieve a better RBF output, it can be easily

demonstrated that an extra centre is needed, e.g. new *SAGO* ≤ previous *SAGO*. Addition of the fifth centre causes overlapping of the centre as it can be seen in Figure (3.4).

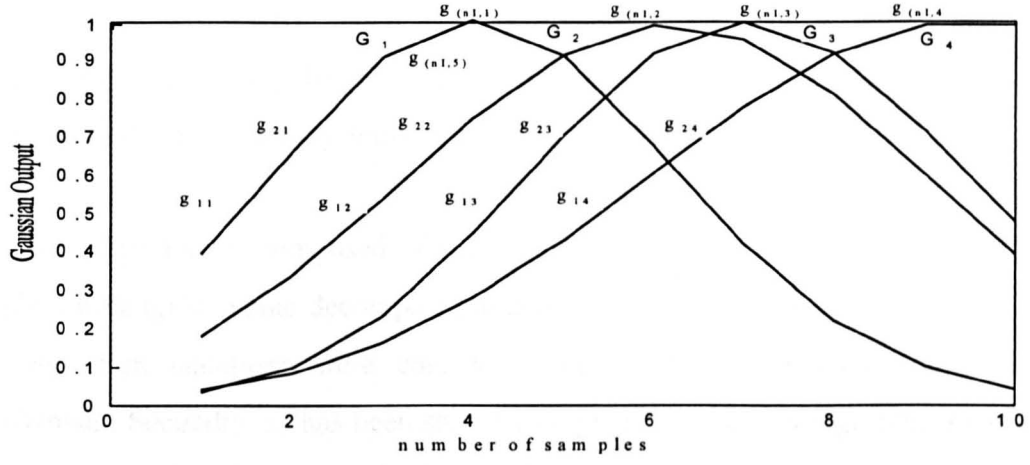


Figure 3.3, Non-overlapping centres

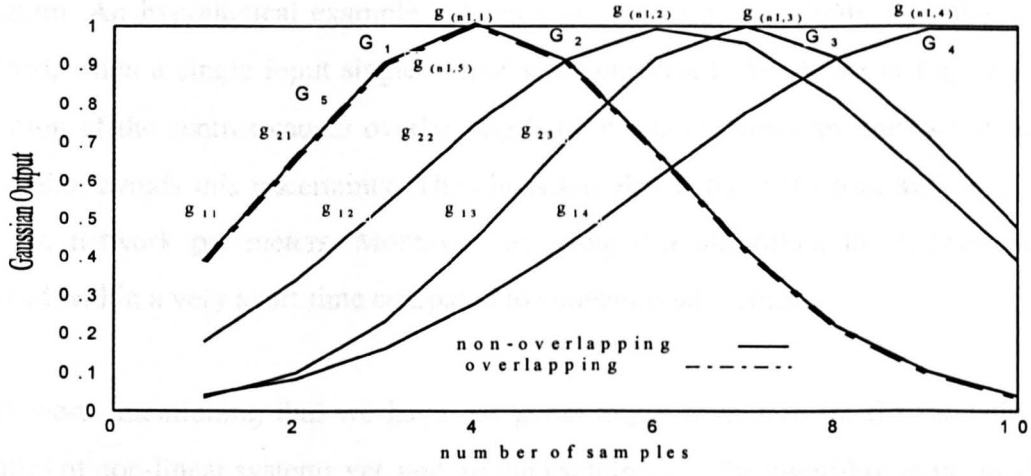


Figure 3.4, Overlapping centres

To avoid overlapping between centres, it is necessary to increase the index number, *P*. The alternate selection of *nc* and *P* has to be repeated several times in order to obtain a reasonable solution. However, the use of the new algorithm avoids this uncertainty. This is illustrated by a simulation study in chapters ***four*** and ***five***.

3.5 Conclusion

The general scope of using the new algorithm, is to select the minimum number and the proper values of the RBF neural network parameters i.e. centres, widths and weights adaptively. Particularly, the algorithm has been developed and tested to obtain the parameters of a non-linear systems model.

The new algorithm is composed of many other algorithms, such as K-means, P-nearest neighbour, singular value decomposition algorithms and the Gaussian function. Firstly, the algorithm equations were considered and explained to show the algorithm's mechanism. Secondly, as has been shown in Figure 3.1, the new algorithm monitors the Gaussian outputs and makes an intelligent decision to increase or decrease the number of centres or the index P.

For simplicity, the algorithm was summarised in seven steps and described in a flow diagram. An hypothetical example has been given in a matrix form, then the network trained, when a single input single output was considered. As shown in Figure 3.4, the addition of the centres causes overlapping between them. However, the use of the new algorithm avoids this uncertainty. The algorithm also reduces the trial and error search for the network parameters. Moreover, by using this algorithm, the network can be trained within a very short time compared to conventional methods.

It is worth mentioning that we have not given any information on the modelling and control of non-linear systems yet, and so the usefulness of the algorithm in the modelling of non-linear systems will be studied using conventional methods in chapter ***four***. The results of using the new algorithm for modelling the same system are shown in chapter ***five***. Finally, the benefits of the new algorithm are shown in chapter ***six***, when it used for selecting the parameters in a real servo system application.

CHAPTER 4

IDENTIFICATION OF NON-LINEAR DYNAMIC SYSTEMS USING ARTIFICIAL NEURAL NETWORK

CHAPTER 4

IDENTIFICATION OF NON-LINEAR DYNAMIC SYSTEMS USING ARTIFICIAL NEURAL NETWORK

4.0 INTRODUCTION

Artificial neural networks (ANN) have been widely studied and successfully applied in a variety of applications such as modelling and controlling non-linear systems. In many engineering solutions the *Backpropagation (BP)* network algorithm is used. This algorithm, however, has many problems, some of them were stated in chapter two. The more popular type of neural network with control engineers is the RBF algorithm discussed in detail in the previous chapters. The increasing popularity of the RBF algorithm is due to its many distinctive advantages, these include best approximation, much faster convergence compared to MLP, the need for a small number of units in the hidden layer, and its simple network structure [63,72,86]. Therefore, these advantages motivated us to use the RBF network to model the *input-output* relationship of equation (1.15).

4.1 RBF Neural Network for the Identification of Non-linear Dynamic Systems

Most physical systems are dynamic in nature and the identification of such systems using input-output data will naturally involve dynamic elements. Four dynamic models were introduced, by Narendra and Partiasarthy (1990) for the representation of *single input-single output (SISO)* non-linear plant. These models have been used in the adaptive systems literature for the identification of linear systems and can be extended and applied also to non-linear systems. The most general model is described by the following non-linear difference equation [66,67]:

$$x(t+1) = f\left[x(t), x(t-1), \ldots, x(t-n+1); u(t), u(t-1), \ldots, u(t-m+1)\right] \tag{4.1}$$

where *[u(t), x(t)]* represents the input-output pair of SISO plant at time *t* and the function *f* is assumed to be a differentiable function of its arguments. The output of equation (4.1), at any instant, is a non-linear function of the past values of both the input and the output. The representation of the model using tapped delay lines (TDL) is shown in Figure (4.1).

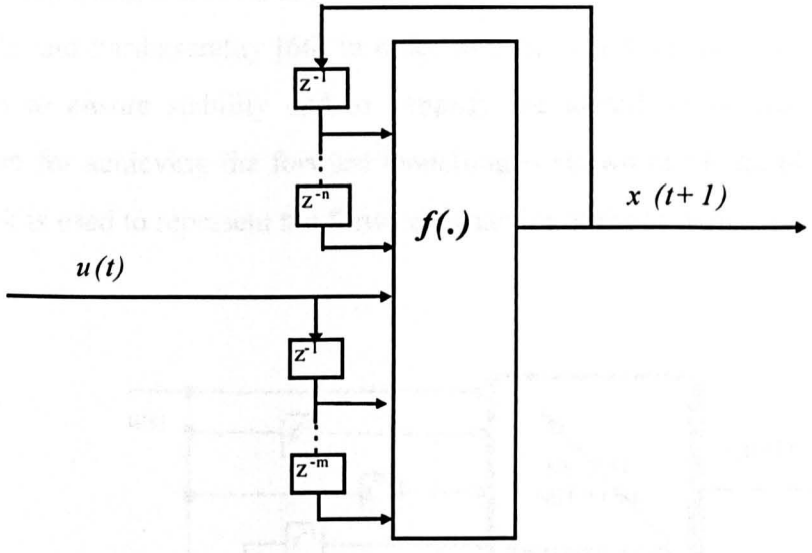


Figure 4.1, Representation of Model (4.1)

In each case the RBF is assumed to contain three layers with a sufficient number of nodes in each layer, so as to match the input-output characteristics of the corresponding non-linear mapping in the given plant. To identify the plant, an identification model is chosen based on a *prior* information concerning the class to which it belongs. The architectures for training the RBF networks to represent non-linear dynamic systems will be discussed in the following section.

4.1.1 Forward Modelling

The problem of identification, consists of setting up a suitably parameterized *identification model* (see equation (2.4)) and adjusting the parameters of the model to optimise a performance function based on the error *e(t)* between the plant input and the identified model output. For modelling the plant, the structure of the neural network is judiciously chosen. Hence, for plant representation the *series-parallel* model is used, this type of modelling is known as *forward modelling*. This model structure has been used by Narenda and Parthasarathy [66] in order to avoid many of the analytical difficulties, as well as to ensure stability and to simplify the identification procedure [57,67]. The structure for achieving the forward modelling is shown in Figure (4.2), where the RBF network is used to represent the forward dynamics of the system.

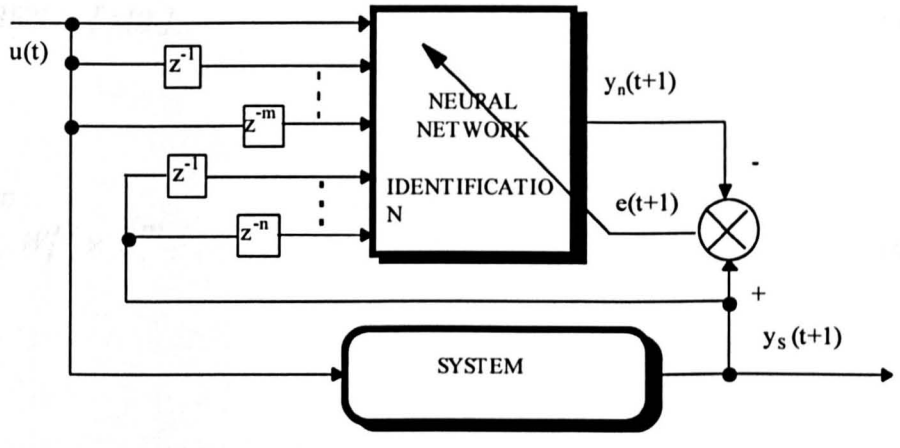


Figure 4.2, Series-parallel identification

As shown in Figure (4.2), the system is placed in parallel with the neural network model and at each instant of time *(t)* the past *(m)* inputs and the past *(n)* outputs of the system are fed into the RBF neural network.

The system is governed by the following non-linear difference equation:

$$y_S(t+1) = f\left[y_S(t),. \quad . \quad .,y_S(t-n+1),u(t),u(t+1),. \quad . \quad .,u(t-m+1)\right] \tag{4.2}$$

where $y_S(.)$ is the plant output and $u(.)$ is the input sequences. The neural based identification model is assumed to have the same structure as that of the plant and is given by:

$$\hat{y}_m(t+1) = \hat{f}\left[y_S(t),. \quad . \quad .,y_S(t-n+1);u(t),u(t+1),. \quad . \quad .,u(t-m+1)\right] \tag{4.3}$$

Where $\hat{f}(.)$ represents the non-linear input-output map of the network. Thus, from the input-output representation (4.3), in which $\hat{f}(.)$ is replaced by the hidden layer feedforward network model *(RBFNm)*, the neural-based identification model output may be described as:

$$\hat{y}_m\,(t+1) = RBFN\,m\,[\,x(t)\,] \tag{4.4}$$

or

$$\hat{y}_m(t+1) = \sum_{i}^{N^m} W_i^m \times \phi_i^m \tag{4.5}$$

where

$$\phi_i^m = e^{-(x_i^m - c_i^m)/\sigma_i^m} \tag{4.6}$$

and the superscript m is a variable related to the plant model order, c_i^m are the RBF centres, σ is the width between the centres, N is the number of the hidden layer units (centres) and $x(t)$ is a data vector which represents the present and past plant outputs and inputs at sample time t as:

$$x^m(t) = [y_S(t),. \quad . \quad ., y_S(t-n+1), u(t), u(t-1). \quad . \quad ., u(t-m+1)]^T \tag{4.7}$$

By substituting equation (4.7) into equation (4.4), we can write the model output as

$$\hat{y}_m(t+1) = RBFN\, m\, [y_S(t),. \quad . \quad ., y_S(t-n+1), u(t). \quad . \quad ., u(t-m+1)]^T \tag{4.8}$$

The difference between the network output $y_m(.)$ and the next observation of the system output $y_S(.)$, is known as the prediction error $e(t)$, and is given by

$$e(t+1) = \hat{y}_m(t+1) - y_S(t+1) \tag{4.9}$$

The estimation of the prediction error is the well known *mean squares error (MSE)* (see equation (2.32)) . This mean value is minimised to correctly model the system [56].

The RBF centres and width parameters are adapted and the weights of the network are then adjusted to minimise the sum of the squared errors between the desired and approximated outputs. These parameters were selected by using the methods discussed in chapters 2 and 3. When the error is of a small value, e.g. $y_S(t+1) \cong \hat{y}_m(t+1)$, the network gives a good representation of the system and so the network parameters are adjusted and no more adaptation is required to train the network. Thus, the model output data lags may be feedback to the model itself to form a feedback function. In this model structure the input vector may be represented as

$$\hat{x}_m(t) = [\hat{y}_m(t),. \quad . \quad ., \hat{y}_m(t-n+1); u(t), u(t-1). \quad . \quad ., u(t-m+1)]^T \tag{4.10}$$

Hence, the output of the model can be realised by

$$\hat{y}(t+1) = RBFN\, m\, [\hat{y}_m(t),. \quad . \quad ., \hat{y}_m(t-n+1), u(t). \quad . \quad ., u(t-m+1)]^T \tag{4.11}$$

This model can be reconstructed to give the feedback representation as illustrated in Figure (4.3). The structure illustrated in Figure (4.3) is used in our simulation studies.

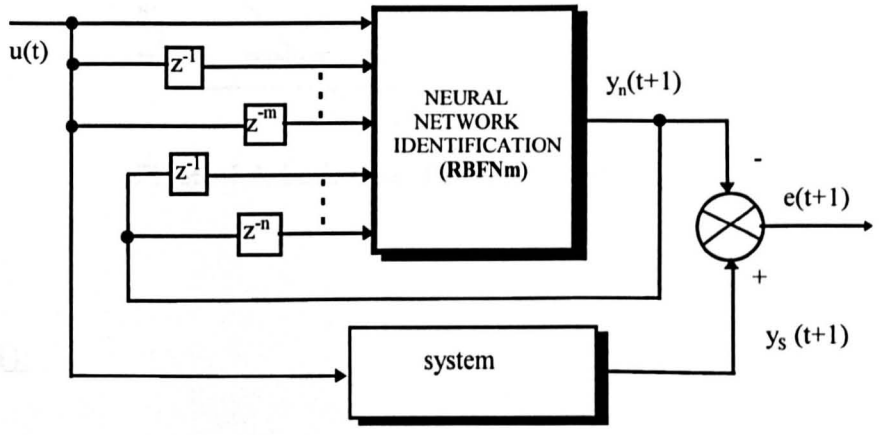


Figure 4.3, Feedback representation of the inputs-outputs of the RBF

4.1.2 Simulation Study

Simulations have been performed using the RBF neural network and different methods for selecting the parameters were considered. Various non-linear plants were tested and very good results were obtained. Two different plants were used as examples to show the properties of the RBF network. These examples have been studied by Narendra [66], Hunt and Sbarrbaro [44] and [1], where Backpropagation was applied. These examples have been chosen in order to obtain reference results to which the accuracy of the RBF results are compared. Figure (4.4), is a block diagram of a series-parallel model. The output of the plant and the network are compared and the resulting error is used to update the network parameters (centres, width and weights).

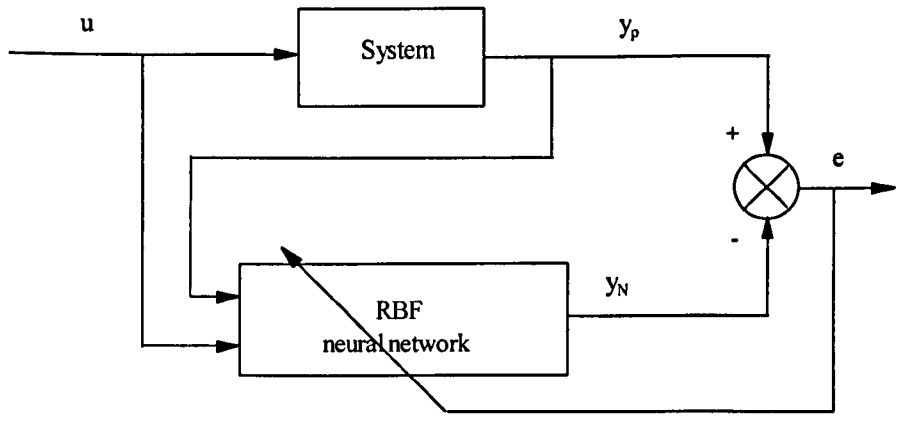


Figure 4.4, Series-parallel identification scheme

Example (1):

The plant considered for identification is a non-linear model governed by the difference equation

$$y_p(t) = 0.8y_p(t-1) + f[u(t-1)] \tag{4.12}$$

where the unknown non-linear function has the form $f[u] = (u - 0.8)u(u + 0.5)$. The system is simulated using the Matlab/Simulink package as shown in Figure (4.5). u(t) and $y(t)$ are the input and output to the network receptively. In this example the training signal is a sinusoidal wave with a frequency of 50Hz. The number of data points from the training signal is 3000, the number of input samples to the net is 60, the signal amplitude is 0.5 and the step size is 0.0001. The parameters of the model are selected in different ways as explained below to show the accuracy and effectiveness of each method.

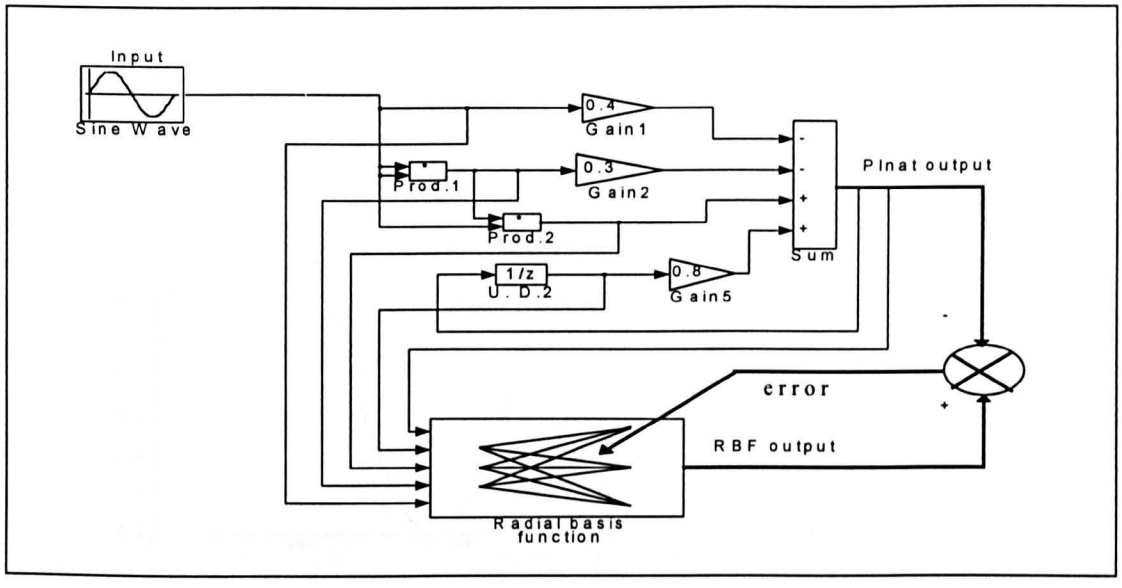


Figure 4.5, Simulink blocks setup of the non-linear model.

I- Random selection of the centres

The centres of the radial basis function network have been set to a random subset of 13 from the input set. The width parameter *(σ)* has been set by a guess to a value of 0.1. The centre vectors were arranged in a grid and the weights were adjusted by minimising a sum of squared errors function using singular value decomposition.

The training signal is shown in Figure (4.6), while the plant and the RBF output are shown in Figure (4.7). The difference between the net and the plant output signal (i.e., the error term) is shown in Figure (4.8). It can be seen from Figure (4.8) that the error level increases rapidly from the first samples and then settles to about 0.48. This error level is very high and is due to inadequate selection of the RBF parameters. This procedure, therefore, is undesirable.

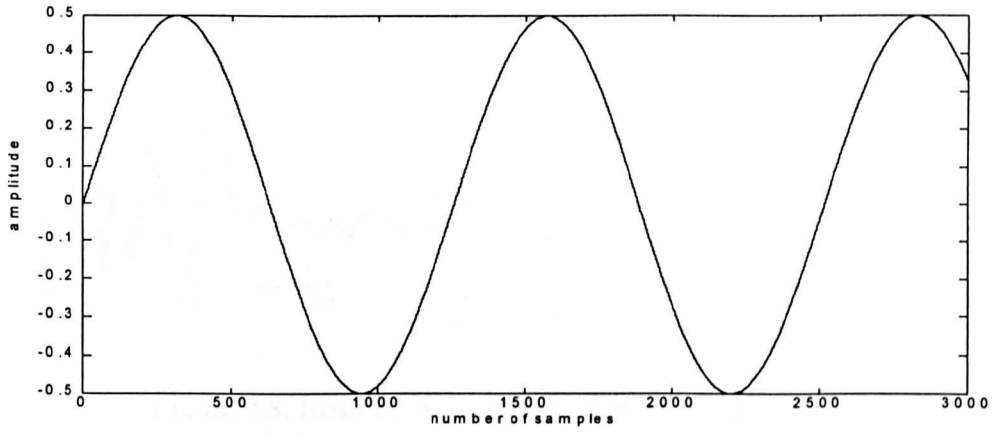


Figure 4.6, Input training data

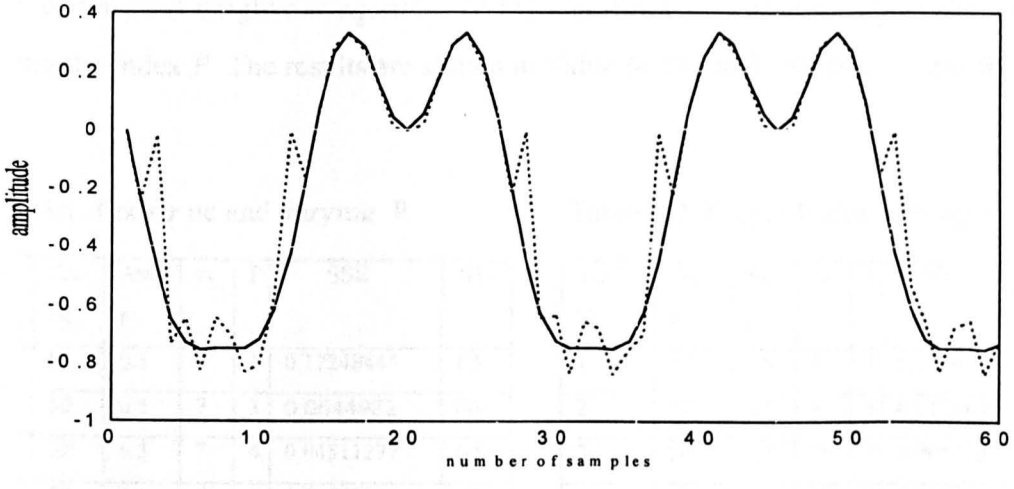


Figure 4.7, modelling the plant, random centres:

Plant output (solid line) and RBFoutput (dotted line)

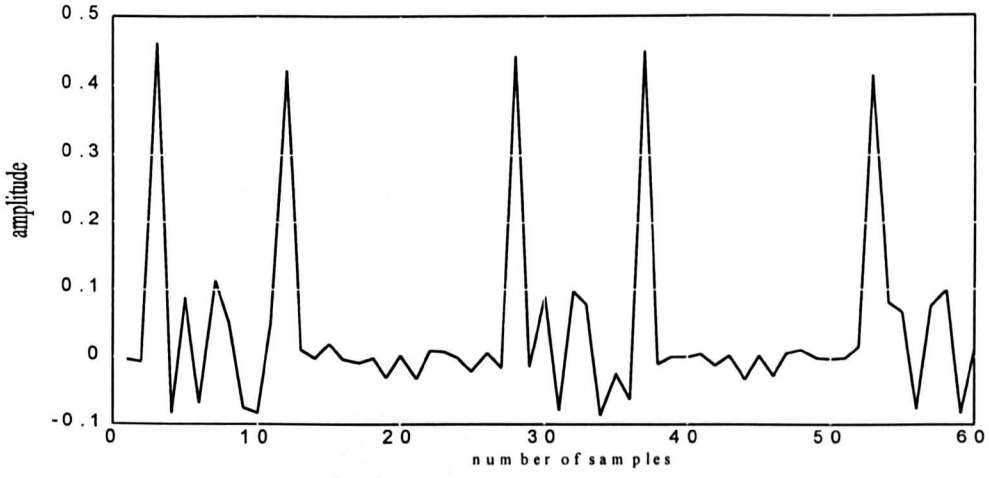


Figure 4.8, Error between RBF and Plant outputs

II- Clustering centres using a conventional method

Here, the K-means clustering algorithm in *equation (2.14)* was used to tune the centres and the P-nearest neighbour *equation (2.15)* algorithm was used to adjust the width by varying the index *P*. The results are shown in Table (4.1) and Table (4.2) respectively.

Table 4.1, Fixing nc and varying P

Trial No.	Fre. Hz	Amp	nc	P	SSE	NS
1	50	0.5	7	1	0.17248445	60
2	50	0.5	7	3	0.0544932	60
3	50	o.5	7	4	0.04511277	60
4	50	0.5	7	6	0.03693646	60

Table 4.2, Fixing P and varying nc

Trial No.	Fre. Hz	Amp	nc	P	SSE	NS
1	50	0.5	5	1	0.216634	60
2	50	0.5	8	1	0.0170851	60
3	50	o.5	10	1	0.001315	60
4	50	0.5	15	1	0.00051171	60

In case (1) the number of the centers *(nc)* selected was 7 and in case (2) the index *(P)* parameter was set to unity. After the nets were trained, the system and the RBF net outputs were compared, as shown in Figure (4.9), using the structure in Table (4.1). As

shown in Figure (4.10), the error settles to within ± 0.05 (i.e., at 14.285% of the plant output amplitude), this result is satisfactory but not optimal. However, the method is inconsistent especially if the system is more difficult to identify. In these situations several trial runs will have to be carried out and in some cases no optimal solution may be found.

