APPROXIMATION OF THE INVERSE KINEMATICS OF A ROBOTIC MANIPULATOR USING A NEURAL NETWORK

BACH HOANG DINH

Thesis submitted for the Degree of Doctor of Philosophy

Heriot-Watt University

Department of Electrical, Electronic

and Computer Engineering

July 2009

Abstract

A fundamental property of a robotic manipulator system is that it is capable of accurately following complex position trajectories in three-dimensional space. An essential component of the robotic control system is the solution of the inverse kinematics problem which allows determination of the joint angle trajectories from the desired trajectory in the Cartesian space. There are several traditional methods based on the known geometry of robotic manipulators to solve the inverse kinematics problem. These methods can become impractical in a robot-vision control system where the environmental parameters can alter. Artificial neural networks with their inherent learning ability can approximate the inverse kinematics function and do not require any knowledge of the manipulator geometry.

This thesis concentrates on developing a practical solution using a radial basis function network to approximate the inverse kinematics of a robot manipulator. This approach is distinct from existing approaches as the centres of the hidden-layer units are regularly distributed in the workspace, constrained training data is used and the training phase is performed using either the strict interpolation or the least mean square algorithms. An online retraining approach is also proposed to modify the network function approximation to cope with the situation where the initial training and application environments are different. Simulation results for two and three-link manipulators verify the approach.

A novel real-time visual measurement system, based on a video camera and image processing software, has been developed to measure the position of the robotic manipulator in the three-dimensional workspace. Practical experiments have been performed with a Mitsubishi PA10-6CE manipulator and this visual measurement system. The performance of the radial basis function network is analysed for the manipulator operating in two and three-dimensional space and the practical results are compared to the simulation results. Advantages and disadvantages of the proposed approach are discussed.

Acknowledgements

Firstly, I would like to thank my supervisors, Dr. M. W. Dunnigan and Dr. D. S. Reay, for their kind supports in my research time. I would particularly like to express my deepest gratitude to my main supervisor, Dr. M.W. Dunnigan for his enthusiasm, patience, guidance and support from the initial to the final level of the research and the writing of this thesis.

Acknowledgement is also given to other members of the Intelligent Robotic Systems Laboratory and staff members in the department. In particular, I would like to thank my friends, Dr. J. T. Hatleskog and Dr. L. C. Tran for their kind helps.

I would also like to thank my parents, my parents-in-law and all members of my family for their support and encouragement throughout the years of my Ph.D study. I am heartily thankful to my wife, Vy, and my son, Tin, for their accompanying to make a great time in Edinburgh.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the thesis.

ACADEMIC REGISTRY
Research Thesis Submission

Name*:*	BACH HOANG DINH		
School/PGI:	EPS		
Version: *(i.e. First, Resubmission, Final)*	Final	Degree Sought (Award **and** Subject area)	Doctor of Philosophy Electrical Engineering

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) the thesis embodies the results of my own work and has been composed by myself
2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3) the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

* *Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate*:*		Date:	

Submission

Submitted By *(name in capitals):*	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in Academic Registry

Received in the Academic Registry by *(name in capitals):*			
Method of Submission *(Handed in to Academic Registry; posted through internal/external mail):*			
*E-thesis Submitted (**mandatory for final theses from January 2009)***			
Signature:		Date:	

Please note this form should bound into the submitted thesis.

Updated February 2008, November 2008, February 2009

Table of Contents

CHAPTER 1
INTRODUCTION

1.1 Introduction

Robots are a vital part of modern manufacturing industries with their inherent capability of executing complex tasks accurately and reliably. Robots are also used extensively in areas where it is hazardous for humans. Examples of this are underwater intervention in oil and gas exploration, nuclear plant decommissioning and space exploration. One of the most popular robots are anthropomorphic robotic manipulators which are used extensively in manufacturing industry. A robotic manipulator is composed of several links connected together (usually in series) through joints to form an arm and/or wrist [1.1]. Figure 1.1 presents the general structure of a series manipulator with revolute joints.

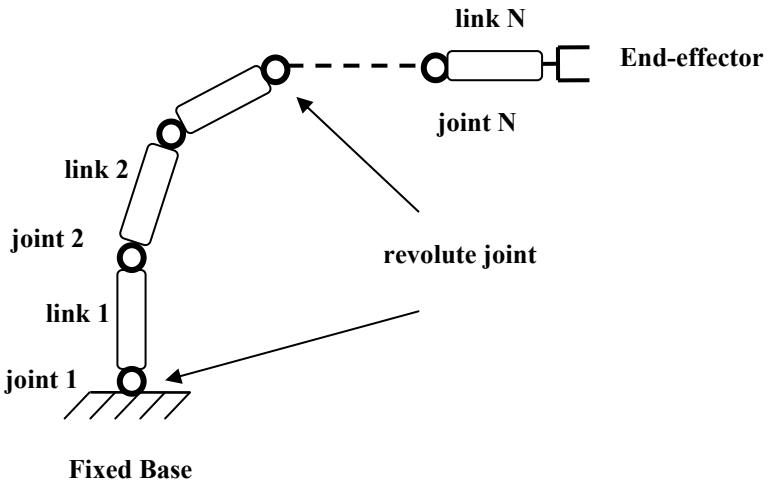


Figure 1.1- The general structure of a manipulator with revolute joints.

When the joints move, the links also move according to the action of the joints which are either revolute or prismatic. As a result, the end-effector attached at the tip of the manipulator can be driven to any location in its workspace. The values of joint angles determine the current configuration of the arm which places the end-effector at a specific location in the environment [1.2]. The operational tasks of the manipulator are

usually planned in the workspace (the Cartesian coordinates), whereas control commands are directly performed in the joint space. One of the principal roles when designing a robotic system is how to determine the inverse kinematics transformation from the workspace to the joint space to set the references for the joint controllers (dynamic control). Figure 1.2 shows the general operational tasks of a robotic system.