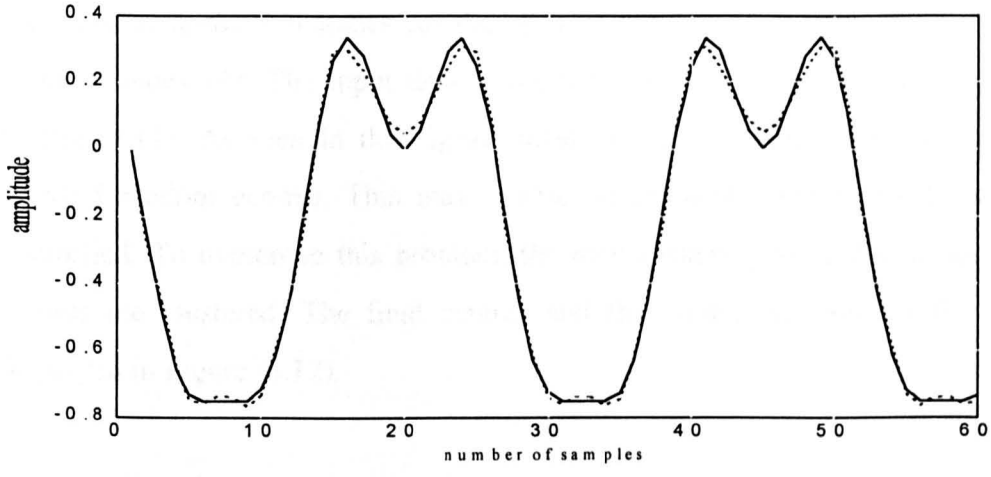


Figure (4.9), modelling the plant, using conventional method:
Plant output (solid line) and RBFoutput (dotted line)

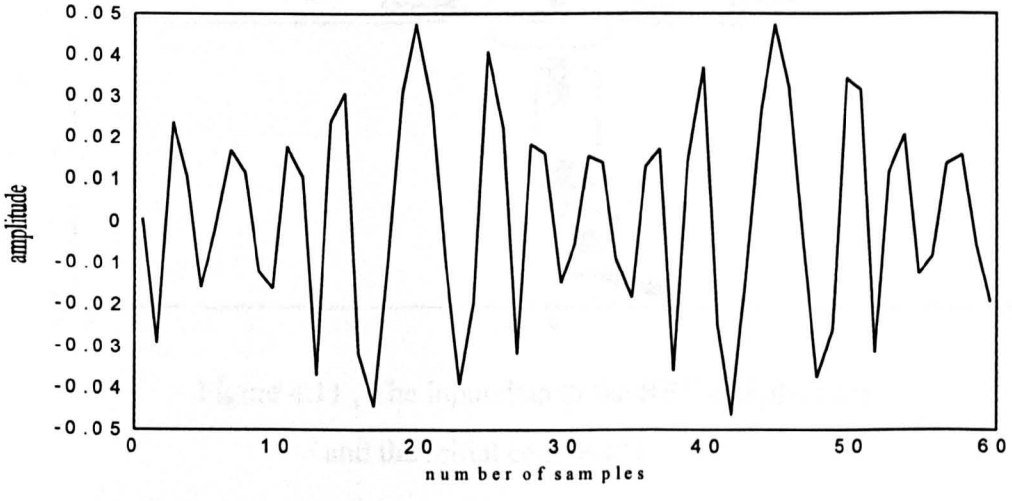


Figure 4.10, The error between the RBF and the plant outputs

III- Clustering the centers using the new algorithm

In this method, the net was trained using the new algorithm, *equations (3.1)-(3.3)*, in which the optimal number of centers *(nc)* and index *P* were selected. The process is carried out only once and so the trial and error situations are avoided. An initial guess was made of 4 for the number of centres and 2 for the value of index *P*. The same number of samples were used as in cases I, II and III with the same training signal waveform. The algorithm *adaptively* searches for the optimal number of centres *(nc)* and the optimal value of index *(P)*. The input data to the network and the initial centres are shown in Figure (4.11). As seen in this figure, most of the data points are far away from the selected random centres. This may cause unsatisfactory results to the system to be controlled. To overcome this problem, the new clustering algorithm is applied and the centres are clustered. The final centres and the input data set for the network are displayed in Figure (4.12).

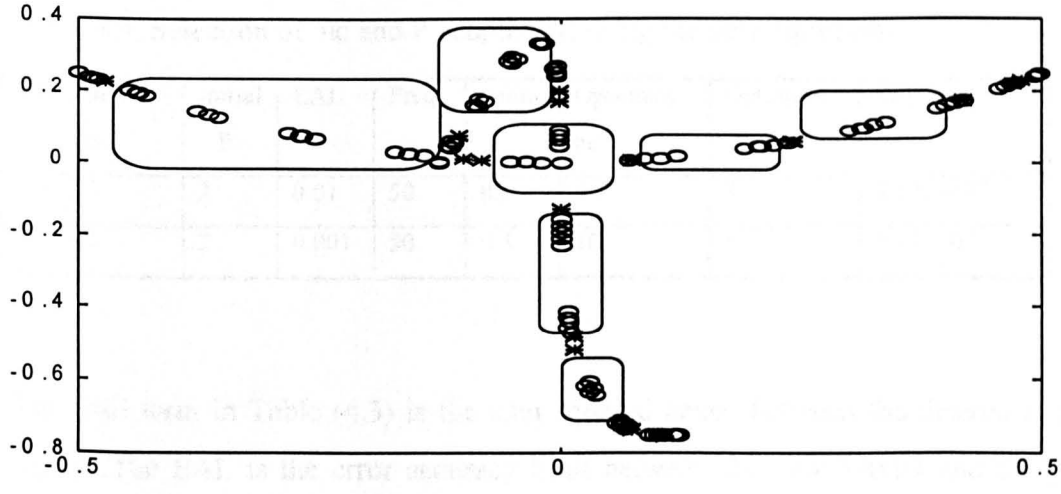


Figure 4.11 , The input data to the RBF and plant (o)
and the initial centres (*)

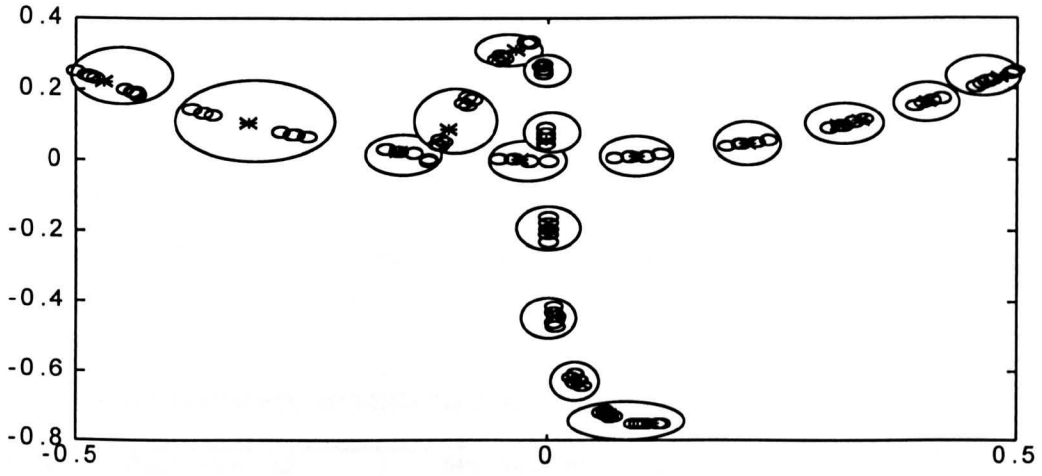


Figure 4.12, The input data to the RBF and plant (small circles), the final centres (*) and groups of clustered data (large circles)

The results are shown in Table (4.3), and the RBF output and the model output are plotted in Figure (4.13). Note that the parameters in the second row have been used for this simulation and the results have been recorded after the simulation run.

Table 4.3, Selection of nc and P adaptively, using the new algorithm

NS	Initial nc	Initial P	EAL	Freq.	Amp.	Optimum nc	Optimum P	SSE
60	4	2	0.01	50	0.5	9	5	3.197×10^{-4}
60	4	2	0.001	50	0.5	10	5	1.15×10^{-4}

The SSE term in Table (4.3) is the sum squared errors between the desired and actual signal. The EAL is the error accuracy limit between the new SAGO and the previous SAGO, which is used for fine tuning the selection of the net parameters. As shown in Table (4.3), the EAL is reduced to 0.001, the index P has been kept the same, but the number of centres is increased to 10 and the SSE decreased by approximately a factor of 3 compared to the first row result. Finally, the error between the plant and the net outputs

is shown in Figure (4.14), where the error level is approximately 0.003, that is 0.857 % of the plant input amplitude. Figure (4.15) shows the comparison between the errors in cases II and III. This Figure shows clearly that the method in case III is significantly better than the other two methods, the error in case III is less than that in case II by a factor 16.7. The percentage error for the three cases is shown in Table (4.4).

Table 4.4, Comparison between the percentage error for I,II and III cases

Case	Plant amplitude	Maximum error amplitude	% error
I	0.35	0.48	137.14286
II	0.35	0.05	14.285714
III	0.35	0.003	0.8571428

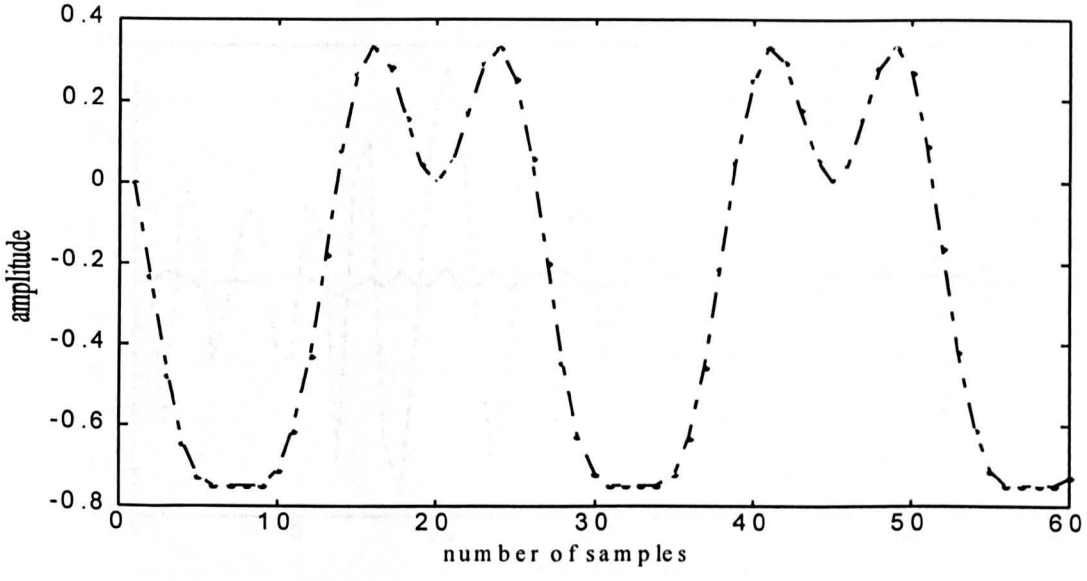


Figure 4.13, modelling the plant using the new algorithm:
Plant output (dash-dot line) and RBFoutput (dotted line)

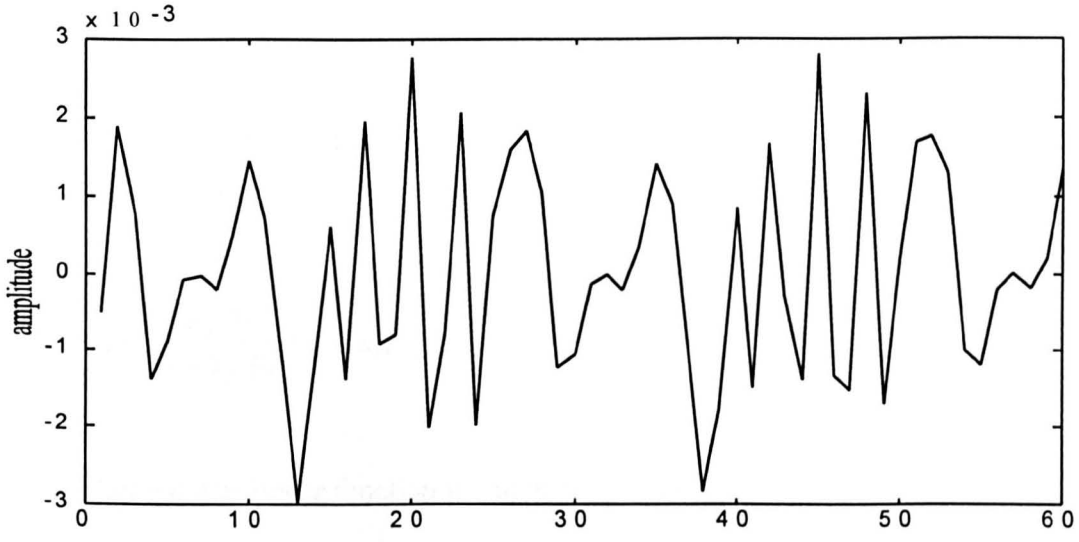


Figure 4.14, The error between the RBF and plant using the new algorithm

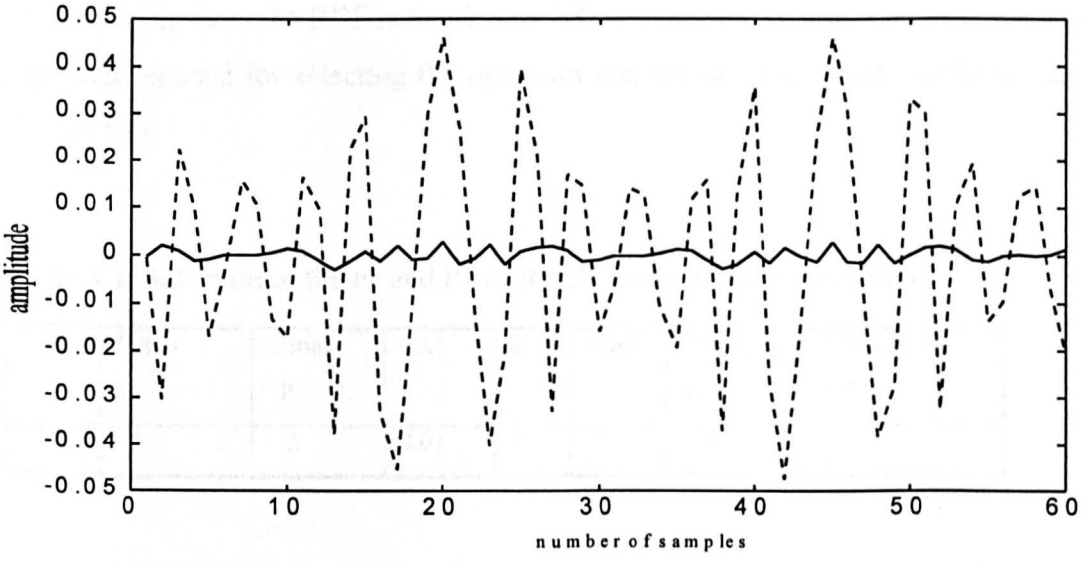


Figure 4.15, Comparison between errors between the RBF and the plant outputs; case II (dotted line) and case III (solid line)

Example (2)

The plant to be identified is a second order non-linear system governed by the following difference equation:

$$y_p(t+1) = \frac{y_p(t)}{1+y_p(t)} + f\,[\,u(t)\,] \tag{4.13}$$

The unknown non-linear function in the plant is assumed to be

$$f[\,u(t)\,] = u^3 \tag{4.14}$$

The input to this plant is a sum of two sinusoids $u(t) = \sin(2\pi t/25) + \sin(2\pi t/10)$. The same procedure explained in example 1 has been used. During the identification process a series-parallel model was used, where the training signal has 500 samples, the number of input samples to the RBF network was 200 and the step size was set to unity. The new algorithm is used for selecting the optimum number of centres and widths as shown in Table (4.5).

Table 4.5, Selection of the nc and P adaptively using the new algorithm

NS	Initial nc	Initial P	EAL	st	Amp.	Optimum nc	Optimum P
200	5	2	0.01	1	7	9	3

The input data to the network and the initial centres are displayed in Figure (4.16). Figure (4.17) shows the final clustering of the centers and the input data to the network. The outputs of the plant and the model are shown in Figure (4.18), and are seen to be

indistinguishable. Figure (4.19) shows the error between the outputs of the plant and the RBF network .

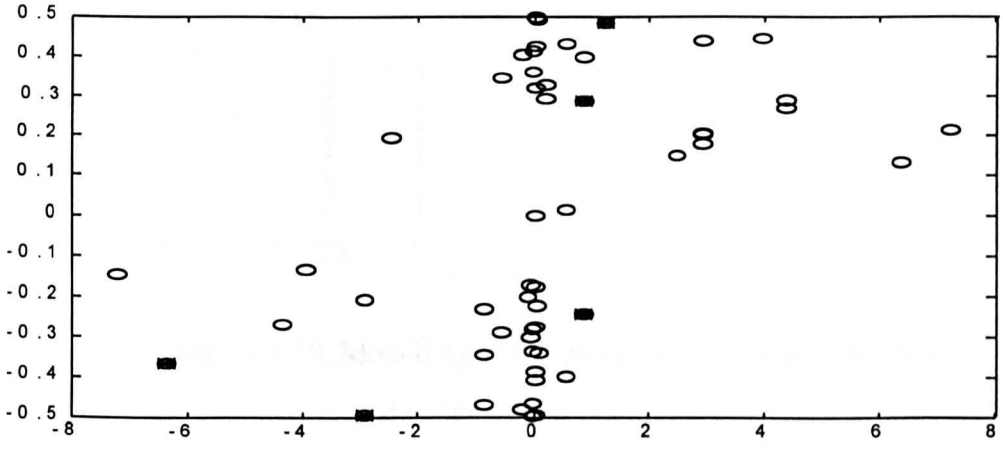


Figure 4.16 , The input data to the RBF and plant (o)
and the initial centres (*)

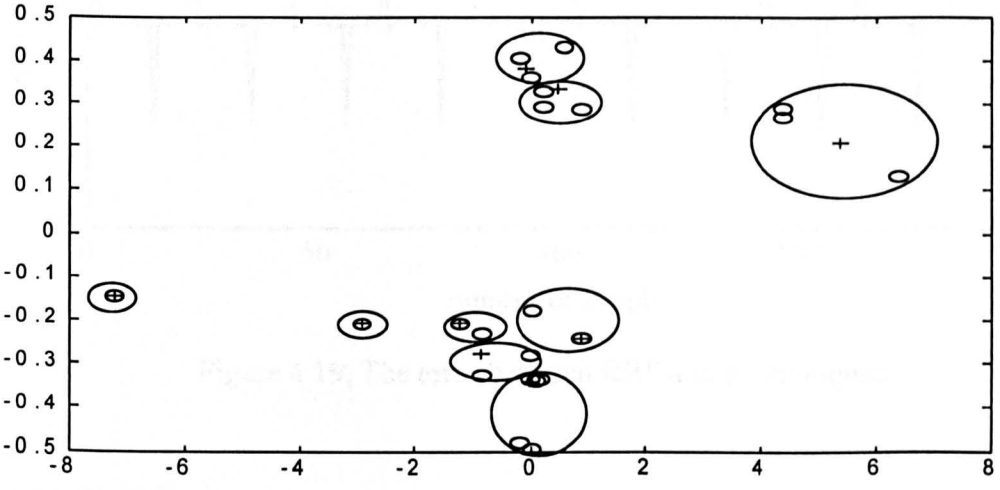


Figure 4.17, The input data to the RBF and plant (small circles),
the final centres (+) and groups of clustered data (large circles)

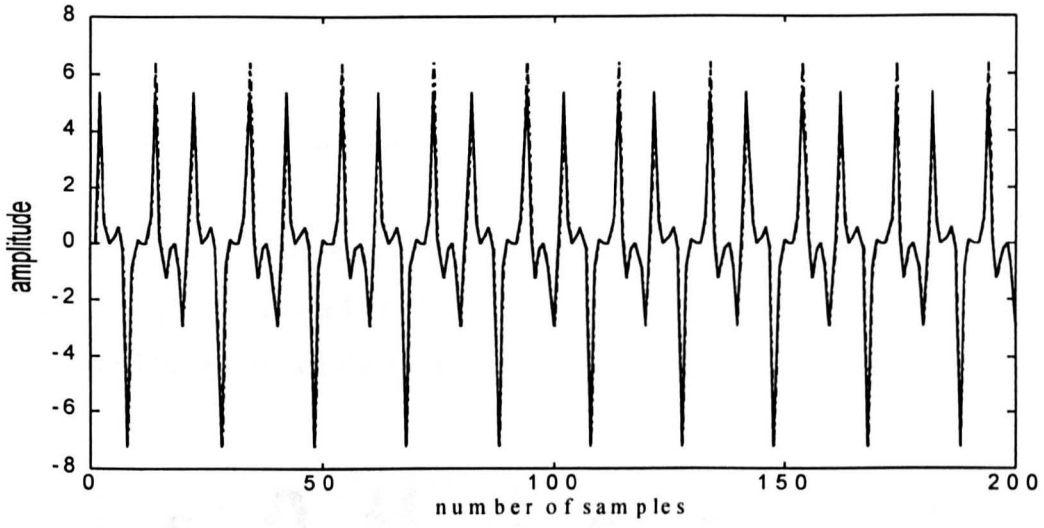


Figure 4.18, Modelling of the plant; desired output (dotted line) and actual output (solid line)

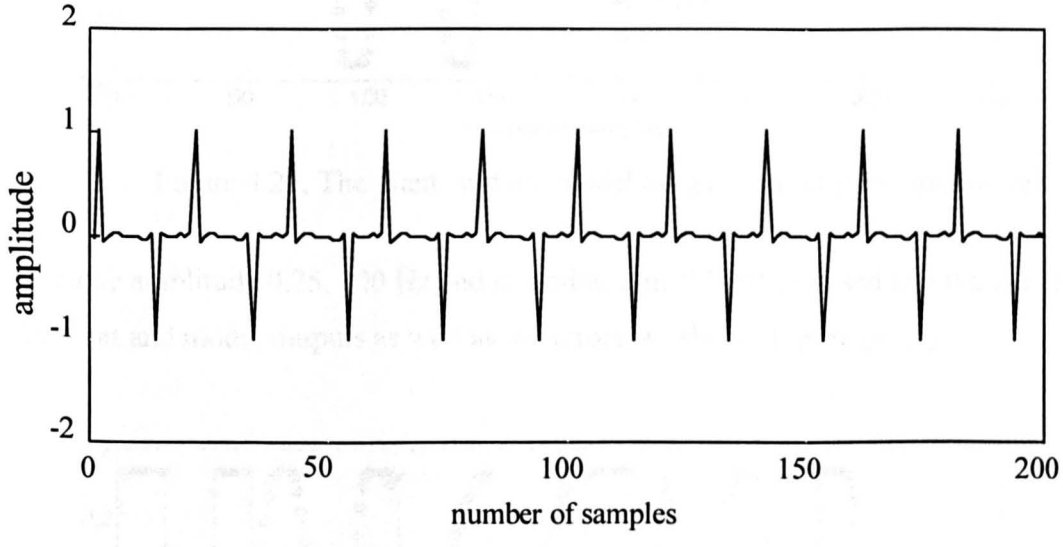


Figure 4.19, The error between RBF and plant outputs

4.1.3 Model test

Once the plant is modelled, it is convenient to apply some different signals to insure that the obtained model is the same as the plant itself i.e. the selected parameters are correct.

In this test, the non-linear plant (4.12) and the obtained model discussed in example (1) section (4.1.2) are considered. The model is tested and the results are compared with the plant output using Matlab/Simulink packages as shown in Figure (4.5) in the previous section. The parameters in second row of Table (4.3) have been used for this test. Different signals with different amplitudes are used and the results are shown. Firstly, the sinusoidal signal is applied and the input amplitude is varied. The plant and the model outputs and the errors are shown in Figure (4.20). Then the different input signals with

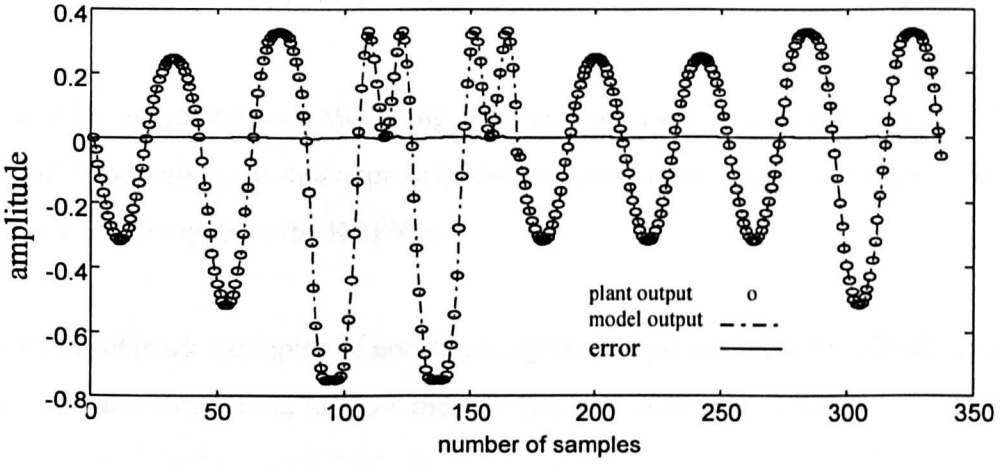


Figure 4.20, The plant and the model outputs (varieng the amplitude)

the same amplitude 0.25, 100 Hz and sampling time 0.0001 are used and the results of the plant and model outputs as well as the errors are shown Figure (4.21).

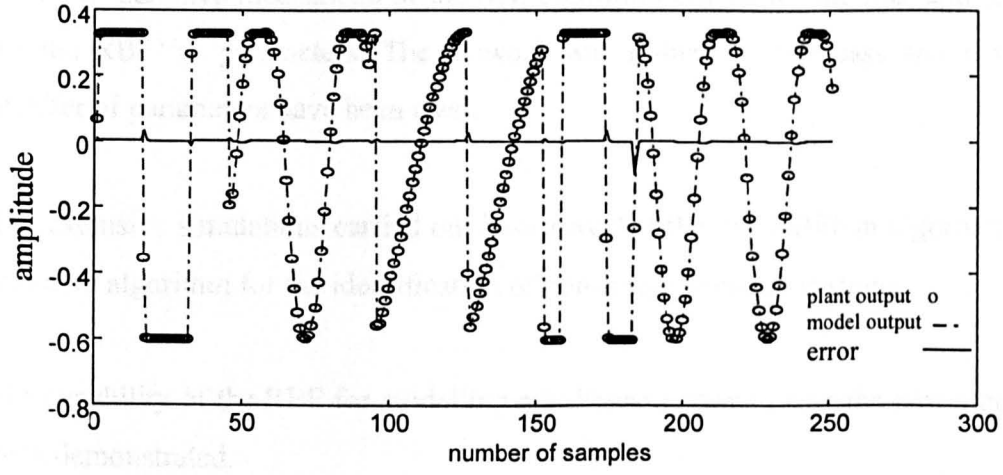


Figure 4.21, The plant and the model outputs (varieng the input form)

As shown in the above Figures, during the change of the signal form and amplitude the desired input tracked the actual signal very well. This indicates that the model and the plant are in agreement.

4.1.4 Conclusion

In this chapter the use of artificial neural networks for the identification of non-linear dynamic models was described and evaluated.

The RBF neural network modelling structures were considered, these included the series-parallel modelling. In this approach, the delayed values of relevant signals in the system were used as inputs to the RBFNm.

Two benchmark examples of non-linear dynamic systems were identified, using RBFNm in computer simulation, to show the effectiveness of the algorithms.

Different approaches for selecting the RBFNm parameters were tested and the benefit of using *"the proposed new method"* has been illustrated. It has been proved by simulation that the selection of optimum values of the network parameters is possible using the new algorithm. The results of the various methods have been summarised in Table (4.4). Thus, the adaptive mechanism of the new algorithm eliminates the trial and error search for the RBFNm parameters. The network was trained in one pass and a very small number of parameters have been used.

The extensive simulations carried out have revealed that the RBFNm algorithm is a very effective algorithm for the identification of non-linear dynamic systems.

The capability of the RBF for modelling non-linear systems using the new algorithm has been demonstrated.

CHAPTER 5

NON-LINEAR DYNAMIC SYSTEMS CONTROL USING RADIAL BASIS FUNCTION

CHAPTER 5

NON-LINEAR DYNAMIC SYSTEMS CONTROL USING RADIAL BASIS FUNCTION

5.0 INTRODUCTION

In this chapter, artificial neural networks (ANN's) are considered for the purpose of controlling non-linear dynamic systems. ANN's are adaptive systems capable of overcoming many of the difficulties encountered by conventional adaptive control techniques, specially with non-linear plants with unknown structures [32,85,86]. Some types of neural networks have been discussed in the previous chapters, each has a problem. The radial basis function is suggested to overcome these problems and is thought to be the best. To the best of author's knowledge this type of neural network has not been widely investigated or used for the control of non-linear dynamic systems [44]. In general, neural network controllers are based on a structure which learns the inverse dynamics of the plant from the observation of the plant ***inputs-outputs*** relationship through time. The inverse model of an unknown plant may be used to generate a command signal to the plant itself, to make it operate in a desired manner. Various types of learning algorithms for inverse control have been suggested in the literature [67,86]. In this study two methods are investigated: the general learning method and the adaptive control method which are presented in Sections (5.1) and (5.2), respectively. Simulation results are presented and evaluated in Section (5.3). Finally, the conclusions are drawn in Section (5.4).

5.1 General Learning Method.

The general learning method, which is also known as *off-line* inverse modelling or direct inverse control, is depicted in Figure (5.1).

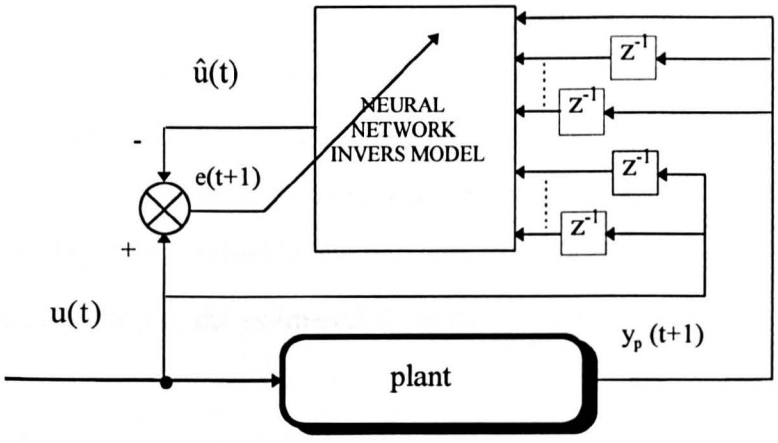


Figure 5.1, General learning diagram

The training structure contains a trained forward model of the RBF neural network placed after the plant. As a special case, for the radial basis function network controller RBFNc, the centres and the width parameters are assumed to have already been selected, as described in section (4.1.1).

From equation (4.3), the non-linear relationship for the network model, the plant inverse is given by:

$$u(t) = \overset{-1}{f}[y(t+1), y(t), .., y(t-n+1), u(t-1), ..., u(t-m+1)] \tag{5.1}$$

where $\overset{-1}{f}$ is the inverse of non-linear function $f(.)$.

The utilised controller network RBFNc can be described by

$$u^C(t) = \sum_{i=1}^{NC} w_i^C . \phi_i^C \tag{5.2}$$

where,

$$\phi_i^c = exp^{-(x^c - c_i)/\sigma_i^2} \tag{5.3}$$

In equation (5.2), the c superscript indicates a variable related to the controller and (w^c) is the connection control weights between the hidden layer and the output layer; these weights are adjusted when the known input (u) is used as the input to the plant to obtain a corresponding plant output (y). Alternatively the plant outputs y and the desired signal u (with their lags) are applied to the network to produce the control signal $\hat{u}_c$. Then the difference equation of the estimated $\hat{u}_c$ is written in the form

$$\hat{u}_c(t) = \overset{-1}{\hat{f}}[y(t+1), y(t), \ldots, y(t-n+1), u(t-1), \ldots, u(t-m+1)] \tag{5.4}$$

where, $\overset{-1}{\hat{f}}$ represents inverse of the system realised by the RBFNc. Thus, for equation (5.4) the neural based prediction control signal can be realised as

$$\hat{u}_c(t) = RBFNc\,[x^c(t)] \tag{5.5}$$

where

$$x^c(t) = [y(t+1), y(t), \ldots, y(t-n+1), u(t-1), \ldots, u(t-m+1)] \tag{5.6}$$

The learning process of the neural network is carried out to minimise the overall square of errors, between the desired output (u) and the actual output $(\hat{u}_c)$, until the error becomes very small (i.e., the values of u equal or nearly equal to that of $\hat{u}_c$). Then the weights are adjusted and the network should be able to take any reference signal y_r (not necessarily the training signal) and produce an appropriate output u_c, which makes the actual plant output y approximately equal to y_r [74,75,85].

Unfortunately, this way of controlling the plant output y is ineffective due to the high level of error; that is the estimated plant output y and the reference signal y_r have a large difference. The reason for this is because we cannot selectively train the plant to respond

correctly to the region of interest since we do not know which plant input corresponds to the desired output. Therefore, this method is rarely used in practice since the learning process of the neural network is carried out *off-line*. An alternative solution to this problem is to train the network using an *on-line* learning approach.

5.2 Direct Adaptive Control Method

The direct adaptive control (or the on-line) method is used to overcome the problems associated with the off-line approach. In this method, the neural network learns during the *on-line* feed forward control period. As shown in Figure (5.2), the controller network is placed in front of the plant, whereby the net output control signal $\hat{u}_c$ is an input to the plant.

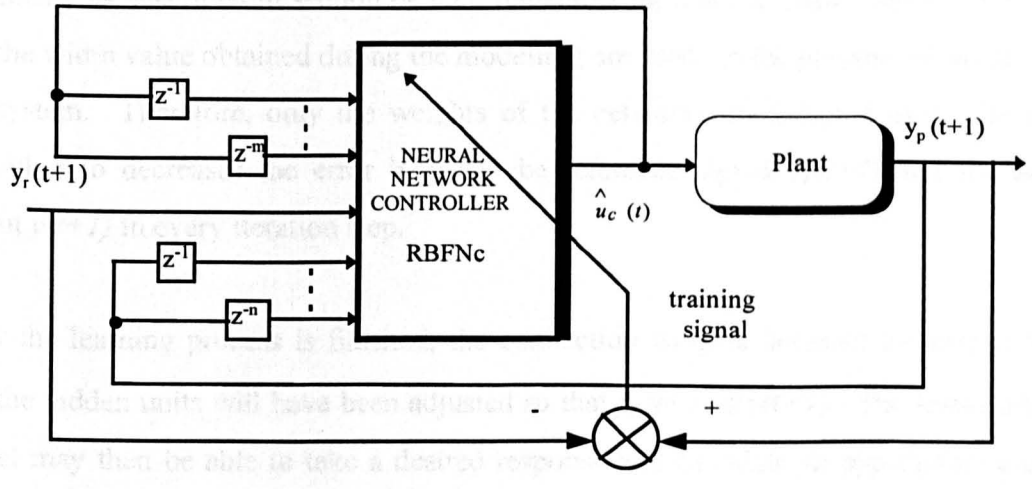


Figure 5.2, Direct adaptive control configuration

The network is trained to produce the plant input that drives the system output to the reference values $y_r(t+1)$. Thus, the input to the network can be obtained from equation (5.6), where the desired reference signal $y_r(t+1)$ was used instead of the unknown $y(t+1)$, this can be rewritten as

$$x^c(t) = [y_r(t+1), y(t), ..., y(t-n+1), u(t-1), ..., u(t-m+1)]^T \tag{5.7}$$

In general it is important that y_r is chosen such that it can be physically achieved by the system, so that the error can truly reach a minimum.

During the training of the network, the centres are assumed selected presumed to have been prior to training and are given as

$$C_{ji} = [c_{11} \quad c_{12} \quad . \quad . \quad . \quad c_{1n}]^T \qquad \begin{matrix} i = 1,2,...,n \\ j = 1,2,...,m \end{matrix} \tag{5.8}$$

where n is the network input space dimension and m is the number of centres (hidden units) with $m \leq n$. It is recommend that the width between the centres is also selected previously as described in section (4.1.1), remembering that the same number of centres and the width value obtained during the modelling are used for the purpose of controlling the system. Therefore, only the weights of the networks are adapted using the LMS algorithm to decreases the error between the reference signal $y_r(t+1)$ and the actual output $y(t+1)$ in every iteration step.

After the learning process is finished, the connection weights between the output layer and the hidden units will have been adjusted so that $y_r(t+1) \cong y(t+1)$. The learnt inverse model may then be able to take a desired response and calculate an appropriate control signal $\hat{u}_c$, forcing the plant output to approach the desired value. Thus, equation (5.7) can be rewritten as

$$x^c(t) = [y_r(t+1), y_r(t), ..., y_r(t-n+1), u(t-1), ..., u(t-m+1)]^T \tag{5.9}$$

The fully learnt RBF network controller is depicted in Figure (5.3). It can be seen from this figure that no adjustment of weights is needed when the desired reference is within the boundary of the training signal.

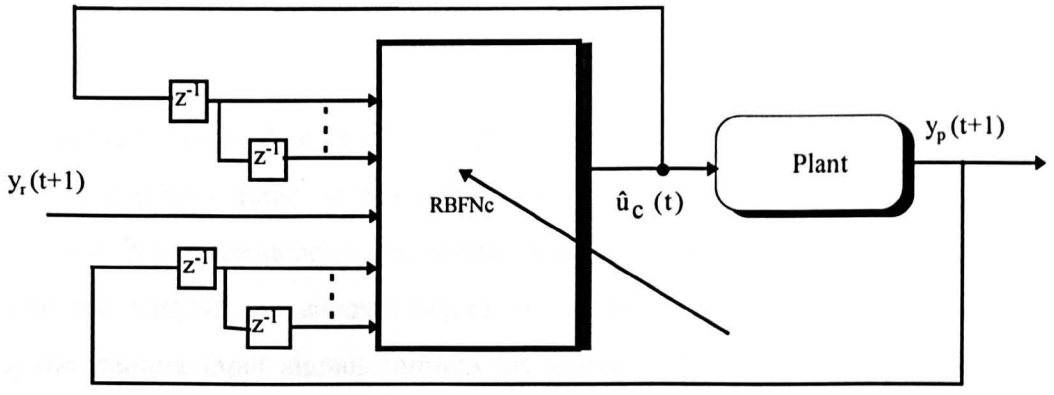


Figure 5.3, Radial basis function controller (RBFNc) configuration

The direct adaptive control method is considered in our work for controlling the non-linear dynamic models due to its simplicity and robustness, especially when the proper parameters are selected. One limitation which might affect the performance of the controller is the choice of the learning factor (η) which controls the convergence of the LMS algorithm. Care must be taken by the user when selecting this value; the correct value can be selected according to the experience or by a trial and error procedure.

5.3 Simulation Results

The performance of the neural network controller is illustrated by the two examples discussed in Section (4.1.2). These examples have been studied by Narendra [66], Hunt and Sbarrbaro [44], where different controller methods and neural network types have been employed.

Regardless of the method being employed, the objective for any control system is to force a given dynamic system to exhibit a desired response. In this study the RBFNc is chosen as the reference controller against which the accuracy and performance of the other neural network controllers may be compared.

Example (1):

A first order non-linear plant (4.12), discussed in section (4.1.2), is considered. In this example, the desired outputs were selected as a square wave, sinusoidal and saw tooth signals with different frequencies and amplitudes. The centres and the width between the centres of the RBFNc are selected before; for more details see previous section. By passing the training input signals through the selected 10 RBFNc nodes (centres), the network controller is configured in the same way as shown in Figure (5.2). Assuming the optimal width has been selected, then the optimal output layer weights are found using the least mean square LMS algorithm *equation (2.34)*. The weight vectors is initially set to zero and the learning factor is varied depending on the behaviour of the desired input. The simulations are carried out using the Matlab/Simulink packages described in the previous section. Thus, the designer can use the block diagram windows to create a model and achieve the desired behaviour. The other important advantage of using Simulink is the simplicity in building and modifying the models. Simulink also enables us to view the progress of a simulation during its run.

The Simulink model of the RBFNc design is shown in Figure (5.4). This is a hypothetical example, where only two inputs with one centre have been shown to illustrate the mechanism.

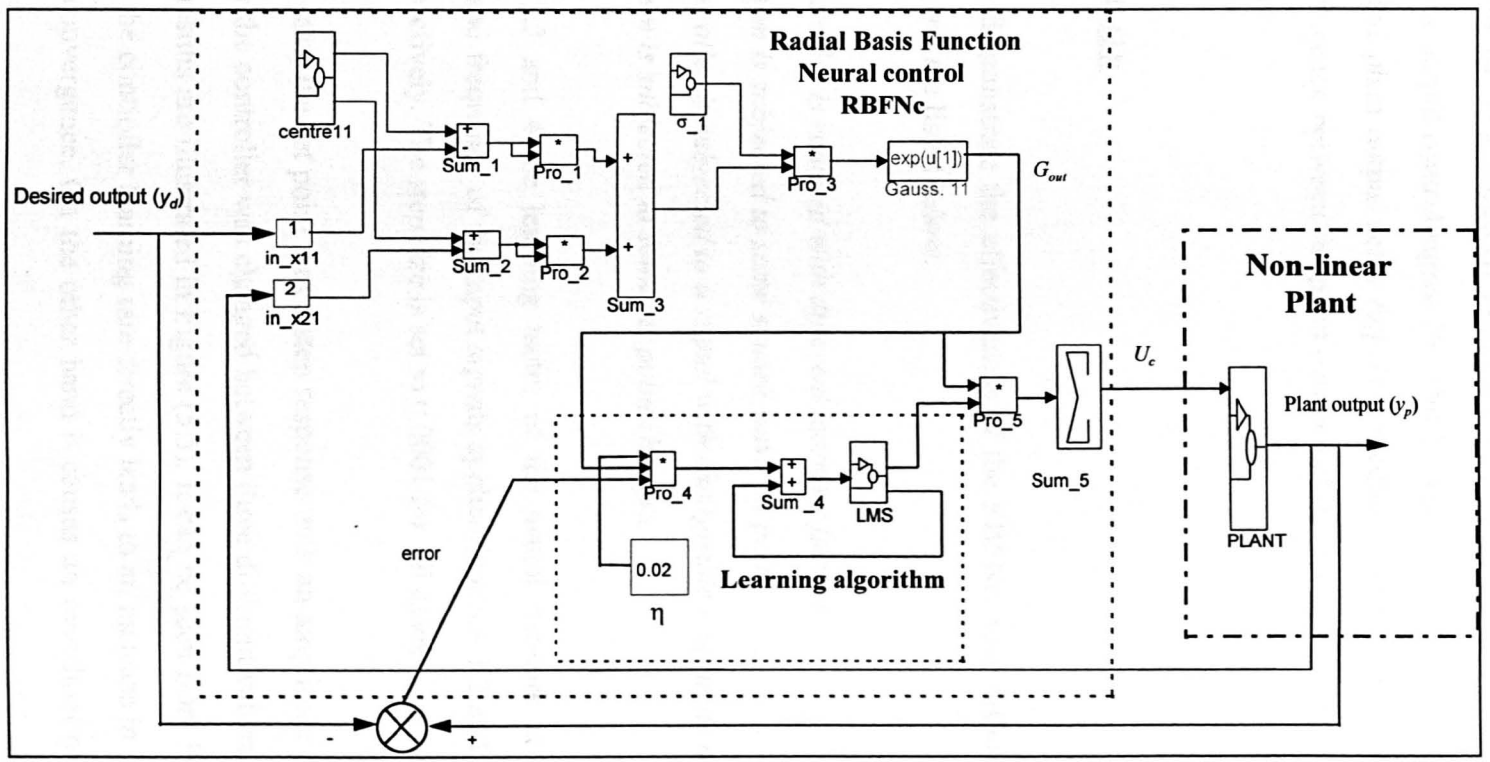


Figure 5.4, Simulink blocks setup to simulate the control of a non-linear plant sing RBFNc

In this example, the designed network consists of one centre (i.e., only one RBF unit). The RBF is loaded with its own centres and width values. The desired and feedback input signals are passed via the RBF unit and the Gaussian output (G_{out}) is calculated. The LMS on-line method is used for adapting the RBF controller weights. Thus, the weighted Gaussian outputs are summed (if more than one output) by the output layer to produce the RBF network output control signal *Uc*. The *Uc* signal drives the plant to give the desired signal y_d. The plant output delay $(y_d - 1)$ is feedback to the network and the error signal (i.e., the difference between the plant output and the actual output) is used for correcting the weights.

5.3.1 Result (1):

In order to demonstrate the effectiveness of the RBFNc, four different tests have been carried out and are listed below.

1- The controller is realised with different learning factors.
2- The system is subjected to some square wave set point.
3- The controller is subjected to a signal with changeable amplitude and frequency.
4- The system is subjected to some set point changes.

For cases 2,3 and 4 the learning factor of the neural network is fixed at 0.05. The amplitude and frequency of the input signals applied in cases 1,2 and 4 are set to 0.3 and 50 Hz, respectively. The step size is set to 0.0001 for all cases.

In the first case, the set point was a step response with an amplitude of 0.3. The learning factor η for the controller was changed between three different values, η = 0.01, 0.05 and 0.099, the results are illustrated in Figure (5.5). It can be seen from Figure (5.5) that any increase in the controller learning rate directly leads to an increase in the desired response and faster convergence. On the other hand it causes an overshoot on the control signal

itself. However, a small value for η cause a slow convergence rate. Therefore, care should be taken when choosing the learning rate values.

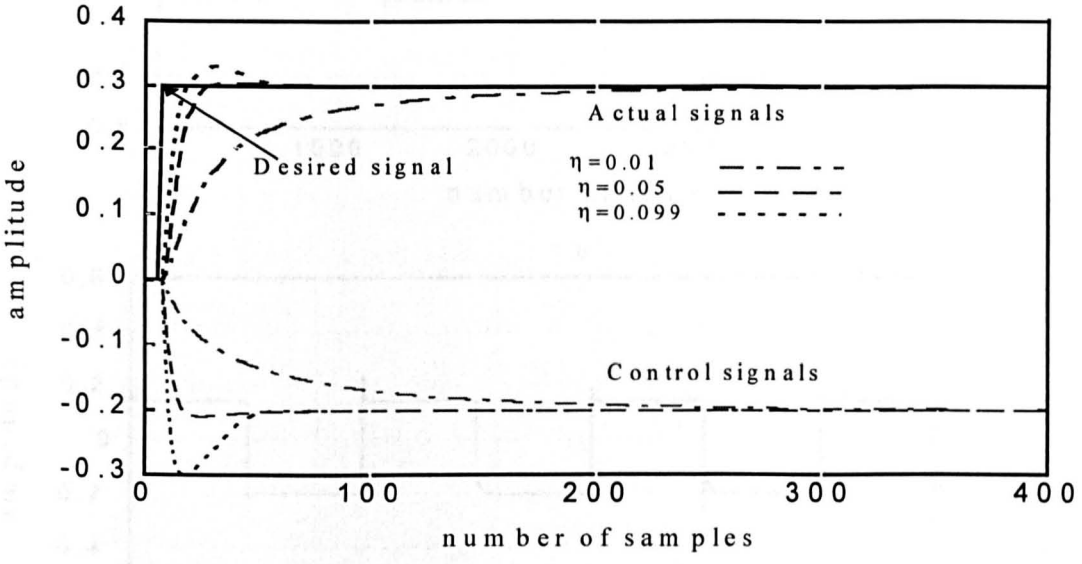


Figure 5.5, The effectiveness of changing Learning rate values

In the second test, the learning rate of the controller was set to 0.05 and the set point applied to the system was a square wave. The control response is illustrated in Figure (5.6), where, the number of samples was 5000 ,the desired signal was a square wave with an amplitude of 0.3. The actual and desired signals are shown in Figure (5.6a) and the control signal and the error between the signals are shown in Figures (5.6b, 5.6c), respectively.

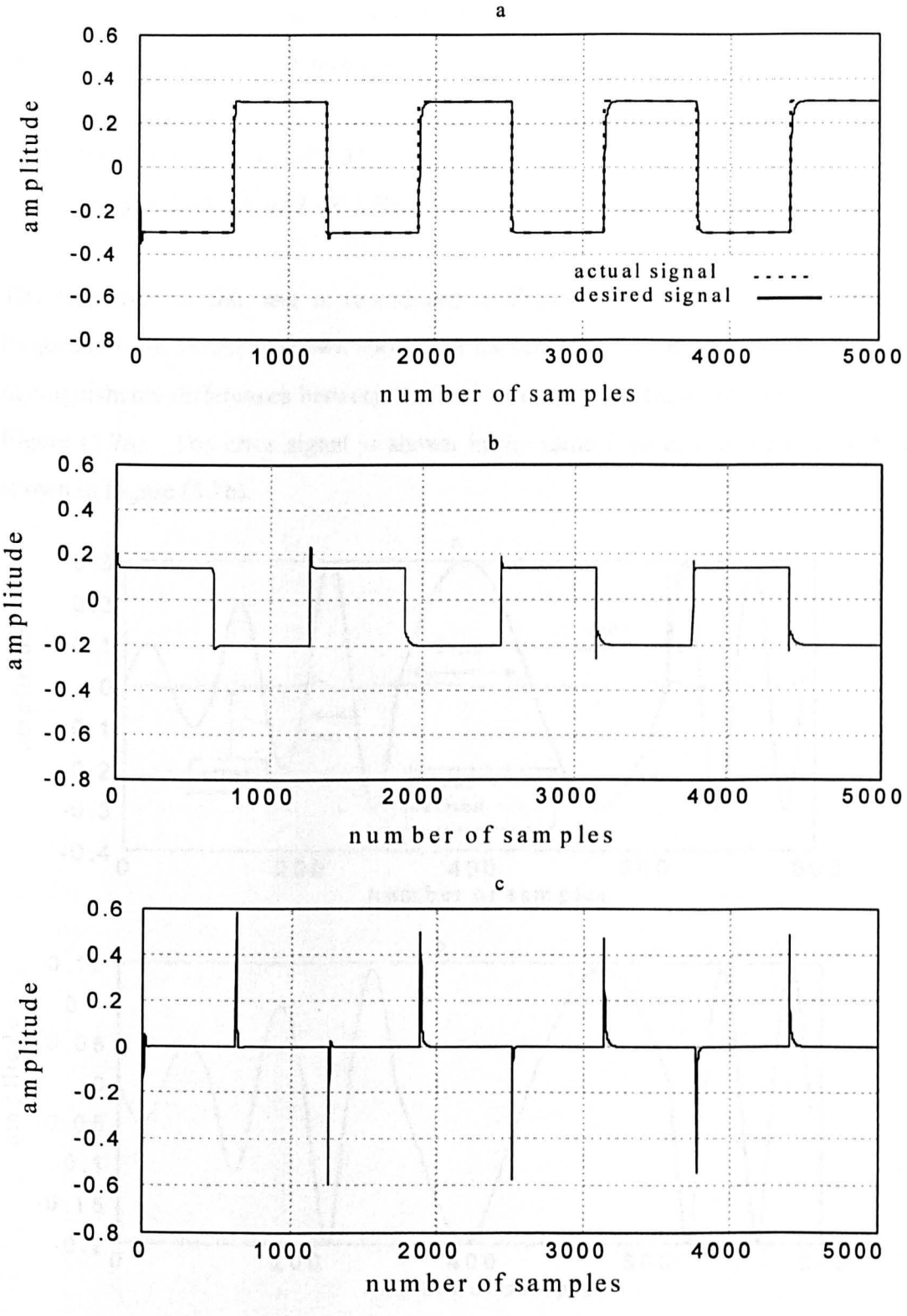


Figure 5.6, Results of square wave input(a), desired and actual signals, (b), control signal and (c), the error signals

In the third test, the learning factor is set as in case 2 and the amplitude and frequency of the sinusoidal signal were changed as follows:

- *amplitude = {0.1, 0.2 and 0.3}*
- *frequency = { 60, 20 and 80 } Hz .*

The responses of this test is reproduced in Figure (5.7). The chosen amplitude and frequency were varied as shown above and the number of samples was 800. There are no distinguishable differences between the desired output and the actual output as shown in Figure (5.7a). The error signal is shown in the same Figure, and the control signal is shown in Figure (5.7b).

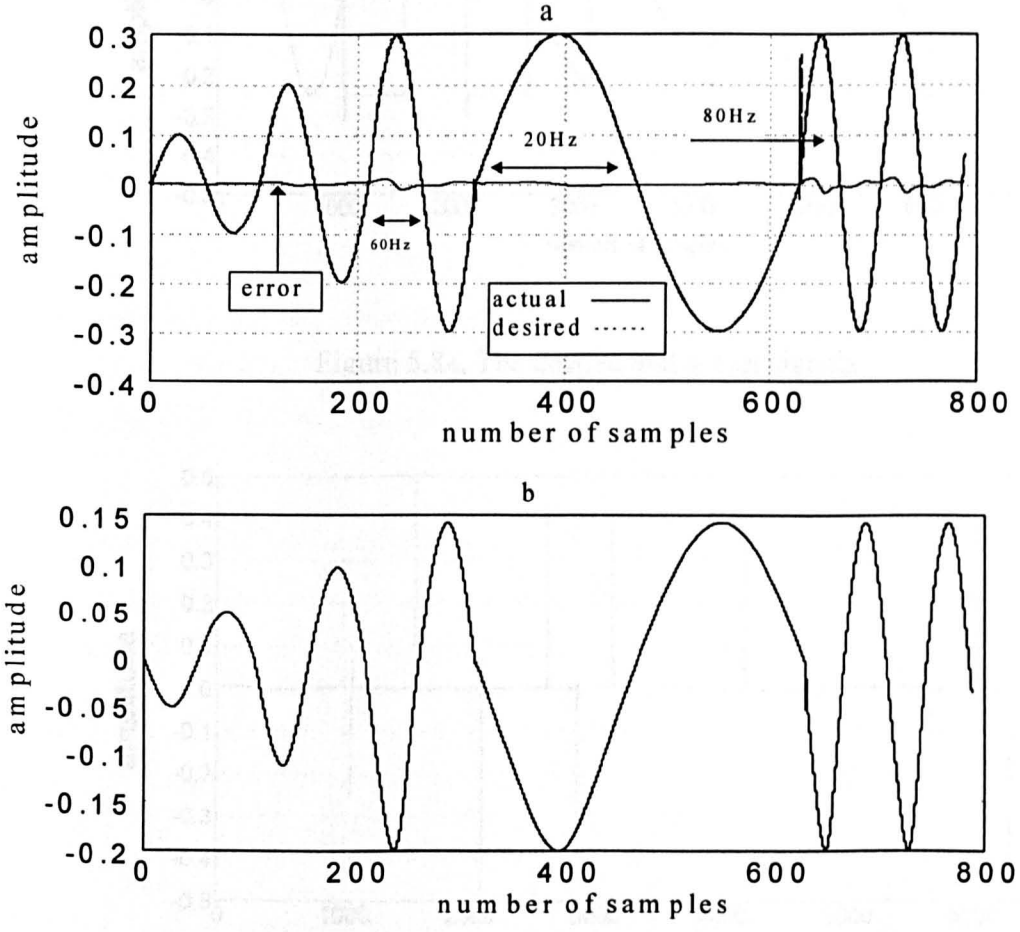


Figure 5.7, Varieng the amplitude and frequencies of the sinusoidal input signal, (a), the desired and actual signals and the error between them and (b), the control signal

In the last case, different shapes of set points were applied to the system and the rest of the parameters are kept as in the case 2. The results are shown in Figure (5.8). At the beginning of the simulation, the sinusoidal was applied followed by square wave, sow wave, sinusoidal and square wave signals. There is no differences between the desired and actual signals as shown in Figure (5.8a). The error signal and the control signals are depicted in Figures (5.8b) and (5.8c) respectively.

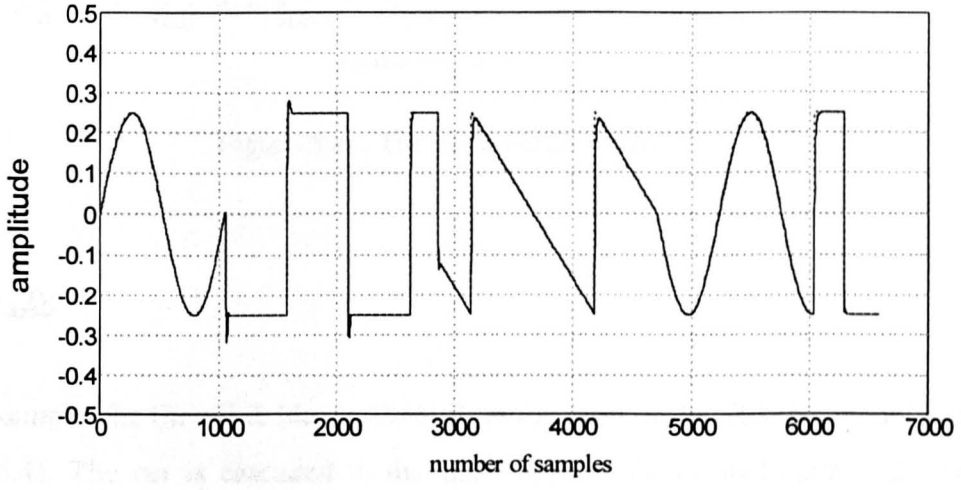


Figure 5.8a, The desired and actual signals

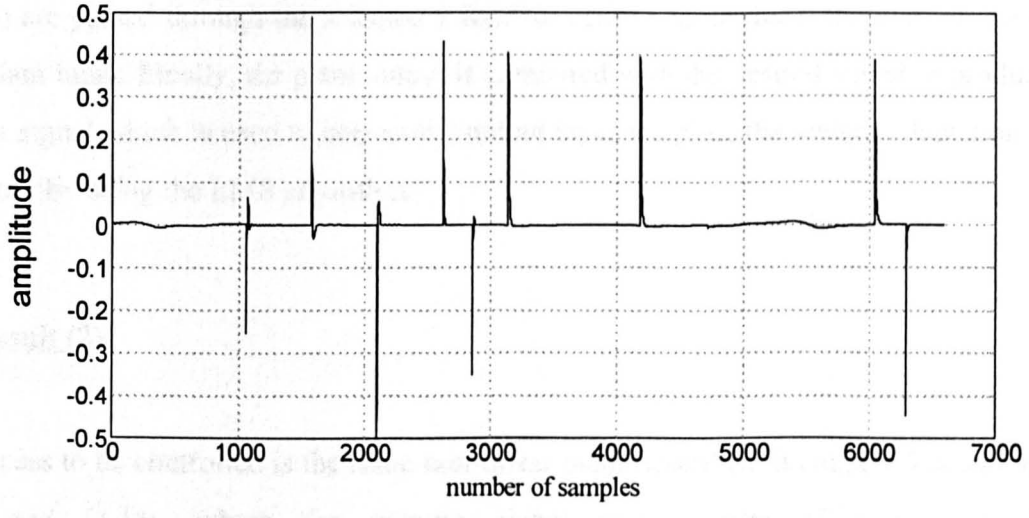


Figure 5.8b, The error between the signals

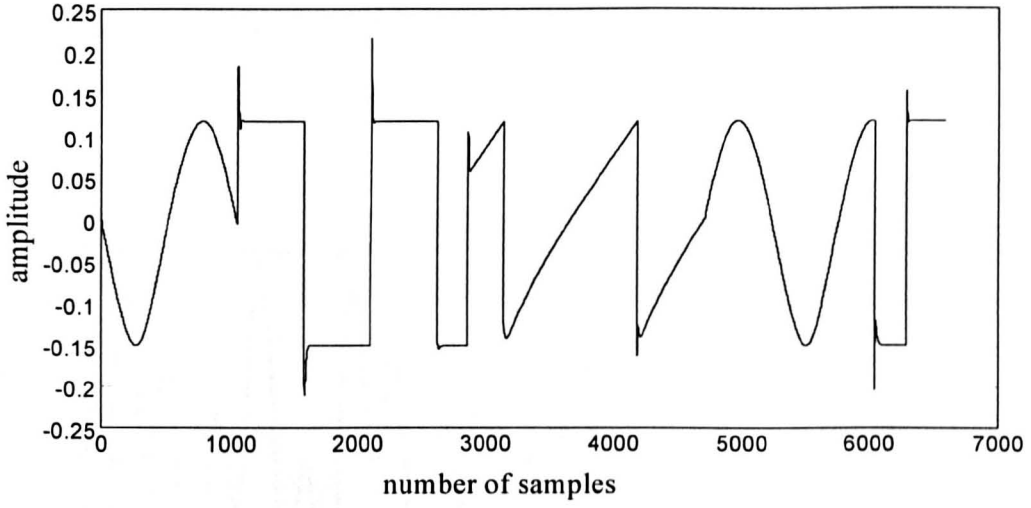


Figure 5.8c, The plant control signal

Example (2):

In this example, the Simulink blocks are built in the form of the RBFNc model shown in Figure (5.4). The net is cascaded in the same way as shown in Figure (5.2), and the second order plant is modelled as discussed in the previous chapter, section (4.1.2) and example (2). The important parameters (centres and widths) were also selected during the identification process. The desired input signal together with the plant output (including the lags) are passed through the selected 9 RBFNc centres to produce the control signal as the plant input. Finally, the plant output is compared with the desired signal to produce the error signal which is used to correct the output layer weights; the weight adaptation is carried out by using the LMS algorithm.

5.3.2 Result (2):

The process to be controlled is the same non-linear plant described in chapter 4 *equations (4.13) and (4.14)*, where the training signal was a sum of two sinusoids

$u(t) = \sin(2\pi t / 25) + \sin(2\pi t / 10)$. The input to the net was 200 samples as shown in Figure (5.9).

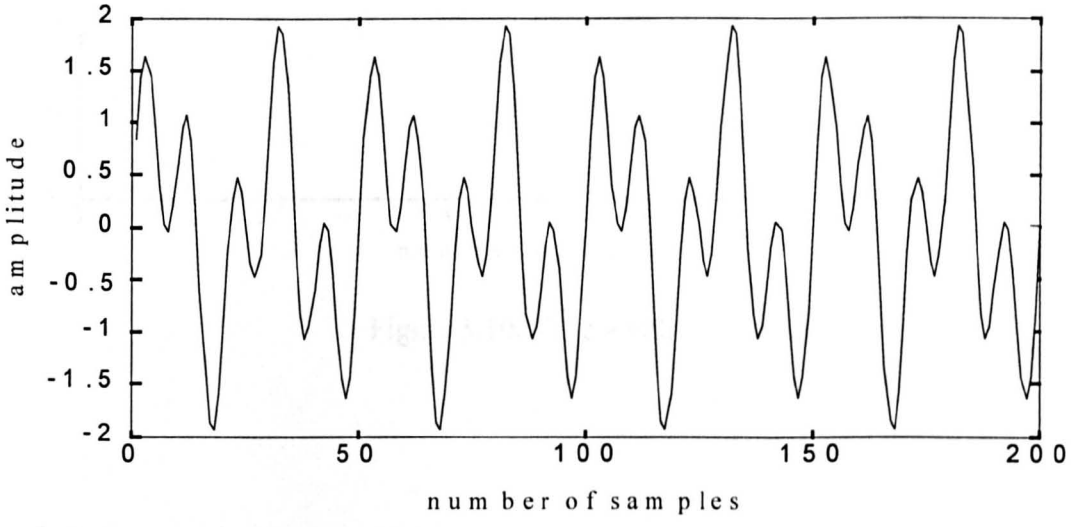


Figure 5.9, The training signal of example two

As discussed in chapter 4 example (2), the new algorithm has been used to select the optimum centres, widths and weights when 200 samples are considered as the input to the feedforward controller and the plant. The same control objective given in example (1) was also considered in this example. The purpose of this example is to discuss and explain some points not covered in the previous example. These points can be outlined in the three cases discussed below. In the first test, the random centres shown in chapter 4, Figure (4.16), and the constant value 0.1 of the centres width shown in Figure (5.10a), are examined to show the effectiveness of the non optimal parameters. The simulation time was 4 seconds and the amplitude of the input signal was set at 0.5. The response is depicted in Figure (5.10b), in which it is shown that, the desired sinusoidal output has not been tracked.

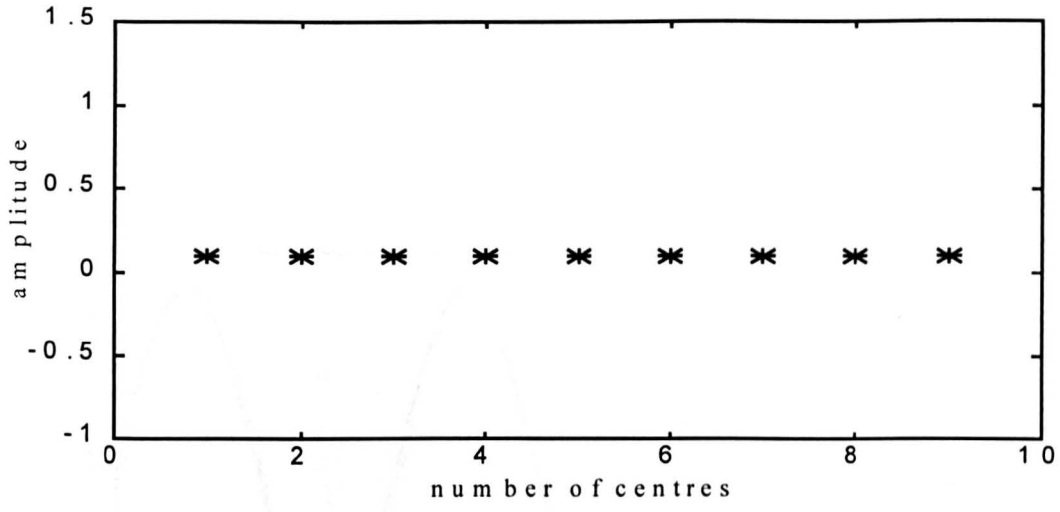


Figure 5.10a, The width for test 1

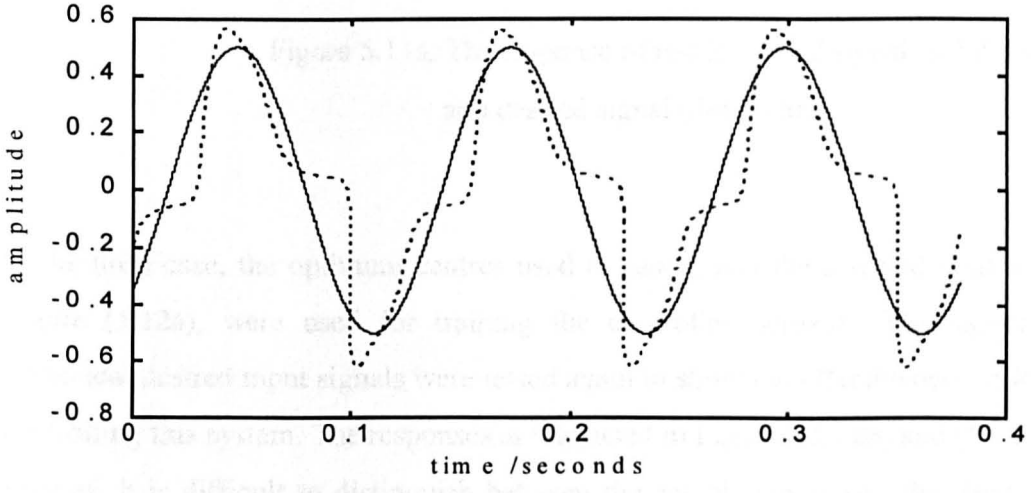


Figure 5.10b, The response of case 1, actual values (solid line), approximated values (dotted line)

In the second test, the optimum centres shown in Figure (4.17) together with the spread of case 1 are used. The same running time, amplitude and desired signal applied in test 1 are used. The result of this case is shown in Figure (5.11a), where, it is seen that the actual

output has improved, compared to the results of case 1, but does not as yet reach the optimum value.

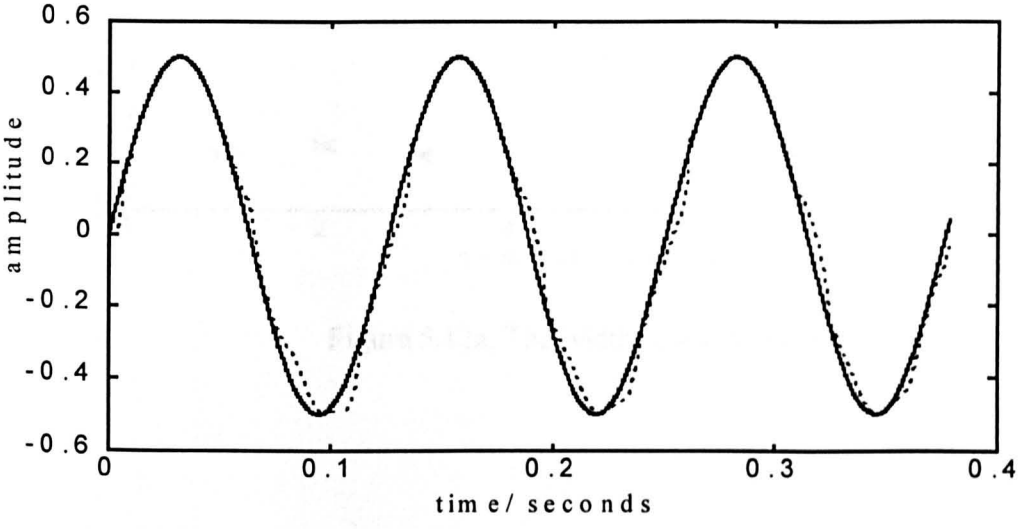


Figure 5.11a, The response of test 2: Actual signal (solid line) and desired signal (dotted line)

In the third case, the optimum centres used in case 2 and the selected widths shown in Figure (5.12a), were used for training the controller network. The square and the sinusoidal desired input signals were tested again to show the effectiveness of RBFNc for controlling this system. The responses are depicted in Figures (5.12b) and (5.12c). In both Figures, it is difficult to distinguish between the actual signals and the desired signals. This proves the RBFNc parameters appropriateness.

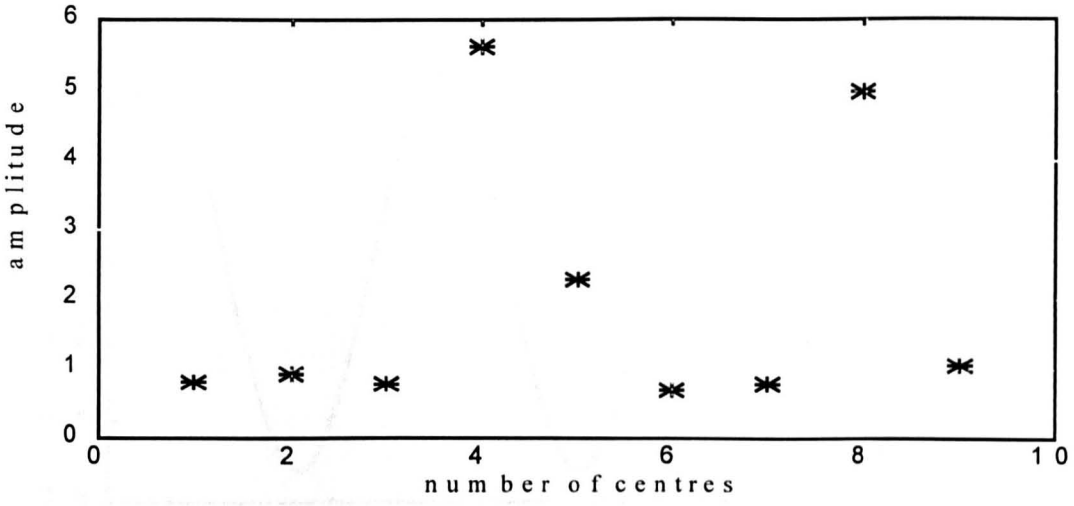


Figure 5.12a, The widths used in test 3

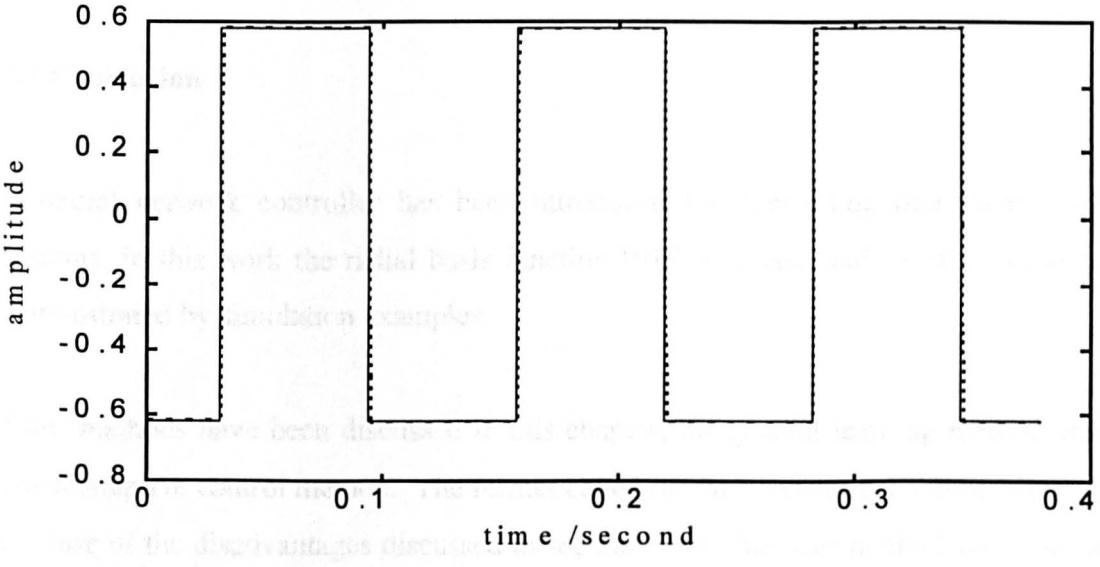


Figure5.12b, Response of square wave: actual signal (solid line)
desired signal (dotted line)

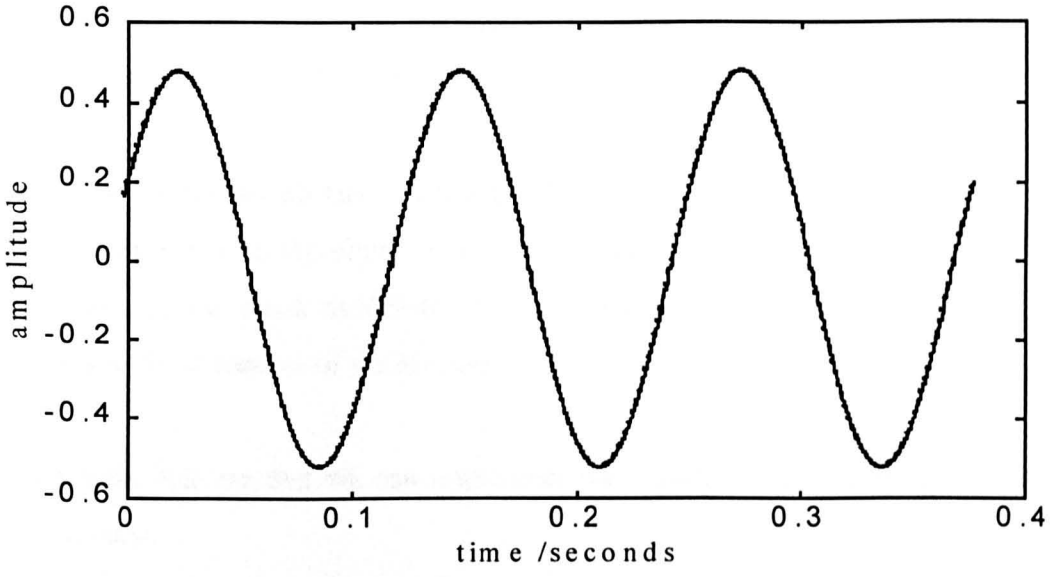


Figure 5.12c, Response to sinusoidal wave: Actual signal (solid line) desired signal (dotted line)

5.4 Conclusion

A neural network controller has been introduced for controlling non-linear dynamic systems. In this work the radial basis function RBF was used and its effectiveness was demonstrated by simulation examples.

Two methods have been discussed in this chapter; the general learning method and the direct adaptive control method. The former called *off-line,* has not been used in our work, because of the disadvantages discussed in section (5.1). The later method was considered and employed. The RBF was used *on-line* to adapt the weights using the *LMS* algorithm to obtain the control signal which can be directly input to the non-linear dynamic system.

The effectiveness of the RBF was demonstrated by examples. Simulation results have shown that the RBF has good performance and ability in controlling dynamic systems. The results have also shown that the parameters selected using the new method were more

appropriate. It can be also shown that the response of the desired signal was heavily dependent on the value of the learning factor chosen.

In conclusion, the results have shown that the on-line RBF neural network controller is an effective and robust algorithm; one of the main advantages is the adaptivity of the algorithm i.e., the quick modification in the behaviour of the controller when there are changes in the dynamics of the process.

The results indicate that we can implement the algorithm in modelling and controlling real systems.

CHAPTER 6

MODELLING AND CONTROL OF A BRUSHLESS DC MOTOR USING RBF NEURAL NETWORK CONTROLLER

CHAPTER 6

MODELLING AND CONTROL OF A BRUSHLESS DC MOTOR USING RBF NEURAL NETWORK CONTROLLER

6.0 INTRODUCTION

Direct drive (DD) motors have received increasing attention for their freedom from backlash and dead zone caused by reduction gearing. These high performance servo motors are used extensively in industrial automation areas such as the car manufacturing. The drive control of such motors is a very important task. However it is very difficult, and in order to obtain a good controller it is advantageous to exploit non-linear control techniques, e.g. neural network. This is because, to some extent, most such systems are non-linear. In motor control system, the non-linearity depends on such factors as fraction hysteresis or saturation [70,81]. These effects are usually neglected in drive control in order to use linear controllers, but this does not give a very good control result.

In order to control a system, the mathematical structure representing a process should be known a *priori*. In reality, the structure of the systems to be controlled is not always known and so, these systems are treated as black boxes. These black box systems are mostly identified by applying various identification methods, using the available system input-output data.

In this chapter a brushless DC motor is considered as a black box system because of the absence of information; such as the mathematical model (model order and time delays). In the first instance the well known ARX model is applied to the real system input-output data. Once the mathematical model is ascertained, the Smith predictor is used for controlling the system and the RBF neural network is then applied for modelling and controlling the system.

The layout of chapter is as follows. The basic brushless DC motor, the hall effects elements, the general dynamic model and the motor transfer function are described in section 6.1. In section 6.2 brief descriptions of the servo system, motor control card, data acquisition and the experiment set-up are given. The identification of the real system using ARX model is given in section 6.3. The Smith Predictor controlling method is briefly discussed and some simulation results are provided in section 6.4. The results of modelling and control of the system using the RBF neural network are given in sections 6.5. Finally, by using an RBF the simulation and the real control results of the system are presented in section 6.6.

6.1 Brushless DC Motor

Conventional DC motors are highly efficient and their characteristics make them suitable for use as servomotors. These motors need a commutator and brushes to connect mechanically, and hence require extensive maintenance. In conventional DC motors, the rotor and the field magnets are placed in the armature and stator. A brushless DC motor is unlike the conventional motor, in that it is very similar to an ac motor. The armature windings are part of the stator, and the rotor is composed of one or more magnets. AC motors are different from brushless DC motors in the way that the later detect rotor position and produce signals for controlling the electronic switches. This type of motor has the advantage of not using brushes. Brush maintenance is no longer required and the problems associated with the brushes such as sparking are eliminated. Another advantage is the motor construction, the rotor is placed inside the stator, so more cross sectional area is available for the armature winding. In addition, the lack of brushgear in brushless motors reduces the motor's size. The brushless DC motor is used extensively because of its high torque delivery.

The controlling of such motor depends on the detection of its rotor position. These motors incorporate some means of detecting the pole position on its rotor. The detectors

which have been used are, light emitting diodes and phototransistors (photo diode), inductors sensitive to inductance variation (not in use recently) and the common one, Hall effect elements which are used in this project. To use of any of these sensors the required elements are fabricated in a complex electronic circuit. This complexity may be one of the disadvantages of brushless DC motors [49,84].

6.1.1 Hall effects element

As mentioned in the previous section, several methods have been used for detecting the rotor pole position. The Hall effect method was discovered by E.H.Hall in 1878, [49] and it is a widely used method for detecting a magnetic field by using semiconductors devices (called Hall elements). With Hall effect elements used as the position sensors, all the elements are incorporated in one integrated circuit chip. For simplicity only one chip is placed in the brushless DC motor as shown in Figure (6.1).

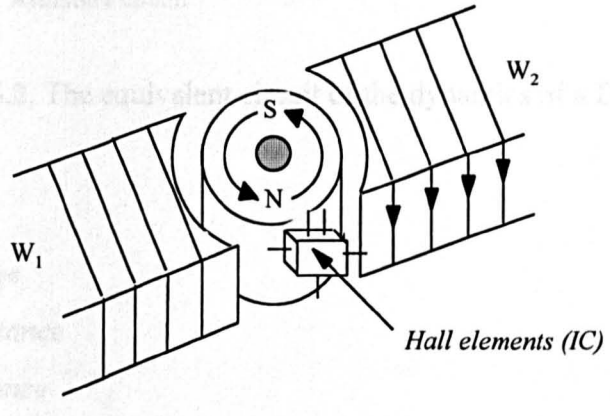


Figure 6.1, Basic principles of the brushless DC motor using Hall effect element

The chip consists of many elements i.e. resisters and transistors. The output signal from the Hall element operates transistors to control the electrical currents in the stator windings W_1 and W_2.

6.1.2 The Motor Dynamic and Transfer Functions

The dynamic characteristics refer to how a motor responds to operational commands. So when considering the dynamic characteristics, it is convenient to represent a motor using an equivalent circuit, which is used to explain and analyse the dynamic behavior of a DC motor. Consider the schematic diagram of the armature controlled DC motor depicted in Figure (6.2), where

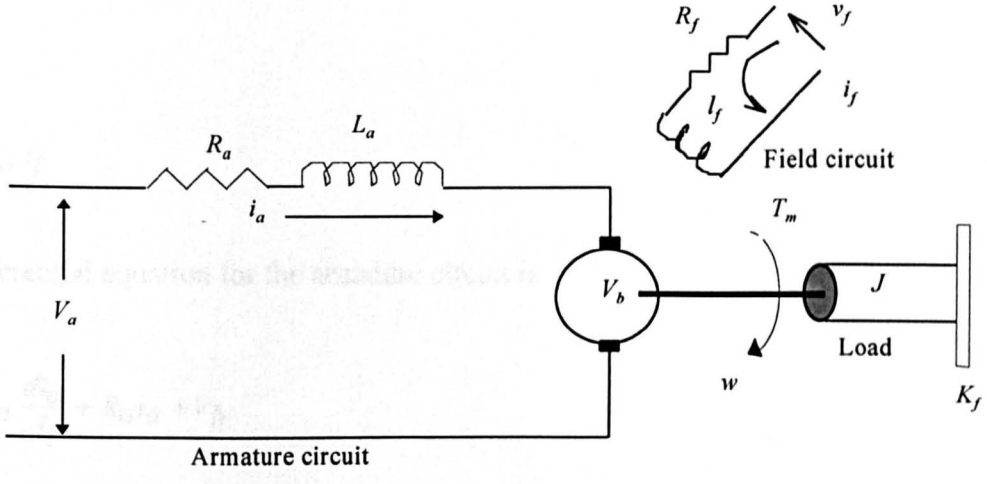


Figure 6.2, The equivalent circuit of the dynamics of a DC motor

V_a = *armature voltage*

L_a = *armature inductance*

R_a = *armature resistance*

i_a = *armature current*

V_b = *back emf volts*

i_f = *field current*

K_f = *viscous-friction*

K_b = *back emf constant*

K_t = *torque constant*

T = *motor torque*

J = *moment of inertia*

w = *angular velocity*

where,

$$V_b = k_b w \tag{6.1}$$

and

$$T = K_t i_a i_f \tag{6.2}$$

the differential equation for the armature circuit is

$$V_a = L_a \frac{di_a}{d_t} + R_a i_a + V_b \tag{6.3}$$

Substituting equation 6.2 into equation 6.5 yields

$$V_a = L_a \frac{di_a}{d_t} + R_a i_a + k_b w \qquad \text{- motor electrical equation} \tag{6.4}$$

The armature current produces the torque that overcome the inertia and friction.

$$T = J \frac{dw}{d_t} + k_f w \qquad \text{- motor dynamic equation} \tag{6.5}$$

Substituting equation 6.2 into equation 6.5 produces

$$K_t i_f i_a = J\frac{dw}{d_t} + k_f w \tag{6.6}$$

Assuming the initial conditions are all zero and taking the Laplace transforms of equations (6.1,6.4 and 6.6) obtaining the following equations

$$k_b w(s) = V_b(s) \tag{6.7}$$

$$(L_a s + R_a) i_a + k_b w(s) = V_a(s) \tag{6.8}$$

$$(Js + k_f) w(s) = K_t i_f i_a(s) \tag{6.9}$$

Simplifying equations (6.7-6.9) , gives the following transfer function

$$\frac{w(s)}{V(s)} = \frac{K_t i_f}{(L_a s + R_a)(Js + K_f) + K_t K_b i_f} \tag{6.10}$$

The inductance l_a in the armature circuit is usually small and may be neglected. If so, then the transfer function given in equation (6.10) can be reduced to

$$w(s) = \frac{K_m}{(T_m s + 1)} V(s) \qquad \text{- desired speed} \tag{6.11}$$

where,

$$K_m = \frac{K_t i_f}{(R_a k_f) + K_t K_b i_f} \qquad \text{- motor gain constant} \tag{6.12}$$

$$T_m = \frac{R_a J}{(R_a k_f) + K_t K_b i_f} \qquad \text{- motor time constant} \tag{6.13}$$

For the field circuit shown in Figure (6.2), there is an inductance l_f in series with a resistance R_f. Thus, for that circuit

$$V_f = L_f \frac{di_f}{d_t} + R_f i_f \tag{6.14}$$

From the above equation, the field current i_f leads to the production of a magnetic field and hence a torque acting on the armature coil as indicated by equation (6.2). Many authors assume i_f to be constant, but in reality it is not. However, if the field current varies, torque varies as well, hence the flux (ψ) will be effected. The relation between field current and flux is shown in Figure (6.3).

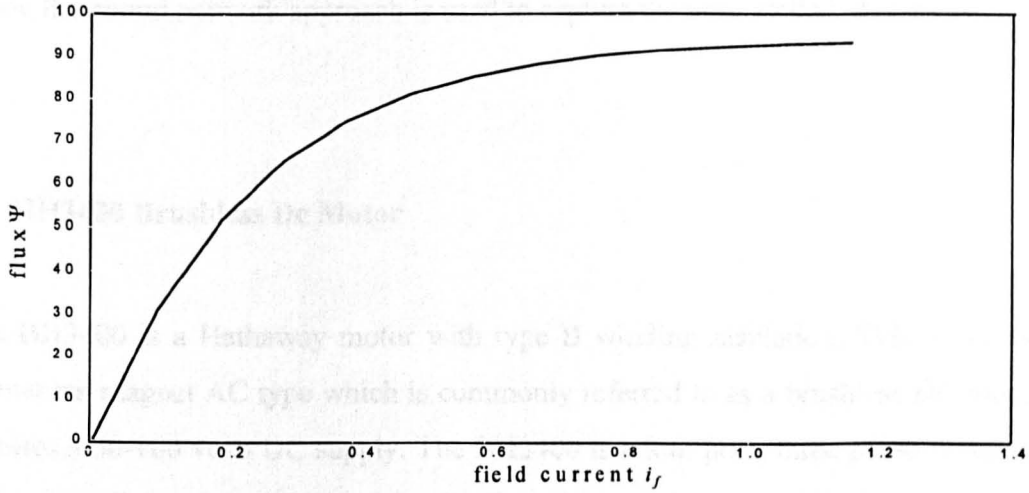


Figure 6.3, Relation between flux and field current

In addition i_a can vary the flux (small effect). The armature reaction may also change the field in the machine which in turn changes the torque constant. According to these assumptions, equation (6.11) can be assumed non-linear and therefore, DC motors are non-linear systems [68,77,79]. Furthermore, in practice most plants contain several non-linear elements which can not be easily described by mathematical models. To some

extent every plant is non-linear because each will eventually reach some physical limit which prevents its output increasing or decreasing indefinitely. This is due to a phenomenon called *'saturation'*. The effects of this phenomenon is to effectively reduce the gain at high amplitudes and to slow the plant response to disturbance. Another example, is *'velocity limiting'*, where the speed of a motor must be limited to prevent centrifugal forces damaging the motor and couplings. Such a velocity limited motor can follow slow positional changes requiring speeds below the limit, but will lag when called to perform high speed changes. Moreover, there are some other factors which cause non-linearities in motors, e.g. Hysteresis (backlash) and Dead zone. Furthermore, the torque speed currents are parallel for a relatively wide speed range but they may not be equidistant; i.e., for a given speed, the torque may not vary linearly with respect to the control voltage [25,68]. In reality the non-linearities exist in some operating conditions, hence the neural network approach is used to capture the unmodelled dynamics.

6.2 BH3400 Brushless Dc Motor

The BH3400 is a Hathaway motor with type B winding insulation. This motor is of a permanent magnet AC type which is commonly referred to as a brushless DC motor and requires a 60-100 volts DC supply. The BH3400 is a four pole, three phase Δ connection and its maximum no load speed is 12000 rpm at maximum input voltage. The control specifications has been given such that the motor achieves and maintains speeds ranging from 1000 rpm to the maximum speed to an accuracy of ± 0.1% .

6.2.1 Brushless DC Motor Controller Card (BMCC)

The servo motor cannot be directly driven by applying a current or voltage from the mains power supply. Therefore, a control card (power amplifier) is needed to control the required input signal. In this project a card is designed for a three phase brushless dc

motor using Hall effect sensors, to derive and control the motor speed. The designed Brushless dc Motor Control Card (BMCC) is an analogue servo amplifier, requiring no software or programming skills to set-up. The control card is supported by a protection circuit to assure system integrity and to save it from overvoltage problems. The card is composed of many components, the main one is a UC3625 an integrated circuit (IC). The UC3625 motor controller IC integrates most functions required for high-performance brushless DC motor control, into one package. When coupled with external power MOSFETs, this IC performs fixed frequency pulse width modulation (PWM) motor control in either voltage or current mode, whilst implementing closed loop speed control and braking with smart noise rejection, safe direction reversal, and cross condition protection. This IC is rated for operation over the temperature range of 0°C to 70°C. The block diagram of the drive circuitry is shown in Figure (6.4). For more details about the drive, see appendix A.

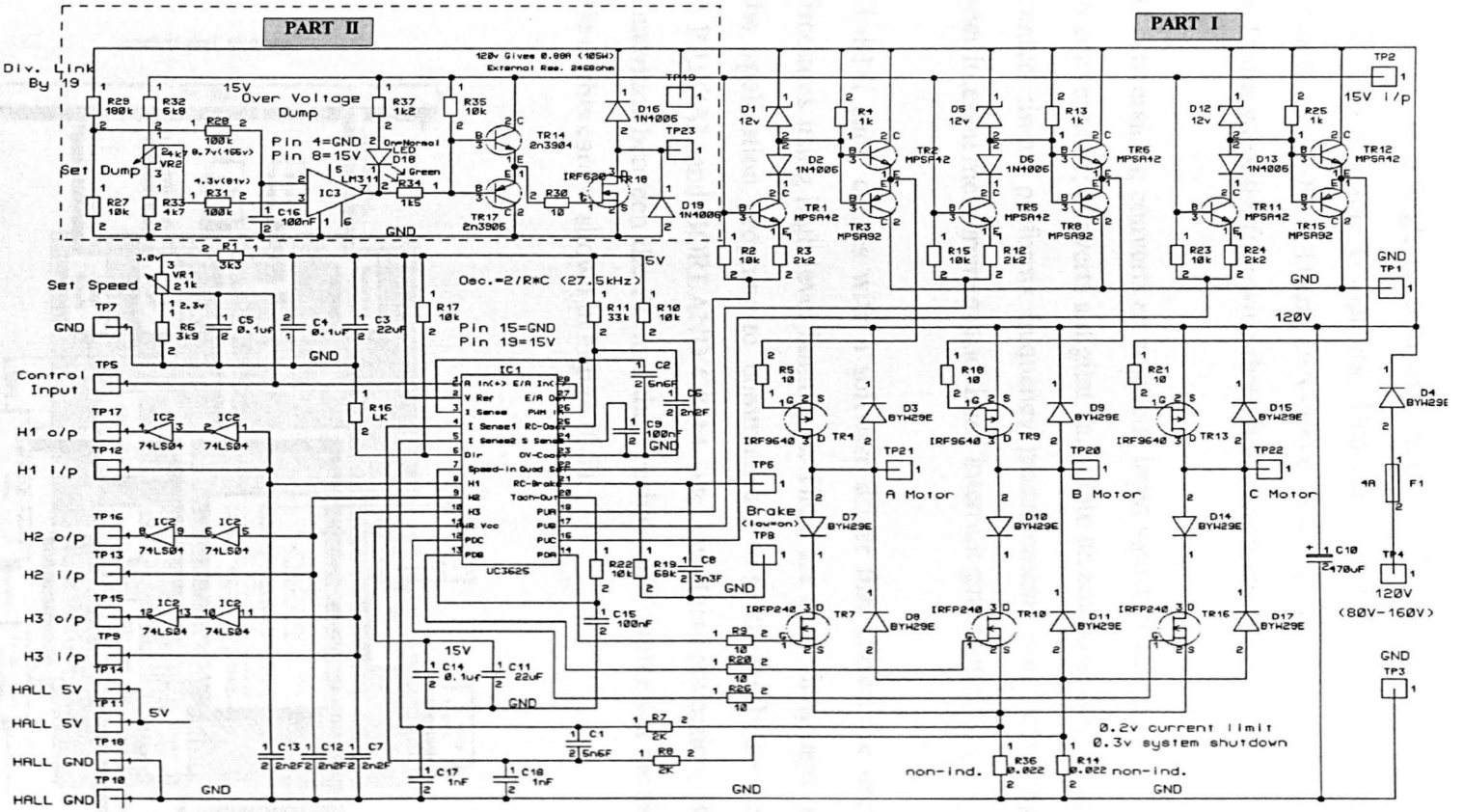


Figure 6.4, Brushless DC motor driver circuit

6.2.2 Data acquisition card

The PLC-818L is a high performance multifunction data acquisition card for IBM PC/XT/AT or compatible computers. This card offers most desired measurement and control functions: 12-bit A/D and D/A conversion, digital input, digital output and timer /counter. In our work the following three functions were used

- A/D conversions; converts an analogue input signal to digital form.
- D/A conversions; converts a digital value into an analogue output signal, and
- Counter/ timer; performs frequency measurement, event counting, pulse output, timer interval measurement and timed interrupt generation.

The PCL-818l card, comes with a software driver that allows the user to control the card's functions using high level languages. There are many language interfaces which allow the application program to communicate with the software driver; BASICA, TURBO PASCAL and BORLAND C/C++ etc. In this application, BORLAN C was used to interface between the PC and the Brushless DC motor via the BMCC [71]. The schematic of this card is shown in Figure (6.5).

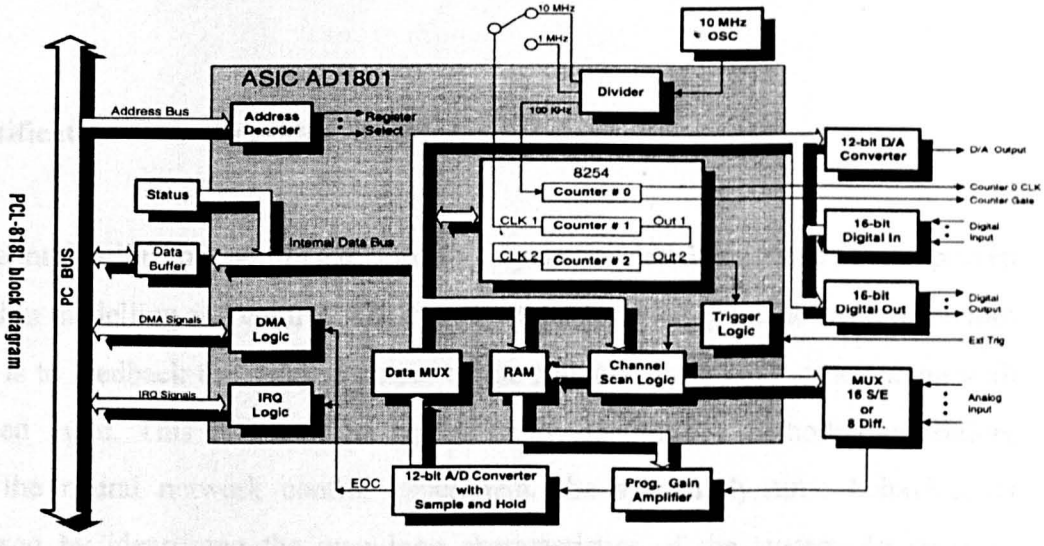


Figure 6.5, The PCL-818l block diagram

6.2.3 Experiment Set-up

The experiment set-up of the brushless DC motor control system, is depicted in Figure (6.6). The experiment consists of a Brushless DC motor, a Frequency to Voltage (F/V) converter, a Personal Computer (PC) and an interface circuit board PLC-818l.

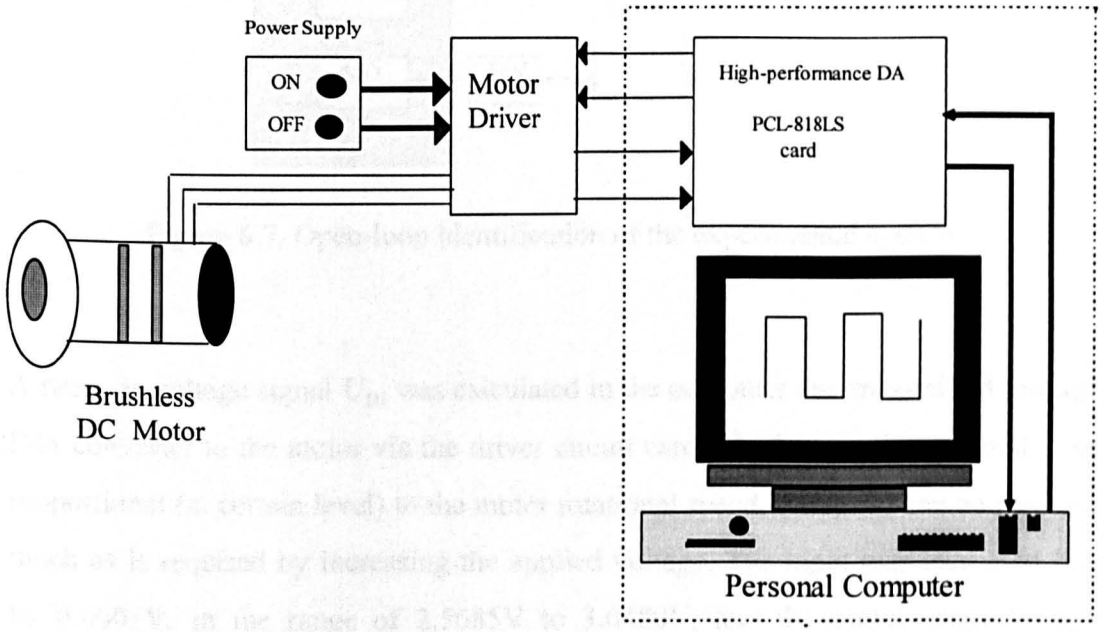


Figure 6.6, The control system setup

6.3 Identification of the Brushless DC Motor

In the identification of non-linear models, dynamic networks have been proven successful in modelling and control. The usual method for making the network behaviour dynamic, is to feedback the delayed output of the network to its input space along with the delayed input. This is known as tapped delay line (TDL) methodology. Before realising the neural network control experiment, the system dynamic behaviour is characterised by identifying the open-loop characteristics of the system. In order to

identify the model, numerous sets of data should be obtained from the real system to ascertain the system behaviour. The motor is loaded by a disc with a total mass of approximately 1.0 kg. The experiment was set-up as shown in Figure (6.7) and the procedure is described in the following.

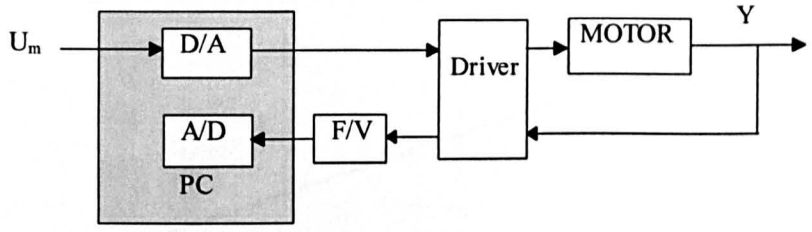


Figure 6.7, Open-loop identification of the experimental system

A ramp dc voltage signal U_m was calculated in the computer and transmitted through the D/A converter to the motor via the driver circuit card. The input voltage signal is nearly proportional (at certain level) to the motor rotational speed, the speed can be increased as much as is required by increasing the applied voltage. The input was increased linearly by 0.0005V, in the range of 2.5685V to 3.0480V, and the motor outputs Y_m were transmitted through the F/V and A/D converters to the PC. The first four samples of the motor input voltage, output voltage and the equivalent frequencies and motor speed are illustrated in Table (6.1).

Table 6.1, The motor inputs-outputs data

No. of samples	Motor input V	motor output V	Output frequency HZ	Motor speed rpm
1	2.5680	1.3194	33	1000.04
2	2.5685	1.9990	50	1515.14
3	2.5690	2.7986	70	2121.21
4	2.5695	3.5982	90	2727.27
.	.	.	.	.

The input-output data is plotted in Figure (6.8). It is clear from the output results shown in Figure (6.8b), that the brushless DC motor exhibited a non-linearity. This non-linearity problem may be attributed to some of the reasons stated in section (6.1.2).

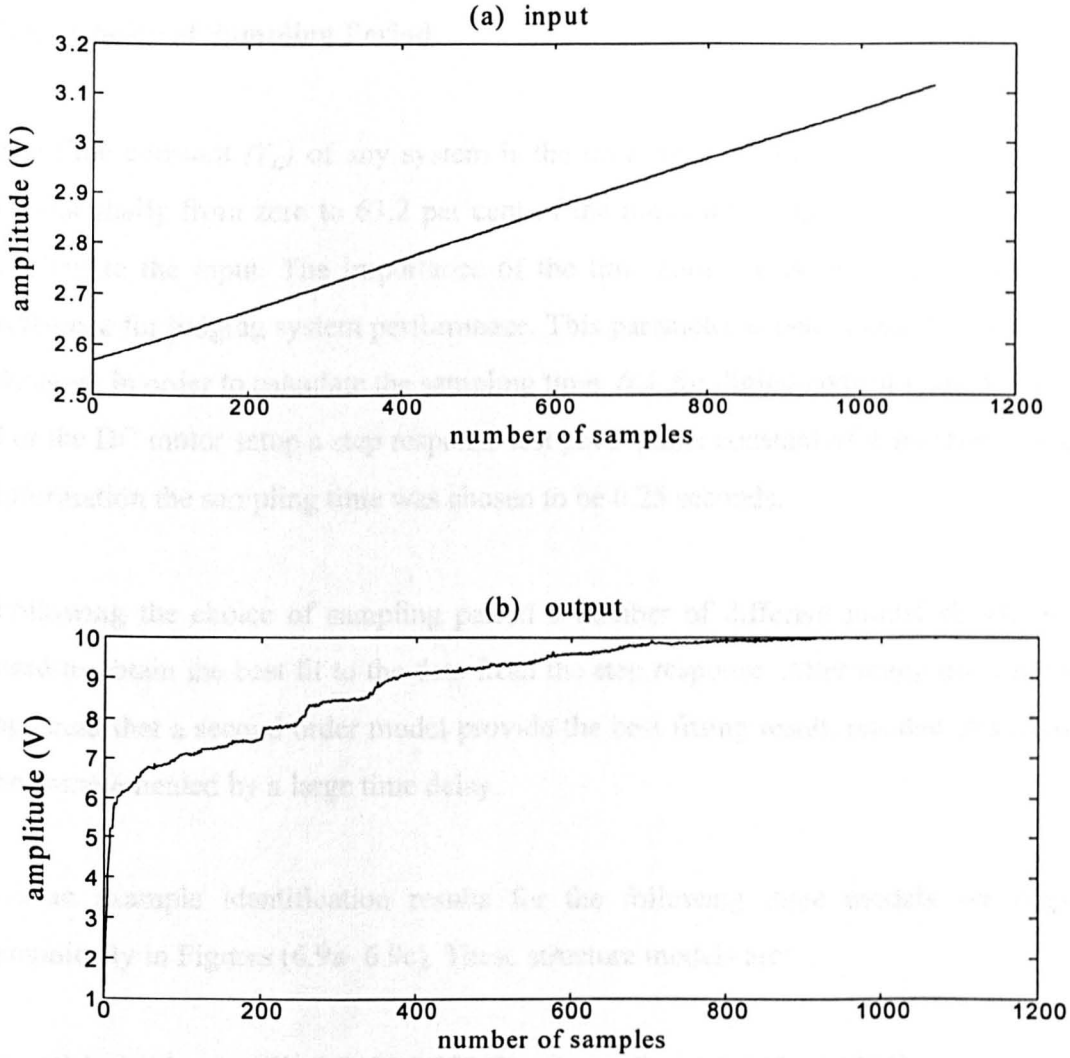


Figure 6.8, A typical DC motor (a) input and (b) output data presentation

Procedures for finding the solutions of problems involving such non-linear systems are extremely complicated. Because of the mathematical difficulty attached to non-linear systems, one often finds it necessary to introduce equivalent linear system in place of non-linear ones. The modelling of such non-linear systems by linear system models may

give quite accurate results. In this work, the MATLAB function ARX was applied to the real system input-output data, for identification purpose.

6.3.1 Choice of Sampling Period

The time constant (T_C) of any system is the time required for the system output to rise exponentially from zero to 63.2 per cent of the maximum output, when a step signal is applied to the input. The importance of the time constant, is that it provides a handy reference for judging system performance. This parameter is determined by the controller designer in order to calculate the sampling time, (t_S), for digital control systems [5,12,43]. For the DC motor setup a step response test gave a time constant of 4 seconds. Using this information the sampling time was chosen to be 0.25 seconds.

Following the choice of sampling period a number of different model structures were used to obtain the best fit to the data from the step response. After many trials it become apparent that a second order model provide the best fitting result, but that this needed to be complemented by a large time delay.

As an example identification results for the following three models are displayed graphically in Figures (6.9a- 6.9c). These structure models are:

model 1 : k =1, A = [1 -1.8491 0.8504], B = [-0.1454 0.1496]
model 2: k=14, A = [1 -1.6917 0.6927] B = [0.0251 -0.0218]
model 3 : k=20, A = [1 -1.7049 0.7053] B = [0.0121 -0.0107]

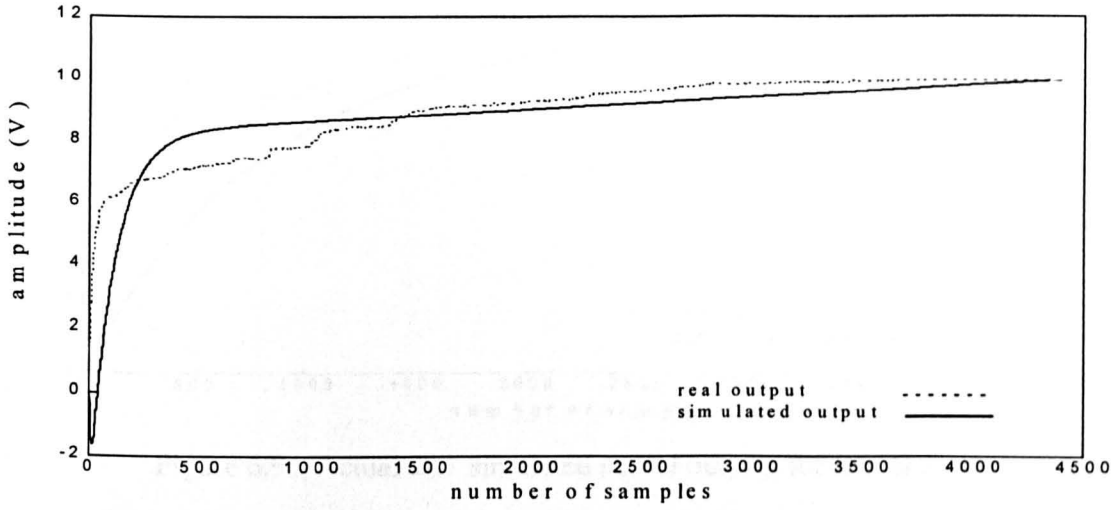


Figure 6.9a, Actual and simulated model outputs for model 1

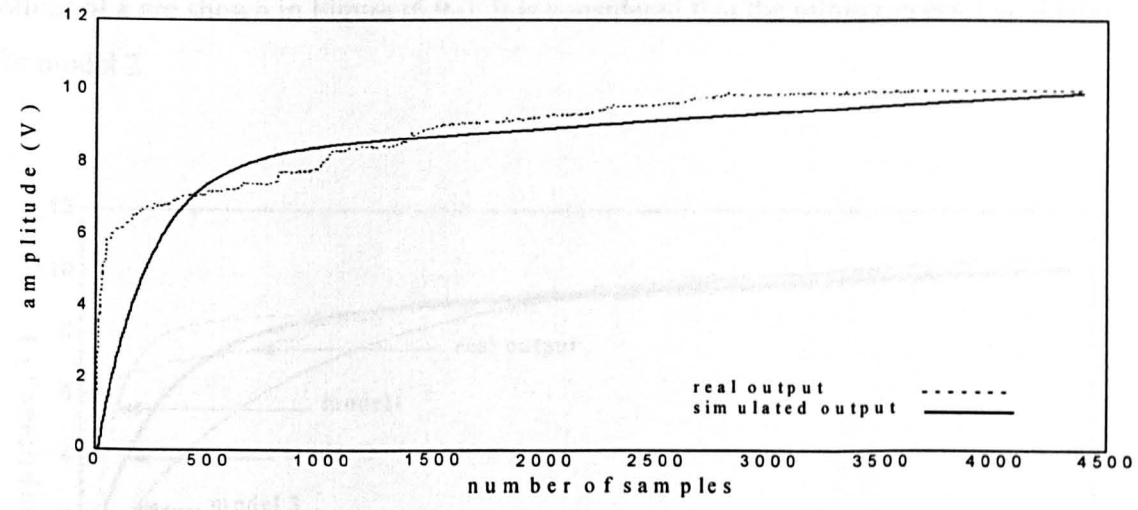


Figure 6.9b, Actual and simulated model outputs for model 2

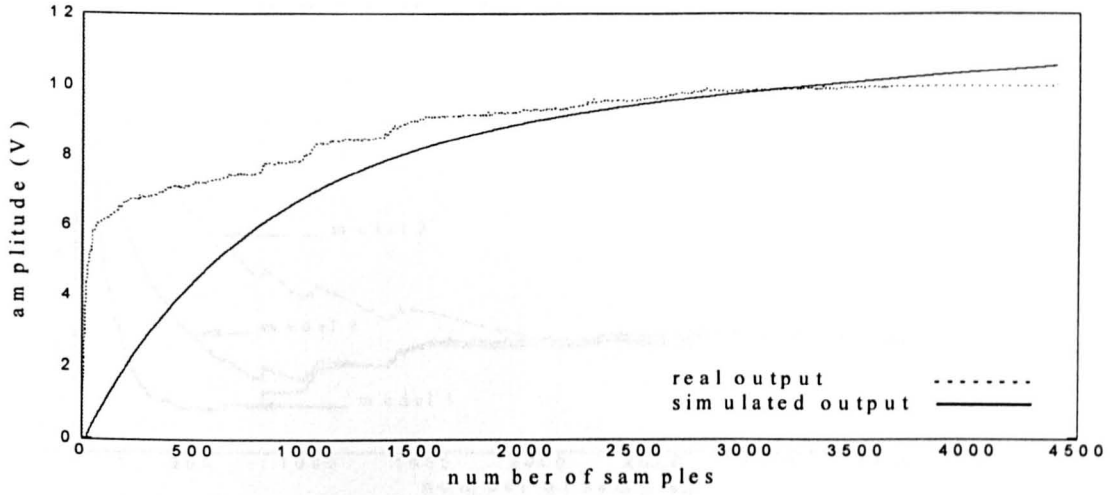


Figure 6.9c, Actual and simulated model outputs for model 3

The combination of simulated outputs is depicted in Figure (6.9d). The comparison between the errors of the real and simulated outputs of different models with different values of *k* are shown in Figure (6.9e). It is considered that the minimum error is obtained for model 2.

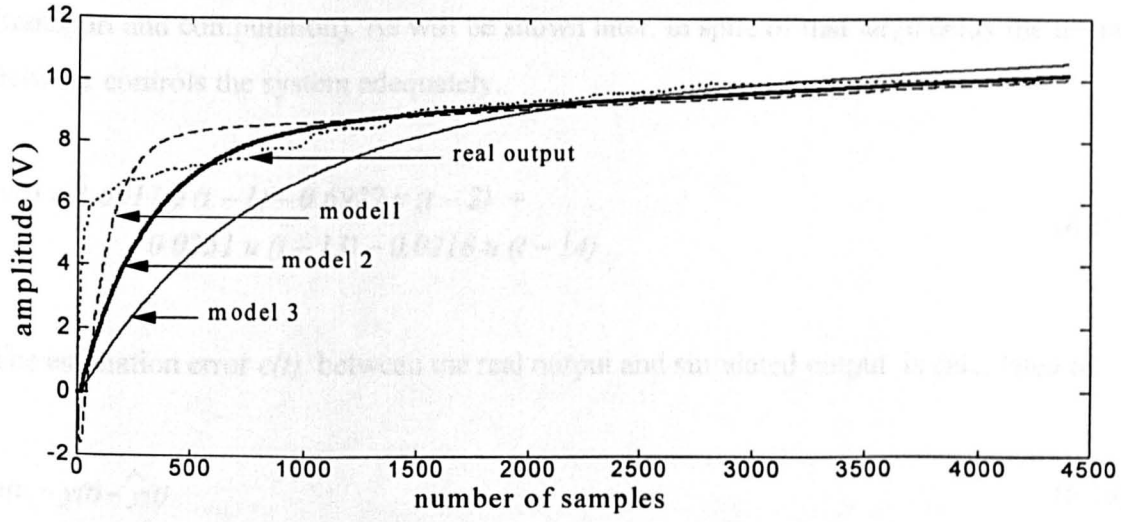


Figure 6.9d, Comparison between the actual and simulated model outputs for models 1,2 and 3

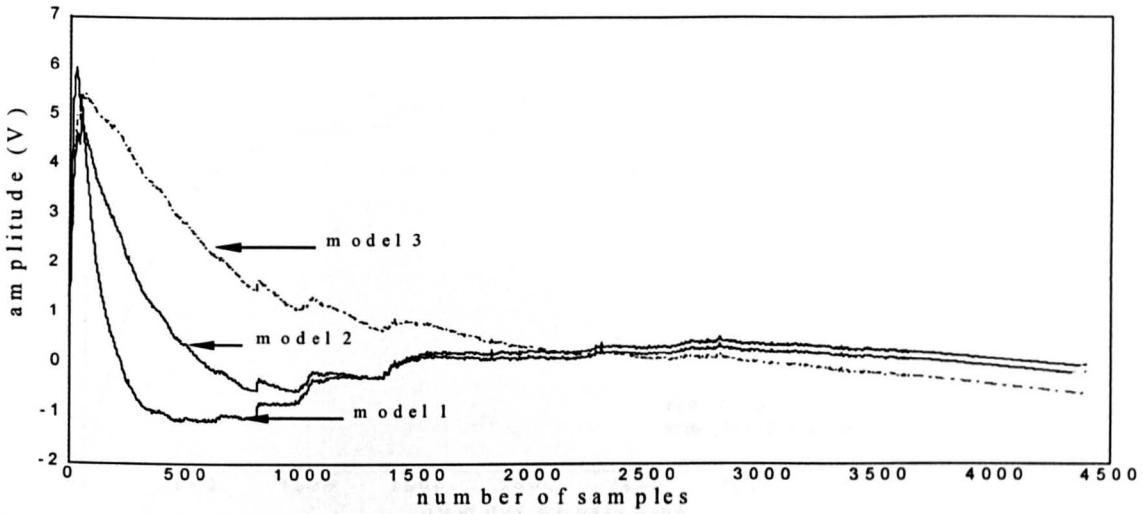


Figure 6.9e, The errors between the actual and simulated model outputs

For this reason, the test with 14 delays was considered and a second order linear model was found for the system using the certainty equivalence criteria. The dynamic model is represented in equation (6.15) below. The validity of the model was tested and the results are shown in Figure (6.10). From equation (6.15) it can be seen that the model provides a very long time delay. This delay can be attributed to the hardware and software programs (transport and computation). As will be shown later, in spite of that large delay the neural network controls the system adequately.

$$y(t) = 1.6917\, y(t-1) - 0.6927\, y(t-2) + 0.0251\, u(t-13) - 0.0218\, u(t-14) \tag{6.15}$$

The estimation error $e(t)$ between the real output and simulated output is calculated as

$$e(t) = y(t) - \hat{y}(t) \tag{6.16}$$

The error is plotted in Figure (6.11).

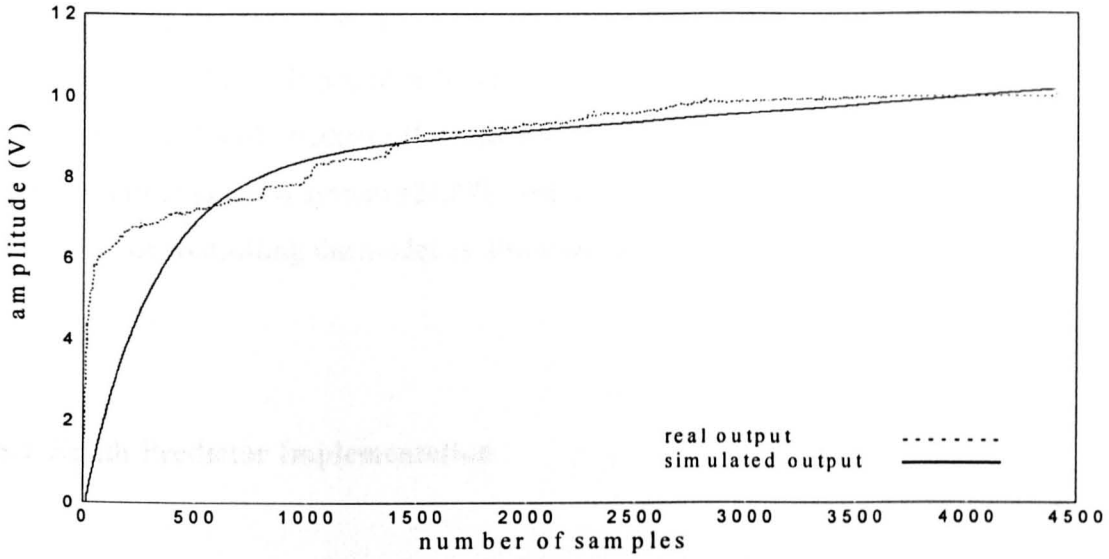


Figure 6.10 , actual and simulated outputs

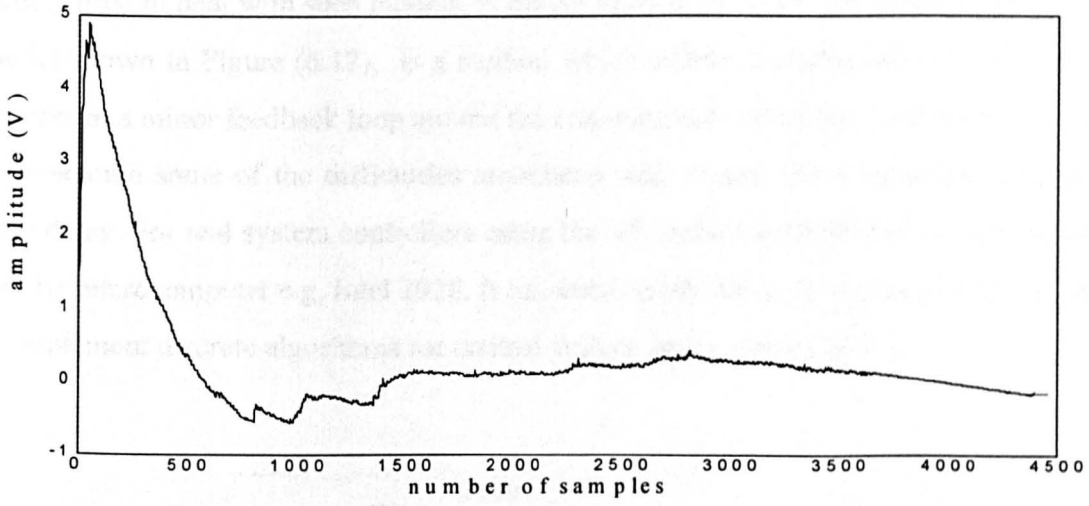


Figure 6.11, Error of the model

As seen in Figure (6.10), the simulated output matched the plant output well, but there is still some room for improvement. It is clear that the fit is an approximation and does not follow the real system. The motor input-output relationships were represented by a linear equation (6.15), in most cases the actual relationships are not quite linear. In fact, a careful study of physical systems reveals that even so-called linear systems are really

linear only in limited operating ranges (i.e. non-linear systems) [51,68]. Hence, it was justified to use the RBF neural network to identify the system. This will be discussed in section 6.5. As Smith Predictor (SP) has been well recognised in control literature to cope well with time delayed system [21,87], before using the neural networks, the SP method was used for controlling the model as discussed in the following section.

6.4 Smith Predictor Implementation

Due to the large delays in the model (6.15), this section is concerned with the effect of these delays on the control of the system. One of the best methods, which has been widely used to deal with such models, is the SP introduced by O.J.M. Smith [87]. The SP model shown in Figure (6.12), is a method which utilises a mathematical model of the process in a minor feedback loop around the conventional controller. This method is used to overcome some of the difficulties associated with controlling a processes with pure time delay. For real system controllers using the SP, some hardware devices are required i.e. the microcomputer e.g. Intel 2920. It has been noted that such processors can be used to implement discrete algorithms for control system applications [24,61].

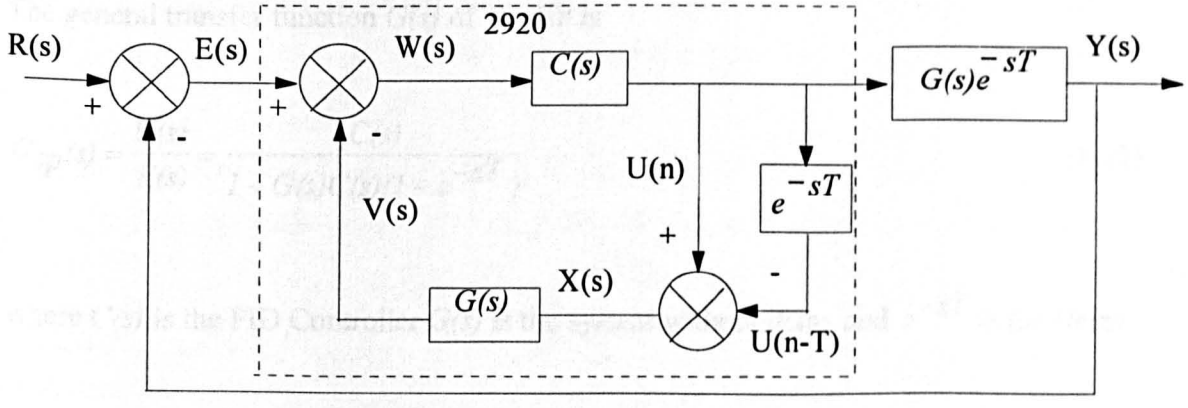


Figure 6.12, The block diagram of Smith Predictor
(arrangement for 2920 implementation)

In general, the predictor has a transfer function $G(s)\ e^{-sT}$ and a controller with transfer function $C(s)$. As shown in the above figure, the format of SP is analogue. In this project the method will be implemented in digital form. By referring to the model (6.15) and to the general form of the SP, Figure (6.12), the whole block diagram of the system, including the controller, is implemented as shown in Figure (6.13).

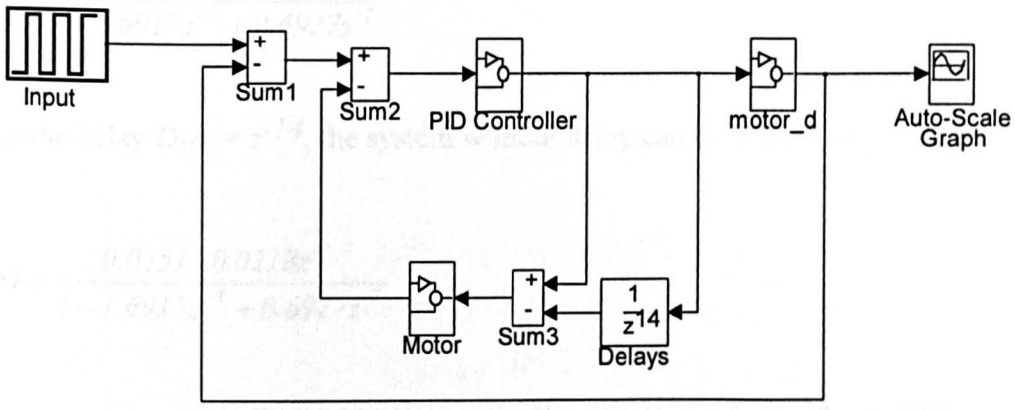


Figure 6.13, Block diagram of the whole system

The system consists of four important blocks: a DC Motor with delays (Motor _d), a DC Motor without delays (Motor), Delays, and a PID Controller. The transfer functions of each block is explained as follows.

The general transfer function $G(s)$ of the SP is

$$G_{sp}(s) = \frac{U(s)}{E(s)} = \frac{C(s)}{1 + G(s)C(s)(1 - e^{-sT})} \tag{6.17}$$

where $C(s)$ is the PID Controller $G(s)$ is the system without delay and e^{-sT} is the Delay.

This $G_{sp}(s)$ was acquired from the minimisation of the SP block diagram depicted in Figure (6.12). Based on $G_{sp}(s)$, equation (6.17), can be rewritten in the digital form $G_{sp}(z)$ as follows:

$$G_{sp}(z) = \frac{U(z)}{E(z)} = \frac{C(z)}{1 + G(z)C(z)(1 - z^{-k})} \tag{6.18}$$

By using the factors of equation (6.15), the system with delays has be obtained to be

$$G(z) = \frac{0.0251z^{-13} - 0.0218z^{-14}}{1 - 1.6917z^{-1} + 0.6927z^{-2}} \tag{6.19}$$

when the delay $D(z) = z^{-14}$, the system without delay can be written as

$$G(z) = \frac{0.0251 - 0.0218z^{-1}}{1 - 1.6917z^{-1} + 0.6927z^{-2}} \tag{6.20}$$

The output response Y would react according to the values of the PID controller parameters, proportional gain *(Kp)*, integral constant *(Ki)* and the derivative constant *(Kd)*. The general form of the PID controller is [27,30,48].

$$C_{PID}(z) = kp\left(1 + \frac{Tz}{ki(z-1)} + \frac{kd(z-1)}{Tz}\right) \tag{6.21}$$

***Test(1)*:** the parameters for the PID and Smith Predictor controller were set at the Ziegler-Nichols settings which were obtained from the open loop step response test. Thus, the obtained values of the parameters *kp, ki* and *kd* are 2.739, 1.6 and 0.4 respectively. Ultimately, the model is tested by using an input square signal with different amplitude was varied alternatively from 0.1- 0.6-0.5. The selected PID parameters are used and the results are shown in Figure (6.14) and the error between the desired and actual signals are depicted in Figure (6.15).

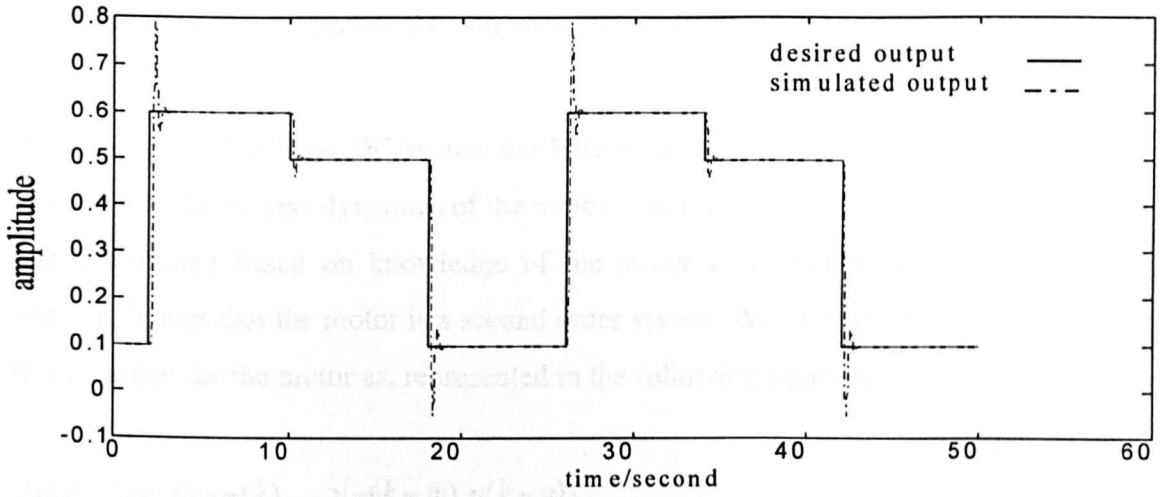


Figure 6.14, DC motor response using Smith Predictor method, desired signal (solid line) and actual signal (dotted line) (Ziegler and Nichols settings)

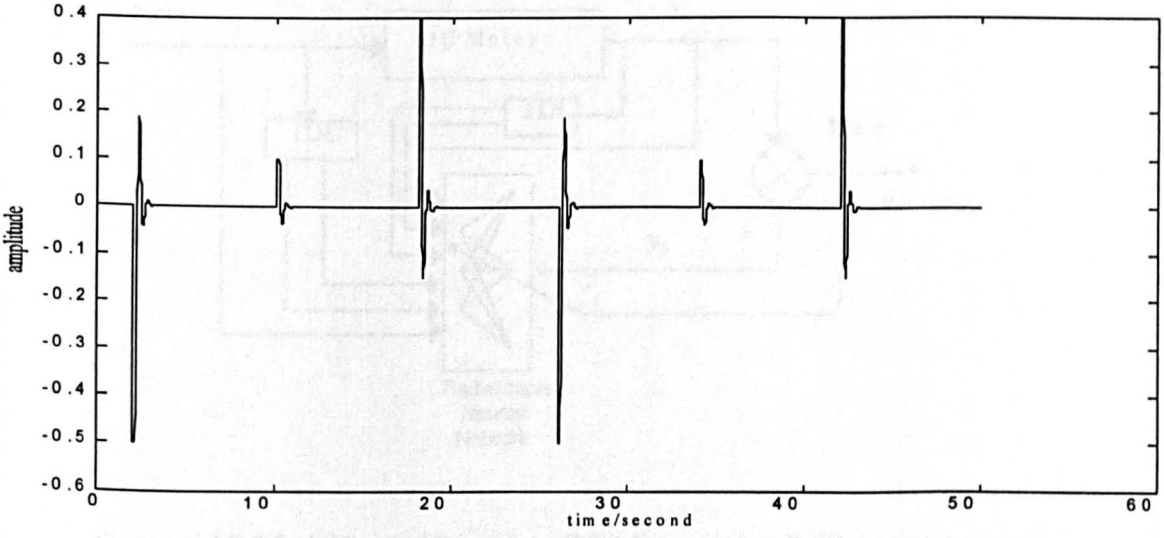


Figure 6.15, The error between the desired and actual signals of the DC motor, using Ziegler and Nichols settings

6.5 Modelling the Brushless DC Motor using an RBF neural network

In modelling a Brushless DC motor, the RBF neural network was trained to model the forward and the inverse dynamics of the motor. The input-output structure of the network was determined based on knowledge of the motor input-output and equation (6.15), which indicates that the motor is a second order system. Accordingly, a non-linear model was assumed for the motor as, represented in the following equation.

$$y_p(k+1) = f(y_p(k),\ldots,y_p(k-m),u(k-n)) \tag{6.22}$$

where *f(.)* is the non-linear function.

The representation of the forward dynamics model of the motor is shown in Figure (6.16).

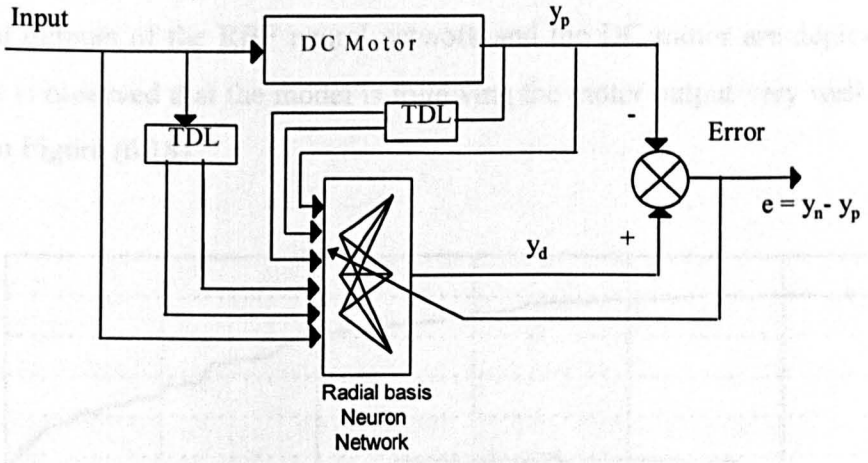


Figure 6.16, Modelling of Brushless DC Motor Using RBF neural network

The modelling procedure for the forward and inverse dynamics of the motor were based on the procedure and the techniques that have been presented and discussed in chapter 5 section 5.2 .

6.5.1 Modelling results

A fit was obtained using the collected input-output data. The time delay was set to 14 and the model orders $n_a = n_b = 2$. The number of input samples was 300 and the input to the network was delayed values of these inputs and the motor outputs. The process is carried out using the MATLAB programs for selecting the optimum parameters, where the motor output gain was 0.1. The parameters and the final results are shown in Table (6.2).

Table 6.2, The optimum motor model parameters.

NS	Initial nc	Initial p	EAL	Optimum nc	Optimum P	sse	SAGO
300	4	2	0.001	10	3	5.071×10^{-3}	2.14×10^{-5}

The final outputs of the RBF neural network and the DC motor are depicted in Figure (6.17). It is observed that the model is following the motor output very well. The error is plotted in Figure (6.18).

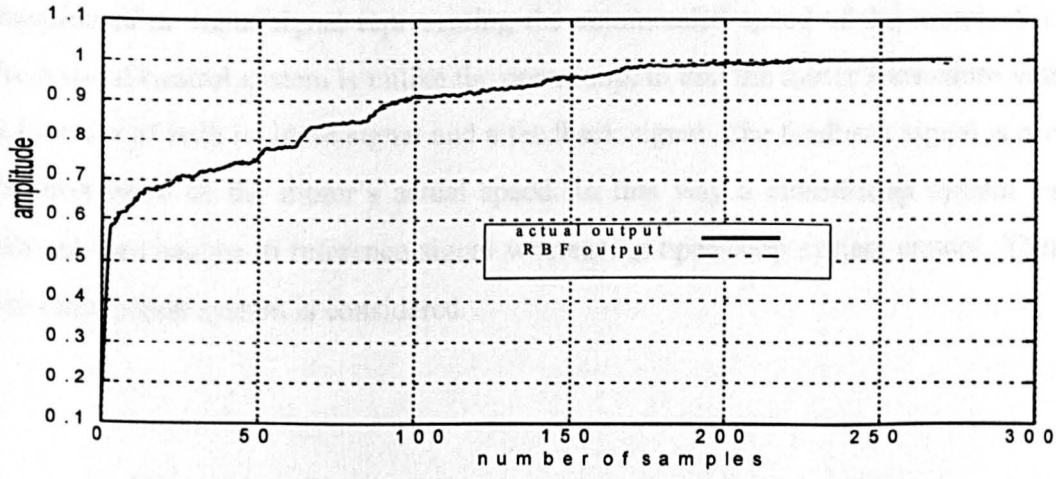


Figure 6.17, The RBF and DC motor outputs

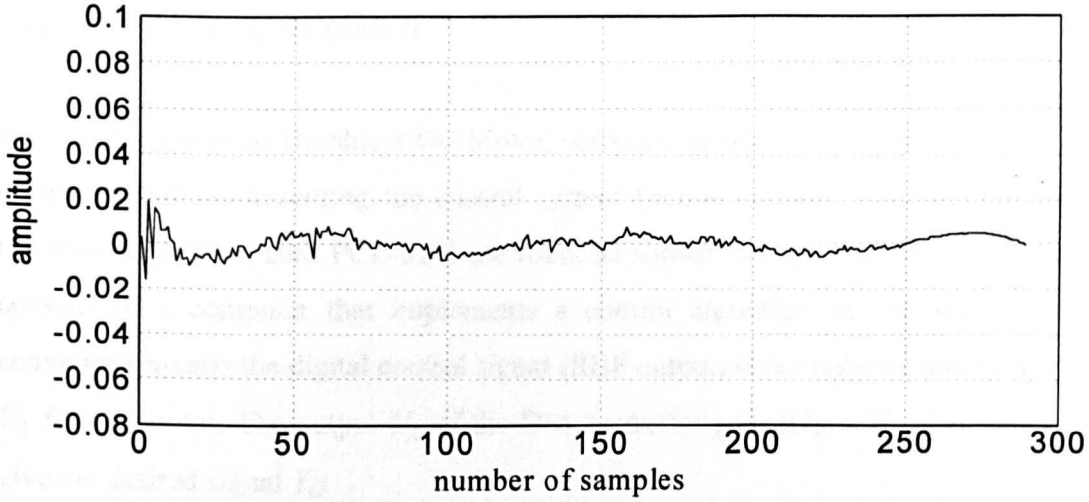


Figure 6.18, The error between RBF and DC motor outputs

6.6 Controlling the Experimental Brushless DC Motor using RBF neural network

The speed control system will be defined here as one in which a motor will run at a preselected speed without the need for an operator to monitor it. This is known as automatic or self control. Most self controllers use armature voltage control *(see equation (6.11))*. The speed control falls into two basic categories; open-loop and closed-loop. An open-loop speed control system is one in which the motor's armature voltage is strictly a function of an input signal representing the commanded speed of the motor. A closed-loop speed control system is unlike the open-loop, in that the motor's armature voltage is a function of both its input signal and a feedback signal. The feedback signal is a voltage representative of the motor's actual speed. In this way a closed-loop system can self correct for changes in reference signal whereas an open-loop system cannot. Therefore, the closed-loop system is considered.

6.6.1 Closed loop speed control

In order to control the Brushless DC Motor, software, hardware, and algorithms for designing and implementing the control system (neural network controller) along with the data acquisition card PCL-818l are used, as shown in Figure (6.19). The controller consists of a computer that implements a control algorithm in real time. The D/A converter converts the digital control signal (RBF output) into a suitable analogue voltage U_c for the motor. The output U_c of the D/A converter gain (G), will drive the motor to give the desired signal Y_d.

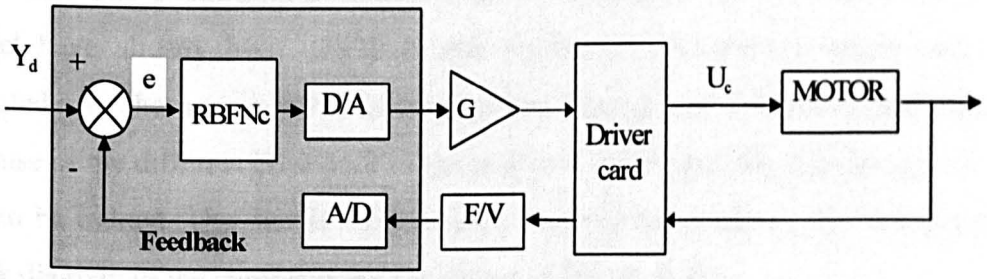


Figure 6.19, The closed-loop diagram for DC motor speed control

The output of the motor Y (motor speed), measured by sensors, is then converted by an F/V converter. The analogue to digital converter A/D converts the digital voltage Y to an analogue form and passes it to the computer as feedback. The feedback signal is subtracted from the desired signal Y_d to create the error signal e. The error signal is used to correct the weights of the RBF network controller which issues the corresponding control action U_c. This signal is applied to the DC motor until the error is minimised. As the value of desired signal Y_d is increased or decreased, the speed of the motor will follow. For changes in speed, corrective action will be taken automatically. In this way the motor will reach and run at the desired speed.

The controller is implemented by computer programs. These programs are written in the C language according to the RBF neural network mechanism. The RBF centres and width parameters were pre-selected as described in section 6.5.1, and only the weights are adapted on-line. The weights are changed to obtain the best control performance.

6.6.2 Implementation structure

The implementation structure of the Brushless DC motor control is shown in Figure (6.19). The objective is to control the motor speed by using the RBF neural network controller. The controlling procedure and the techniques for controlling the dynamics model have already been explained and discussed in chapter 5, hence they are not repeated. As the accuracy of control system depends on the motor parameters, and because of the different behaviour of the real-world plant and the simulation, two models had to be trained. The first is the simulated one and the second is the real system. The block diagram of the control system is shown in Figure (6.20).

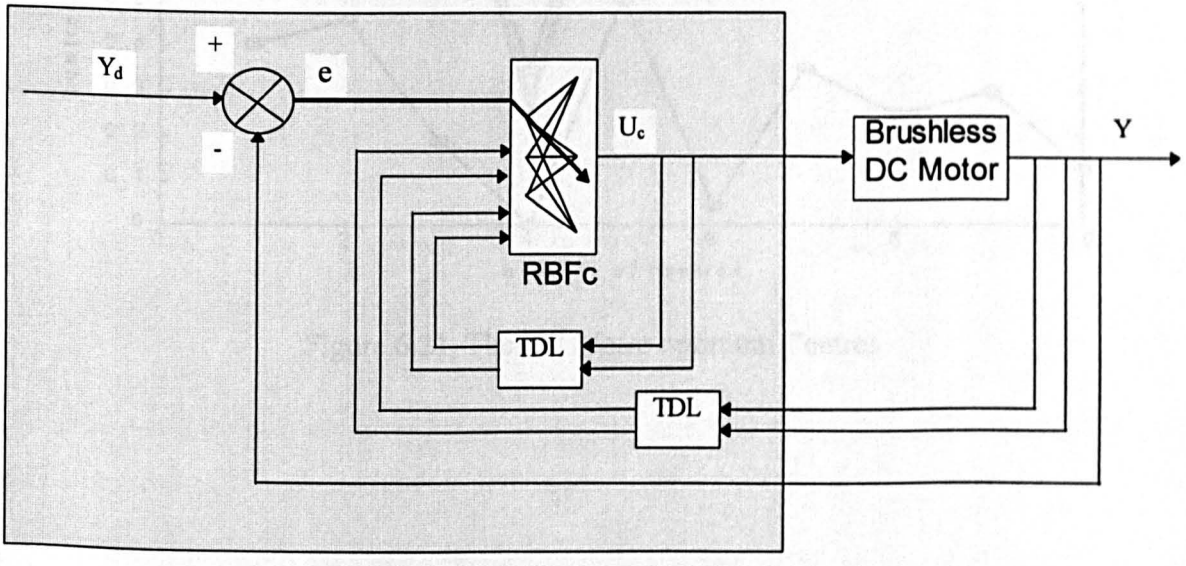


Figure 6.20, The Brushless DC Motor control system block diagram

6.6.2.1 Control of the Motor model.

During the modelling procedure, the RBF neural network parameters were selected by using the new algorithm discussed in chapter 3. The selected 10 centres and the widths of the centres are as shown in Figures (6.21) and (6.22), and have been used for training the neural network controller. The training data together with the selected centres are depicted in Figure (6.23). Simulink has been used to simulate and control the DC motor. The selected parameters are loaded into the Radial Basis Function network in the way explained in chapter 5, section 5.3. The motor model is cascaded with the RBFc controller in the same manner as in Figure (6.20) and two different signals have been used for testing the ability of the network controller.

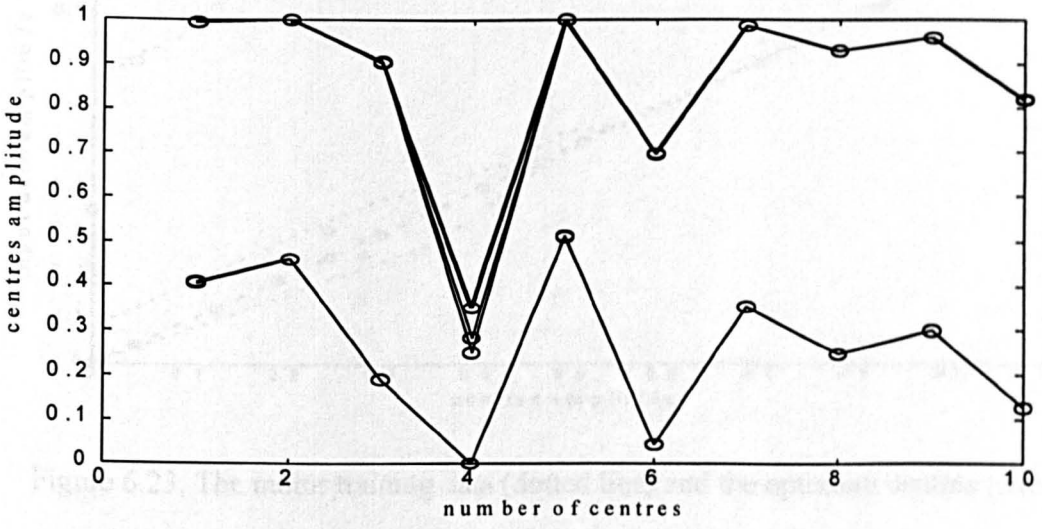


Figure 6.21, The DC Motor optimum Centres

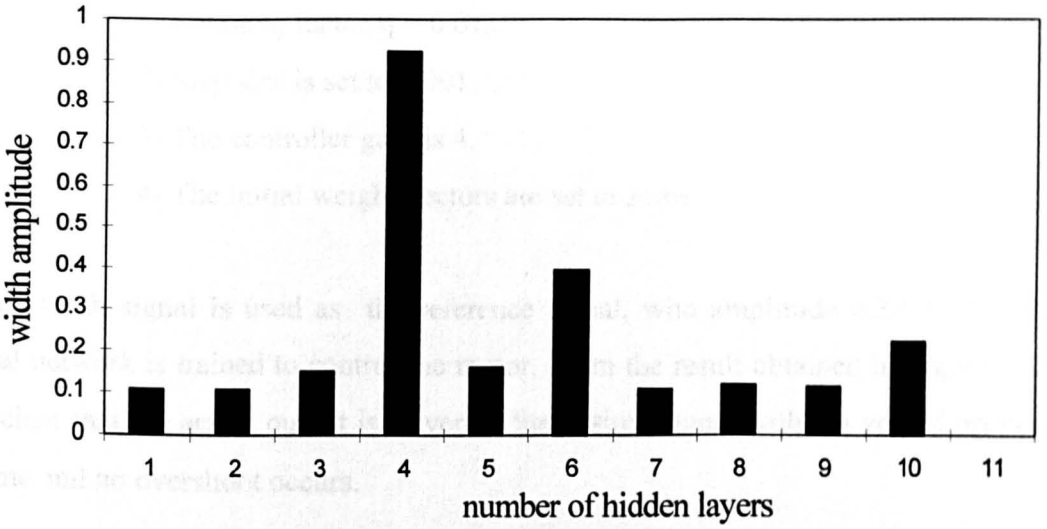


Figure 6.22, The selected widths between the motor centres

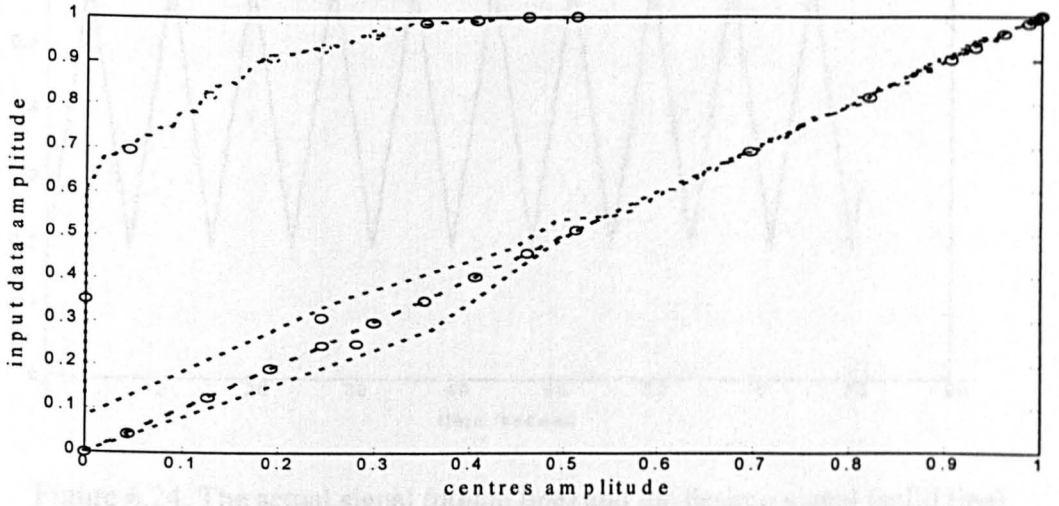


Figure 6.23, The motor training data (dotted line) and the optimum centres (circle)

Test(1): in this simulation, the controller was tested for input signal change and the other factors are set as follows.

1- Learning factor $\eta = 0.01$.

2- Step size is set to 0.001.

3- The controller gain is 4.

4- The initial weight vectors are set to zeros.

A saw tooth signal is used as the reference signal, with amplitude 0.2-0.6. The RBF neural network is trained to control the motor. From the result obtained in Figure (6.24), it is clear that the actual output is driven to the desired signal within a very short period of time and no overshoot occurs.

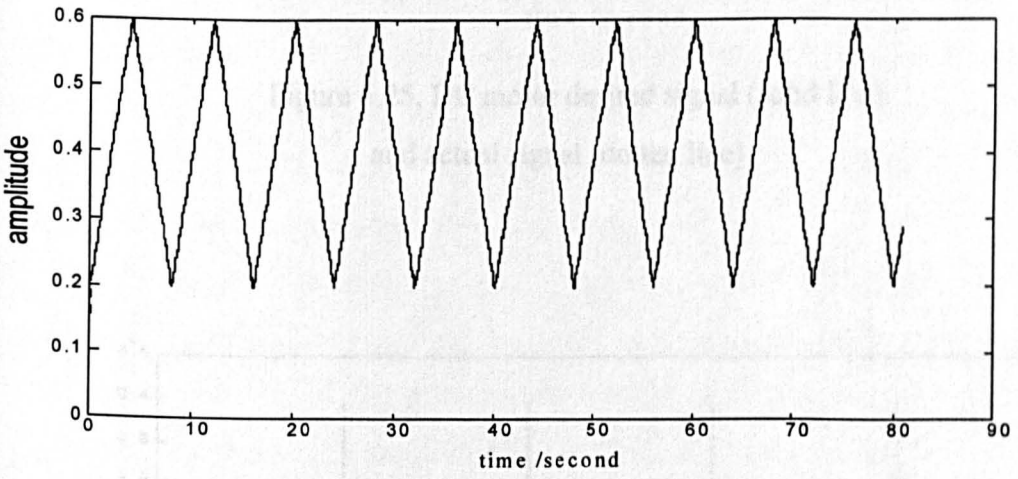


Figure 6.24, The actual signal (dotted line) and the desired signal (solid line)

***Test (2)*:** the second test is the more important one, where the desired output of the real-time system (motor) is a square wave. This signal is chosen to be the reference signal during the motor speed control simulation. The parameters are set as in the test (1) and the reference signal amplitude is varied alternatively from 0.1-0.6-0.5. The results of this test are shown in Figure (6.25). At the beginning of the simulation, overshoot occurs but only for a short period. The overshoot is eliminated when the weights of the controller

network are adjusted and the actual signal tracks the reference signal very accurately. The error signal between the signals is depicted in Figure (6.26).

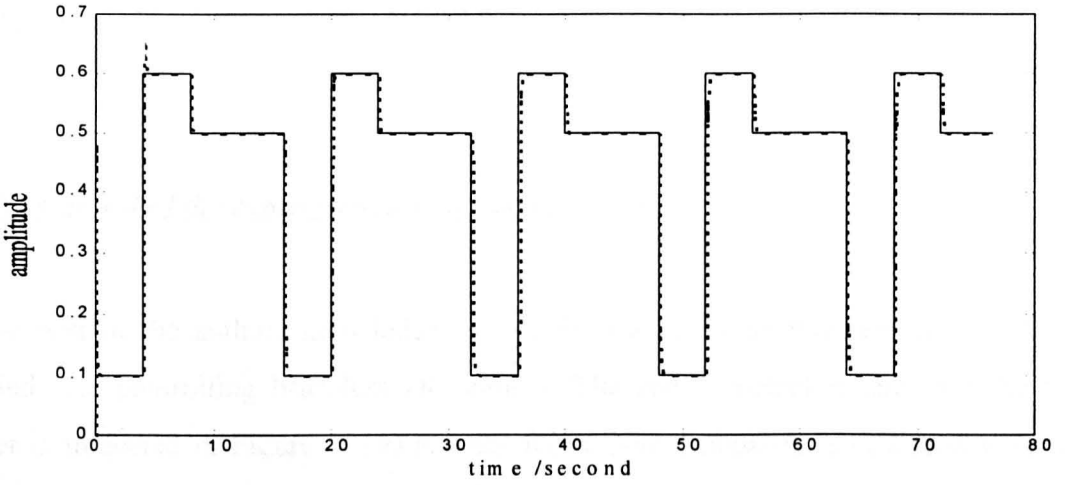


Figure 6.25, DC motor desired signal (solid line)
and actual signal (dotted line)

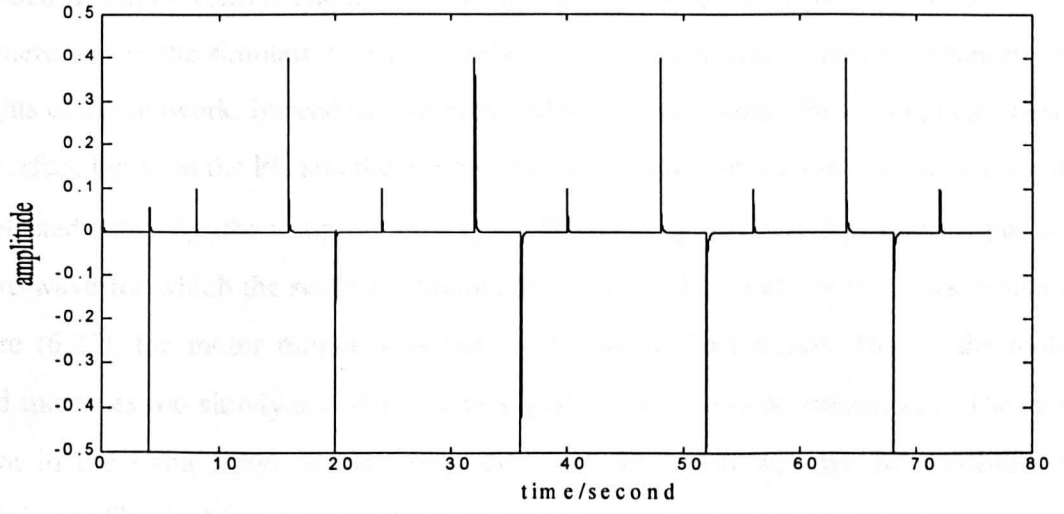


Figure 6.26, The error between the reference and
actual signals of the DC motor

In comparison, with the Smith Predictor method when the P+I controllers were used, the simulation results have clearly demonstrated that for a system containing a time delay, the use of this method dose not always give a good controller. Conversely the RBF neural network controller performed very well, revealing this method has the ability to control the real system.

6.6.2.2 Control of the experimental Brushless DC Motor

In the best of the authors knowledge, as yet the Radial Basis Function has not been applied for controlling brushless DC motors. The speed control scheme for the DC motor is indicated in Figure (6.19) and the RBFc neural network is cascaded with the motor as shown in Figure (6.20). The speed is set between 1000 to approximately 10000 rpm, which corresponds to a range of voltages on the input of the A/D converter, i.e. 2.5685-3.0480 volts. The desired square wave is applied to the DC motor via the RBFc controller and the results are described as follows.

Test (1): the results of the real-time control system using the RBFc neural controller are indicated in Figure (6.27). The motor is controlled by using the same centres and widths parameters as in the simulation test and only the LMS algorithm is used for adapting the weights of the network. Instead of Simulink and Matlab programs, the C language is used to interface between the PC and the motor. In this practical application two factors should be selected carefully: the sampling time t_s and the learning factor η. The input signal is a square wave for which the selected parameters were $t_s = 0.25$ and $\eta = 0.01$. As shown in Figure (6.27), the motor output does not follow the desired signal. That is the motor speed increases too slowly and the control signal *Uc*, is unable to compensate. The error shown in the same figure is high and the LMS algorithm has not the capability to minimise it. This is due to the fact that the learning parameter was set too low.

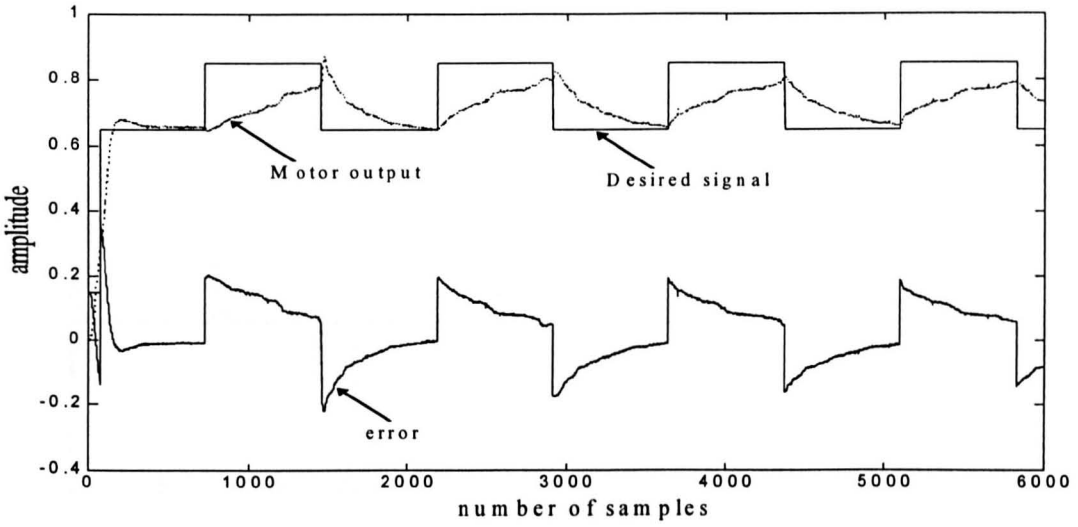


Figure 6.27, The results of test 1; motor output, desired signal and the error

Test (2): the reference signal and the sampling time are kept as in test 1 and the learning factor is changed to 0.095. The reference signal and the motor output are shown in Figure (6.28). The comparison of this results with the previous test indicates that a smother start-up and shorter adaptation time are achieved with this parameter setting. In this case the actual signal follows the desired signal well. As shown in Figure (6.29), a large error *e* appears in the beginning of the adaptation, but it is quickly minimised. The control signal is displayed in Figure (6.30).

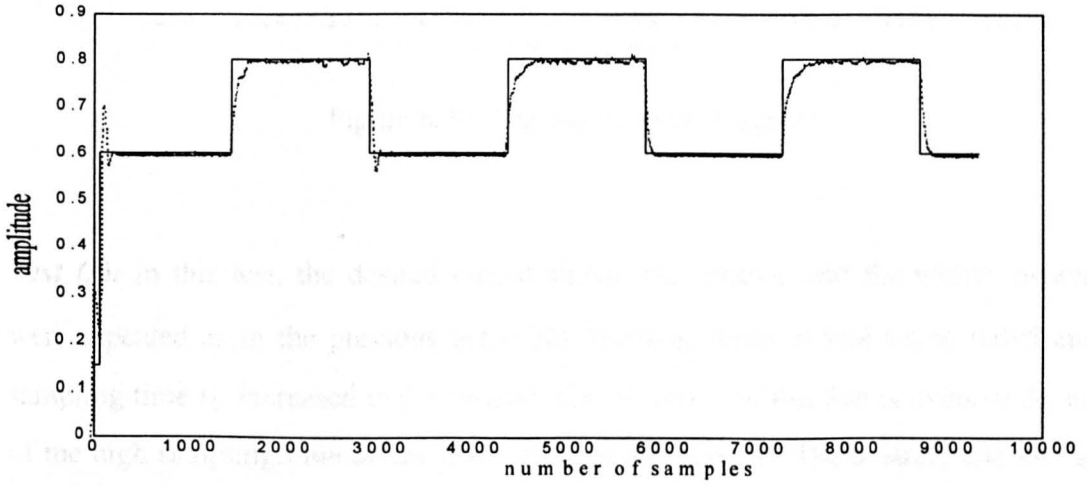


Figure 6.28, The motor reference signal (solid line) and
actual signal (dotted line)

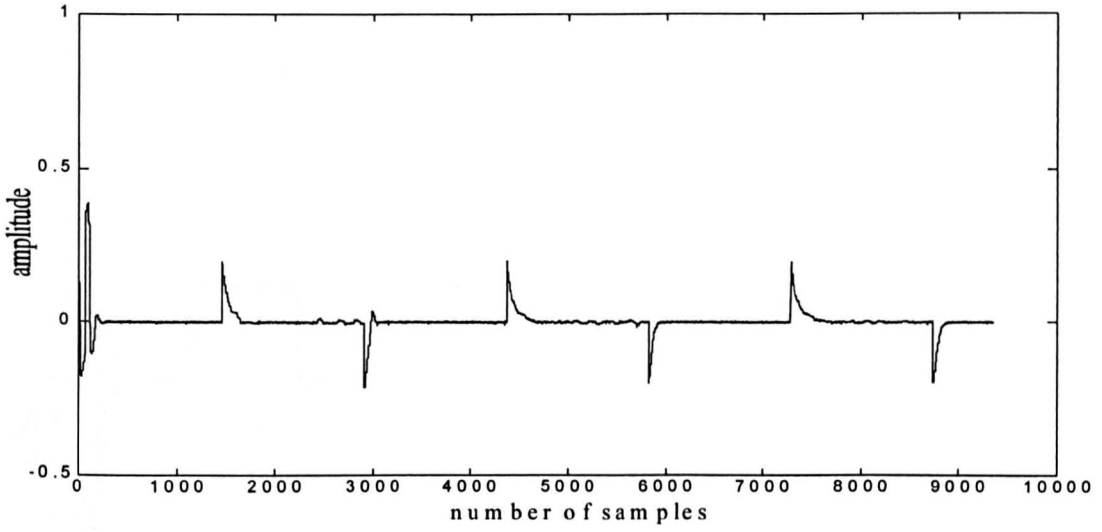


Figure 6.29, The error between the actual signal and the desired signal

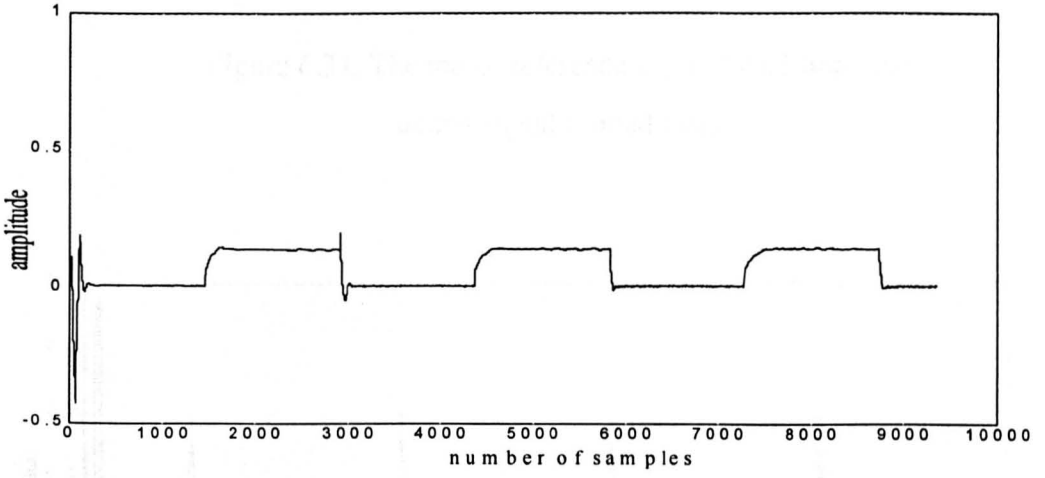


Figure 6.30, The motor control signal

Test (3): in this test, the desired output signal, the centres, and the widths parameters were repeated as in the previous tests. The learning factor η was set to 0.095 and the sampling time t_S increased to 0.4 second. The objective of this test is to show the effects of the high sampling time on the controlling of the system. The desired, and the actual output signals are depicted in Figure (6.31) below. As seen in this figure, the actual signal does not track the desired one very well, and the response of the system is slow,

especially during the decreasing of the Motor speed. The error and the control signal are shown in Figure (6.32).

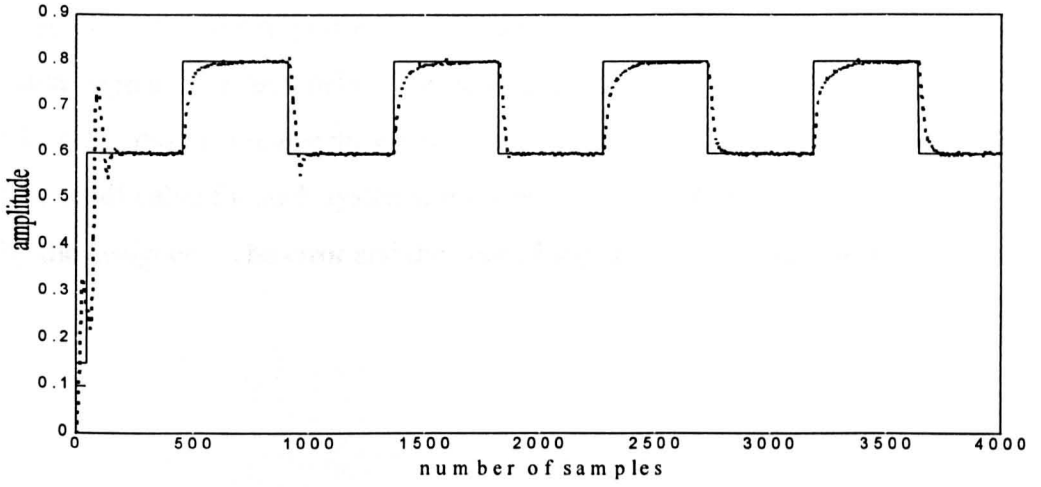


Figure 6.31, The motor reference signal (solid line) and actual signal (dotted line)

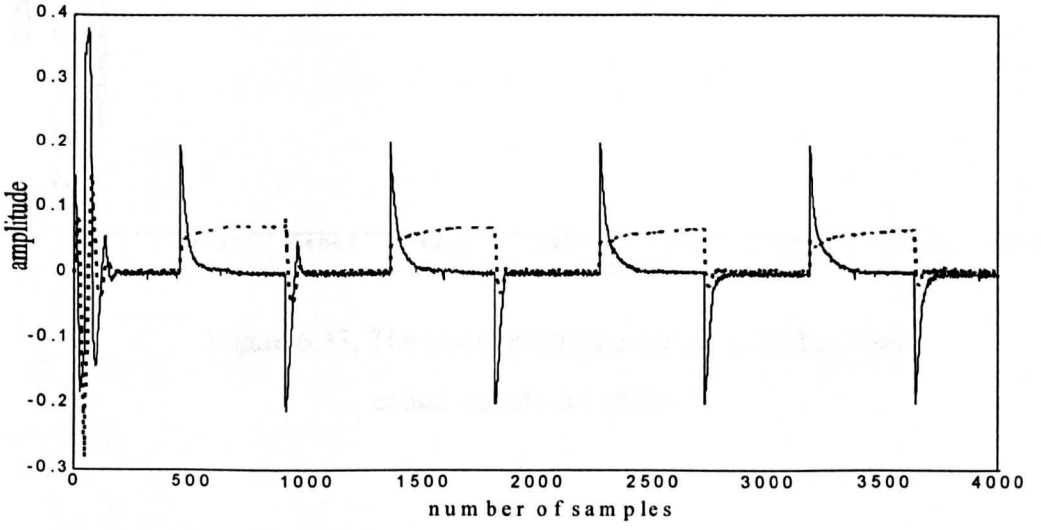


Figure 6.32, The Motor control signal (dotted line) and the error between the actual and desired signals (solid line)

Test (4): the same reference signal, centres, widths and procedure in test 1 to test 3 were used. In this examination, the learning factor was set to 0.095 and the sampling time decreased to a small value t_S = 0.1. The results of the motor response and the desired signal are shown in Figure (6.33). Comparing these results with that in test two and three, there is a very fast response and less delay between the signals. On the other hand, the actual signal decreases below the desired signal. This means the motor speed decreases whilst the motor is under the control action. Therefore, the decrease of the sampling time to a small value for such systems, must be avoided and the correct value must be selected by the designer. The error and the control signal of this test are shown in Figure (6.34).

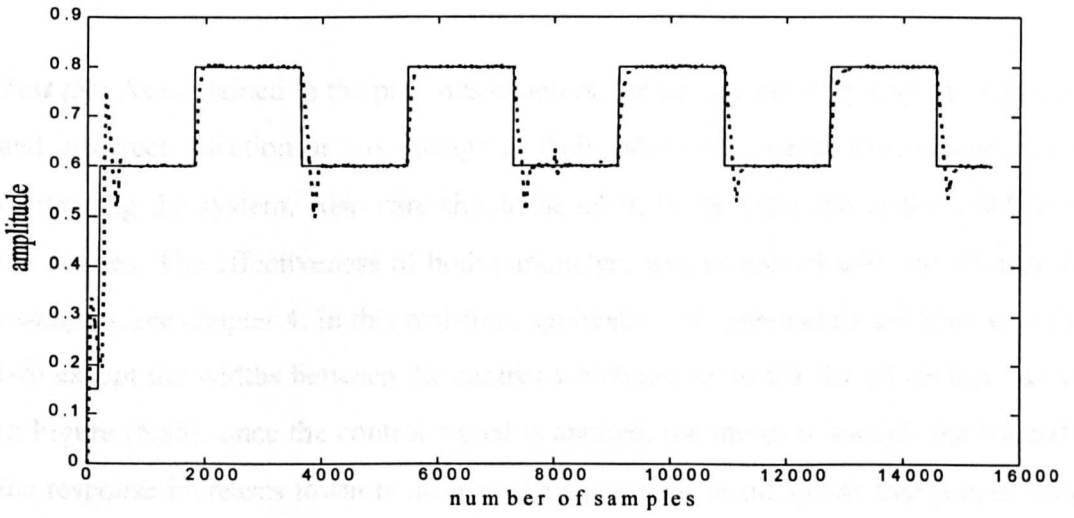


Figure 6.33, The motor reference signal (solid line) and actual signal (dotted line)

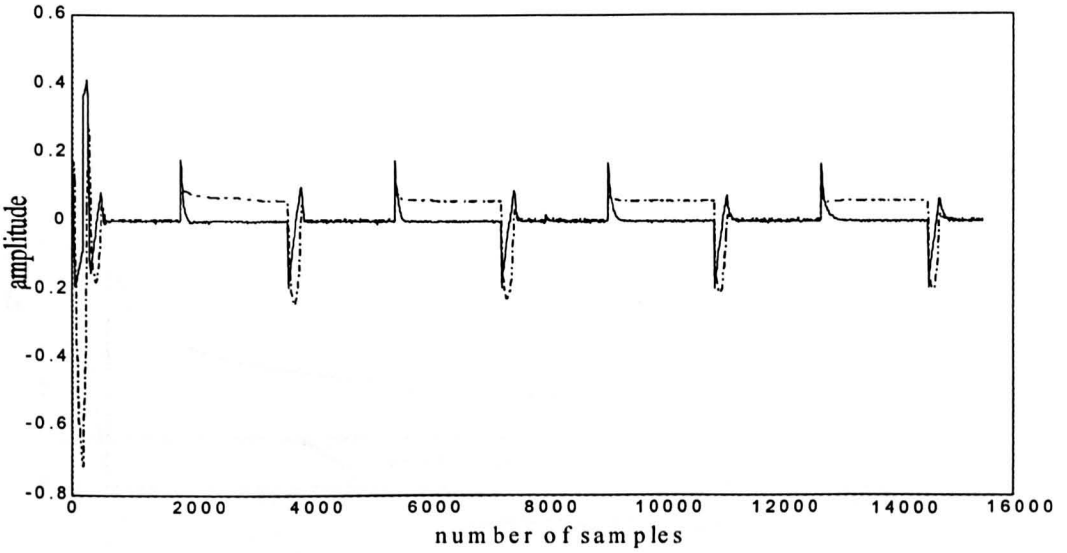


Figure 6.34, The Motor control signal (dotted line) and the error between the actual and desired signals (solid line)

Test (5): As explained in the previous chapters, the centres are very important parameters and incorrect selection or any change in their values may cause severe problems when controlling the system. Also care should be taken in the selection of the width between the centres. The effectiveness of both parameters was explained with the simulation test examples, see chapter 4. In this real-time application, all parameters are kept as in the test two except the widths between the centres which are set to 0.1 for all centres. As shown in Figure (6.35), once the control signal is applied, the motor is started, but immediately the response increases towards infinity. This practical result shows that proper selection of parameters is very important and without the adaptive algorithm presented in chapter 3 it would have been extremely difficult to find the optimum centres and widths by trial and error.

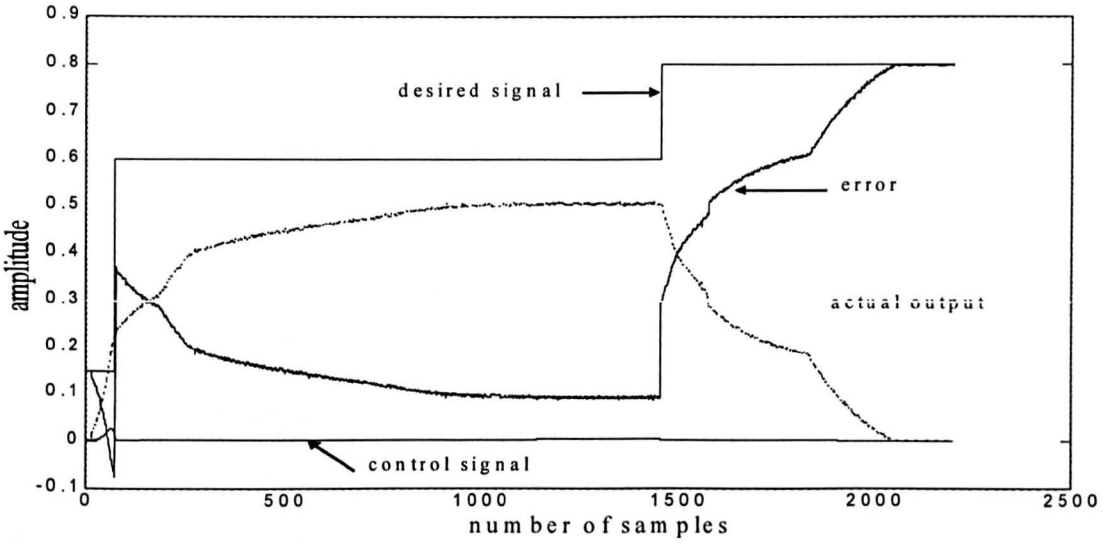


Figure 6.35, Motor output, reference signal and
the error signal (width = 0.1)

We can conclude from the above results that selection of the optimum parameters is very important for control tasks. The RBFc controller has been successfully applied for the simulated and the real-time speed control of the brushless DC motor. The results of the practical application have demonstrated the improvement of control performance. It was also proved that the sampling time and the learning factor should also be selected carefully.

CHAPTER 7

CONCLUSIONS AND FURTHER WORK

CHAPTER 7

CONCLUSIONS AND FURTHER WORK

The artificial neural network has been widely used for solving many problems, particularly non-linear modelling and control which has been investigated by many researchers. The common type of neural network is the backpropagation algorithm. This method has been intensively studied and applied for different problems with satisfactory results. On the other hand the method suffers from slow convergence and the complexity of the network construction. A brief discussion of the method was given in chapter two section 2.2.

An alternative method is the Radial Basis Function neural network (RBF). This method has been investigated by many authors and has been proven to learn faster than the backpropagation method.

One major problem in using RBF networks is the selection of the network parameters, such as the centres and the widths. Currently no mathematical method for finding such parameters exists. Therefore, in this thesis, a new adaptive algorithm for selecting the parameters was developed. The algorithm consists of a set of other algorithms: K-means clustering algorithm, P- nearest neighbour and the singular value decomposition or least mean squares algorithm. The new algorithm description with an example was given in chapter three.

The new algorithms ability to select the optimum centres and the width between the centres was proved in chapter four. The accuracy of the selected parameters are demonstrated by simulation examples of non-linear dynamic systems, where the obtained results are compared with those obtained by using the RBF conventional method.

In chapter five, the optimum parameters selected for modelling the systems in chapter four are used for controlling the same non-linear system. Thus, the direct adaptive method, using the RBF neural network, and the least mean square method were used on-line to adapt the network weights. The examples have been demonstrated when reference signals with different amplitudes, shapes and frequencies were applied. The effects of changing of the width values between the centres was also shown.

The important thing discussed and proved in this thesis is the modelling and control of real non-linear dynamic systems by means of the RBF neural network. The real system is a high speed brushless DC motor. In the best of the authors knowledge, such systems have not been controlled by this type of the neural network before.

The most important aspects in controller design of any system is the knowledge of the process characteristics. To obtain some knowledge of the DC motor, the ARX identification technique was applied to the real input-output data. Then the RBF network was configured and trained to represent a Non-linear Autoregressive Exogunes (NARX) input-output model structure and the performance of the trained network was investigated. In modelling a real brushless DC motor, the function $f(.)$ is assumed to be a differentiable unknown continuous non-linear function.

After the investigation, it was found that the system provides a very long delay, therefore the Smith Predictor method was used for controlling it. The system was then modelled and controlled by use of the RBF. The results of both methods were compared and the robustness of the network was demonstrated.

An RBF network was implemented for modelling and control of a real system with centres and widths selected by the new algorithm. For system implementation, two circuits: brushless DC motor control card and a frequency to voltage converter were designed. The RBF neural network program was written in C and compiled to interface the entire system. Satisfactory results were obtained and shown in chapter six. These

show the RBF network has the ability to model and control high speed motors; a difficult non-linear system.

Further work in this area should proceed in several directions. In this thesis only one real system was modelled and controlled, so more investigations into problems which require on-line performance, when a little *a priori* information about the system is available, are recommended.

The RBF neural network studied in this thesis can only be used for single input single output systems. The method therefore needs to be extended to multi-input multi-output systems. Some other method such as the neuro fuzzy is recommended as well.

In our work the C language has been used for programming the real system RBF controller and for interfacing the real system with the PC. This is time consuming and involves implementation difficulties, so the Simulink package is recommended in the future with appropriate hardware interface. These hardware are readily available by vendors though still very expensive.

APPENDICES

APPENDIX A

A1- Main Brushless DC Motor Drive Circuit

The main component for designing the brushless DC control card is UC3625 (IC1). This circuit is equipped with MOSFETs (metal oxide semiconductors field effect transistors) power devises capable of self-switching OFF/ON very rapidly. This devices together with many other electronic components are used in IC1, see Figure (6.4) part I.

The six transistors (three high-side and three low-side) are connected to the motor windings A, B and C. The rotating of the motor is depends on the relation between the OFF/ON states of these transistors and the motor speed depends on the values of the motor control signal (TP5).

A2- Protection circuit

In general the motors are expensive and any damage caused by overvoltage may cost the user money, time and might be some other problems. Therefore, the protection circuit shown in Figure (6.4) part II, was designed to limit the input voltage to the brushless DC Motor.

The circuit is very simple construction and its protective mechanism may be activated by an overvoltage conditions at pins 2 and 3 in the integrator circuit (IC_3) . The IC_3 can maintain full overload protection while operating at up to 150 V.

A3- Reversing the direction of rotation

When it is not convenient or possible to reverse the polarity of the input command to the

driver card to cause the direction of rotation change for a given input, the wiring between the driver card and the motor may be changed to effect a reversal. As an example, the motor rotates clockwise for a positive input command at control input (TP5). If it is desired that the motor rotate counter clockwise for the same input sign, the procedure outlined below has to be followed.

1- Swap the connection of 'A-MOTOR' and 'C-MOTOR'

2- Swap the connection of 'HALL A' and 'HALL B'

APPENDIX B

B1-Operating procedure for brushless DC servo system

To control the system, the following steps must be adhered to:

Step(1): load the hardware interface with the software programs, use the DOS command line edit

(i) PCL-8181

(ii) TCC -ml file name.c 8181cl.lib

(iii) File name

Step(2): determine the RBF inputs, hidden and outputs nodes number and defined the parameters file names i.e. centres and widths, assuming the learning factor and the sampling time set *a priori*.

Step (3): rune the program and save the desired output signal, actual output signal and the error between the signals in files.

Step(4): check the results.

The structure of the main program is look like the following:

```
# include <    >
.
.
include "userrbf.h"
                                        /* Radial Basis Function */
Initialisation ( )
                {
  example: learning factor =0.02
```

```
        }
Get centres and widths ( )
        {
    defined the centres and widths files name
        }
                        /* Brushless DC Motor */
    {
define PCL-8181 functions

example: pcl818L(13,param);      /* Function. 13: "N" times of D/A output */
         pcl818L(4,param);       /* Function. 4: A/D initialisation */
         pcl818L(5,param);       /* Function. 5: "N" times of A/D trigger */

determine the sampling time ts
    }
                        /* main program */
main ( )
    {
                /* Parameter table */
```

In the main program, determine the pcl-8181 board number, Base I/O address, A/D and D/A converters number etc.

```
example:  param[0]=0;        /*Board number          */
          param[1]=0x300;    /* Base I/O address     */
          param[14]=1;       /* A/D conversion number */

END PROGRAM
    }
```

REFERENCES

REFERENCES

1. Abdulaziz. A. A., (1994), "Neural Based Controller Development for Solving Non-Linear Control Problems", PhD. Thesis , University of Newcastle Upon Tyne, UK.

2. Aleksander I. and H. Morton, (1995), "An Introduction to Neural Computing", London : International Thomson Computer Press.

3. Astrom K. J. and B. Wittenmark, (1989), "Adaptive Control", Addison Wesley Publishing Company.

4. Beale R. and T. Jackson, (1992), "Neural Computing: and Introduction", Bristol; Philadelphia : IOP, 1992, c1990.

5. Bennet S., (1988), "Real-time Computer Control", Prentice Hall International UK LTD.

6. Billings .S.A., and S. Y. Fakhouri, "Theory of Separable Processes with Applications to the Identification of Non-linear Systems", Pro. IEE, Vol. 125,No..9, 1987, pp.1051-1058.

7. Billings S. A. and S. Y. Fakhouri, (1978), "Identification of a Classes of Non-Linear Systems Using Correlation Analysis", Proc. IEE, Vol. 125, No.7, pp.691-697.

8. Billings S. A., S. Y. Fakhouri, (1982), "Identification of Systems Containing Linear Dynamics and Static Non-linear Elements", Automatica, Vol.18, No.1,pp.15-26.

9. Billings S. A., and I. J. Leontaritis, (1981), " Identification of Non-linear Systems using Parameter Estimation Techniques", IEE Conference on Control and its applications, Warwick, March 1981.

10.Billings. S. A, (1980), "Identification of Non-Linear Systems A Survey", IEE Proceeding Vol.127, Part D, No. 6, pp.272-285.

11.Bishop C. M., (1995), "Neural Networks for Pattern Recognition". Oxford : Clarendon Press ; New York : Oxford University Press.

12.Bolton W., (1992), "Control Engineering", Longman Group UK LTD.

13.Broomhead D. S and D. Lowe, (1988), "Multivariable Functional Interpolation and Adaptive Networks", Complex Systems Publications Inc. Vol. 2, pp.321-355.

14.Brown M, and C. Harris, (1994), "Neurofuzzy Adaptive Modelling and Control", New York : Prentice Hall.

15.Chen S. and S. A. Billings, (1989), "Representations of Non-linear Systems: the NARMAX model ", Int. Journal of Control,Vol.49,No. 3, pp.1013-1032.

16.Chen S. and S. A. Billings, (1992), "Neural Networks For Non-linear Dynamic System Modelling and Identification", Int. Journal of Control,Vol.56,No. 2,pp.319-346.

17.Chen S., Billings S. A. and W. Lou, (1989), "Orthogonal Least Squares Methods and Their Application to Non-linear System Identification", Int. Journal of Control, Vol. 50, No. 5, pp.1873-1896.

18.Chen S., C. F. N. Cowan, and P.M. Grant, (1990), " Practical Identification of NARMAX Models Using Radial Basis Function", Int. Journal of Control, Vol. 52, No.6, pp.-1327-1350.

19.Chen S., C. F. N. Cowan, and P.M. Grant, (1991), " Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks", IEEE Transactions on Neural Networks, Vol. 2, No.2, pp.-302-309.

20.Chen S., C. F. N. Cowan, and P.M. Grant, (1991), " Orthogonal Least Squares Learning Algorithm for Training Multi-output Radial Basis Function Networks", IEE Conference publication No.349, Second International Conference on Artificial Neural Networks, pp.336-339.

21.Copp D.G., K. J. Burnham and F. P. Locett, (1997), "Model Identification for Predictive Control of Automatic Air/Fuel Ration Systems", Proc. 12th Int. Conf. on Systems Engineering, ICSE'97 Coventry University, UK, 9-11 September 1997.

22.Cybenko G., (1989), "Approximation by Superpositions of a Sigmoidal Function", Mathematics of Control Signals Systems, Vol. 2, pp.303-314.

23.Doner Hush and Bill G. Horne, (1993), "Progress in Supervised Neural Networks", IEEE Signal Processing Magazine, pp. 8-39.

24.Donoghue F. J., (1977), "A comparison of the Smith Predictor and Optimal Design Approaches for Systems with delay in the Control", IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. IECI-24, No. 1, pp. 109-117.

25.Emanuel Perrcles, (1985), "Motors, Generators, Transformers and Energy", Prentice-Hall.

26.Farolow S. J., (1984), "Self-Organizing Methods in Modelling GMDH Type Algorithms" New York: Marcell Dekker.

27.Farsi M., (1986), " Simplified Self-Tuning Algorithm and Model Reduction for Robot Control", Ph.D. Thesis, University of Newcastle Upon Tyne, UK.

28.Fathala G. and M. Farsi, (1996), "Modelling and Control of Non-linear System Using Radial Basis Function Neural Network Proc. 11th Int. Conf. on Systems Engineering, ICSE'96 University of Nevada, Las Vegas, USA, 9-11 July 1996, pp.25-30.

29.Fathala G. and M. Farsi, (1997), "Modelling and Control of a Brushless DC Motor Using RBF Neural Networks", Proc. 12th Int. Conf. on Systems Engineering, ICSE'97 Coventry University, UK, 9-11 September 1997,pp. 248-251.

30.Franklin G. F., J. D. Powell and M. L. Workman, (1990), "Digital Control Dynamic Systems", Addison Wesley. Pub. Co.

31.Freeman J. F. and D. M. Skapura, (1992), "Neural Networks, Algorithms, Applications, and Programming Techniques", Reading, Mass. : Addison-Wesley.

32.Fukuda. T and T. Shibata, (1992), "Theory and Application of Neural Networks for Industrial Control System", IEEE Trans. on Industrial Electronics, Vol.39, No. 6, pp. 472-489.

33.Gallant S. I., (c1993), "Neural Network Learning and Expert System", Cambridge, Mass. : MIT Press.

34.Garica C. E., D. M. Prett and M. Morari, (1989), "Model Predictive Control: Theory and Practice -a Survey", Autotmatica, Vol. 25. No.3, pp.335-348.

35.Giorsi F and T. Poggio, (1990), "Networks and the Best Approximation Property", Biological. Cybernetics, Vol. 63, pp.169-176.

36.Golub G. H., Van Loan and F. Charles (c1989), "Matrix Computations, second edition", Baltimore, Md. : Johns Hopkins University Press.

37.Goodhart S. G, K. J. Burnham and D. J. G. James, (1994), "Bilinear self-tuning Control of a High Temperature Heat Treatment Plant", IEE Proc. Control Theory Appl. Vol. 141, No.1, January 1994.

38.Gorinevsky D., (1995), "On the Persistency of Excitation in Radial Basis Function Network Identification of Non-linear Systems", IEEE Transactions on Neural Networks, Vol. 6, No. 5, pp 1237-1244.

39.Gorinversky D. and L .A. Feldkamp, (1996), "RBF Network Feedforward Compensation of Load Disturbance in Idle Speed Control", IEEE Control System Magazine, Vol. 16, No. 6, pp.18-27.

40.Harris G. J. and S. A. Billings, (1981), "Self-Tuning and Adaptive Control: Theory and Applications", Peter Peregrinus LTD.

41.Hassun M. H., (1995), "Fundamentals of artificial neural networks", The MIT Press.

42.Haykin S., (1994), "Neural Networks, A Comprehensive Foundation', Macmillan College Publishing Company.

43.Healey M., (1975), "Principle of Automatic Control", The English University Press LTD.

44.Hunt K. J. and D. Sbarbaro, (1991), "Neural Networks for Non-linear Internal Model Control", IEE proceeding part D, Vol. 138, pp.431-438.

45.Hunter I. W. and M. J. Korenberg, (1986), "The Identification of Non-linear Biological Systems: Weiner and Hammerstein Cascade Models", Biological Cybernetics, Vol.55, pp.135-144.

46.Ivakhnenko. A.G., (1991), "Polynomial Theory of Complex Systems", IEEE Trans. ,Vol.SMC-1, No.4, pp.346-378.

47.Jian Tan, Hong Xie, and Yung Cheng Lee, (1995), "Efficient Establishment of a Fuzzy Logic Model for Process Modelling and Control", IEEE Transaction on Semiconductor Manufacturing, Vol.8, No.1, pp 50-61.

48.Karam Z. K., (1988), "Fast Adaptive Control Algorithms and their Application to industrial Robots", PhD Thesis, University of Newcastle Upon Tyne, UK.

49.Kenjo T. and S. Nagamori, (1984), "Permanent Magnet and Brushless DC Motors", Oxford : Clarendon Press.

50. Kiernan L., J. D .Maso and K. Warwick, (1996), "Robust Initialization of Gaussian Radial Basis Function Networks Using Partitioned k-means Clustering", IEE Electronic Letters, Vol. 32, No. 7, pp.671-673.

51.Kou. C. B., (1987), "Automatic Control Systems", Prentice-Hall International, Inc.

52.Kuo L. E. and S. S. Melsheimer, (1994), "Using Genetic Algorithms to estimate the optimum width parameter in Radial Basis Function Networks", Proceedings of the American Control Conference, Baltimore, Maryland, pp. 1368-1371.

53.Lau Clifford, (1992), "Neural Networks: Theoretical Foundations and Analysis", New York : IEEE Press.

54.Leonard J. A. and Kramer M. A., (1991), "Radial Basis Function Networks for Classifying Process Faults" IEEE Control System Magazine, Vol. 11, No. 3, pp. 31-38.

55.Leontaritis I. J. and S. A. Billings, (1985), "Input-Output Paramitric Models for Non-Linear Systems, Part I: Deterministic Non-linear Systems ", International Journal of Control. Vol. 41, No. 2, pp.303-328.

56.Leontaritis.I. J. and S. A. Billings,(1985), "Input-Output Paramitric Models for Non-Linear Systems , Part II: Deterministic Non-Linear Systems ", International Journal of Control. Vol. 41, No. 2, pp. 329-344.

57.Levin A. U. and K. S. Narendra, (1996), " Control of Non-Linear Dynamical Systems Using Neural Networks-Part II: Observability, Identification, and Control", IEEE Transactions on Neural Networks, Vol. 7, No. 1, pp. 30-42.

58.Linkens D. A., (1993), "Parallel Processing for Self Organizing in a Control Systems", In Parallel Processing in a Control Systems Environment (E. Rogers and Y. Li, ed), Prentic Hall.

59.MacQueen J., (1967), "Some Methods for Classification and Analysis of Multivariate Observations", In Proceedings of the Fifth Berkeley Symposium on Mathematical, Statistics and Probability, Vol. 1, pp. 281-297.

60.Mao J. and A. K. Jain, (1996), " A Self-Organizing Network for Hyperellipsoidal Clustering (HEC)", IEEE Transactions on Neural Networks, Vol.7, No.1, pp. 16-29.

61.Meyer C, D. E. Seborg and R. K. Wood, (1976), "A Comparison of the Smith Predictor and Conventional Feedback Control", Chemical Engineering Science, Vol. 31, pp. 775-778.

62.Moody J. and C. J. Darken, (1988), "Learning with Localized Receptive Fields", In Proceedings of the 1988 Connectionist Models Summer School, eds. Touretzky, Hinton and Sejnowski. Morgan-Kaufmann, Publishers. Omohundro, S. 1987. Efficient Algorithms with Neural Network Behaviour. Complex Systems, Vol.1, pp.133-143.

63.Moody J. and C.J.Darken, (1989), "Fast Learning in Networks of Locally-Tuned Processing Units", Neural Computation, Vol.1, pp. 281-294.

64.Musavi M. T., W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, (1992), "On the Training of Radial Basis Function Classifiers", Neural Networks, Vol.5, pp.595-603.

65.Narendra K. S and P. G. Gallman, (1966), "An Iterative Method for The Identification of Non-Linear Systems Using A Hammerstein Model ",IEEE Transactions On Automatic Control, Vol.11, pp.546-550.

66.Narendra K. S. and K. Parthasarathy, (1990), "Identification and Control of Dynamical Systems Using Neural Networks", IEEE Trans. Neural Networks, Vol. 1,No.1, pp.4 - 27.

67.Narendra K. S., (1990), " Adaptive Control Using Neural Networks", MIT Press.

68.Ogata K., (1970), " Modern Control Engineering", Prentice Hall.

69.Owen D. B., (1984), "Self-Organizing Method in Modelling", Statistics Textbook and Monographs.

70.Parr. E. A., (1996), "Control Engineering", Oxford : Butterworth-Heinemann.

71.PCL-818l-Hight Performance, Low Cost Data Acquisition Card, (1995), "User's Manual", Bede Technology LTD, 48 Cuthbert Court, Bede Industrial Estate, Newcastle Upon Tyne, UK.

72.Piegat Andrzej and M. Plucinski, (1995), "Application of the Radial basis Function (RBF) in Modelling and Identification of Linear and Non-linear Systems", Proceeding of the 12th International Conference on System Science, Vol. 1, pp. 266-274.

73.Powell M. J. D., (1992), "The Theory of Radial Basis Function Approximation in 1990", W. Light, ed., Advances in Numerical Analysis, Oxford: Clarendon Press, Vol.2, pp.102-205.

74.Psaltis D., A. Sideris and A. Yamamura, (1988), "A Multilayered Neural Network Controller", IEEE Control Systems Magazine, Vol. 8, No. 2, pp. 17-21.

75.Raiskila P. E. and H. N. Koivo, (1990), "Properties of a Neural Network Controller, ICARV", In Proc. 90 Intl. Conf. on Automat. Robotics, Comput. Vision, pp. 1-5.

76.Renolds J. and L. Tarassenko, (1991), "Isolated word recognition with the Radial Basis Function classifier", IEE Conference publication No.349, Second International Conference on Artificial Neural Networks, pp.345-349.

77.Robbins and Myeos/ Electrol- Craft, (1988), "DC Motors, Speed Control , Servo Systems", An Engineering Handbook.

78.Rumelhart D. E., J. L. McClelland and the PDP Research Group, (1986), "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", The MIT Press.

79.Say M. G. and E.O. Jaylor, (1980), "Direct Current Machines". Pitman Books LMT.

80.Schetzen. M, (1980), " The Voltera and Wiener Theory of Non-linear Systems", John Wiley & Sons, Inc.

81.Shieh Jang-Hang. and Kai-Tai Song, (1995), "An Experimental Study of Learning Control Design for a Non-Linear Servosystem", Dept. of Control Engineering National Chiao Tung University, Taiwan, pp.1-22.

82.Soucek B. and M. Soucek, (1988), " Neural and Massively Parallel Computers: The Sixth Generation", New York : John Wiley and Sons, Inc.

83.Stephen P. B., (1986), "Control System Engineering : modelling and simulation, control theory and microprocessor implementation", Prentice-Hall.

84.Takashi Kenjo, (1994), "Power electronics for the microprocessor age", Oxford University Press.

85.Tanomaru J. and S. Omatu, (1992), "Process Control by On-Line Trained Neural Controllers", IEEE Transactions on Industrial Electronics, Vol.39, No. 6, pp. 511-521.

86.Warwick K., G. Irwin and K.J Hunt, (992), "Neural Networks for Control and Systems", Peter-Peregrinus Ltd UK.

87.Warwick .K and D. Rees, (1986), "Industrial Digital Control Systems", IEE Control Engineering, Series 29, Peter Peregrinus Ltd.

88.Wasserman P. D., (1989), "Neural Computing: Theory and Practice", New York : Van Nostrand Reinhold.

89.Wasserman P. D., (1993), "Advanced Methods in Neural Computing", New York : Van Nostrand Reinhold.

90.Wellstead P. E. and M. B. Zarrop, (1991), "Self-tuning Systems: Control signal processing", John Wiley and Sons, Inc.

91.William H. P., S. A. Teukolsky, W. T. Vetterling and B. P. Flanery (1992), "Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition", :Cambridge [Cambridgeshire] ; New York : Cambridge University Press.

92.Yamda T. and T. Yabuta, (1992), "Neural Network Controller Using Autotuning Method for Non-Linear Functions", IEEE Trans. on Neural Networks, Vol.3, No. 4, pp. 595-601.

93.Ye X. and N. K. Loh, (1993), "Dynamic System Identification Network", Proceeding of the American Control Conference, San Francisco, pp.2912-2916.

94.Zalzala A. M. S. and A. S. Morris, (1995), "Neural Networks for Robotics Control Theory and Applications", New York : Ellis Horwood.

95.Zhu. Q. M., K. Warick and J. L. Douce, (1991), "Adaptive General Predictive Controller for Non-linear Systems", IEE proc. D, Vol. 138, No.1, pp.33-40.

96. Zurada J. M. (1992), " Introduction to Neural Systems", West Publishing Company.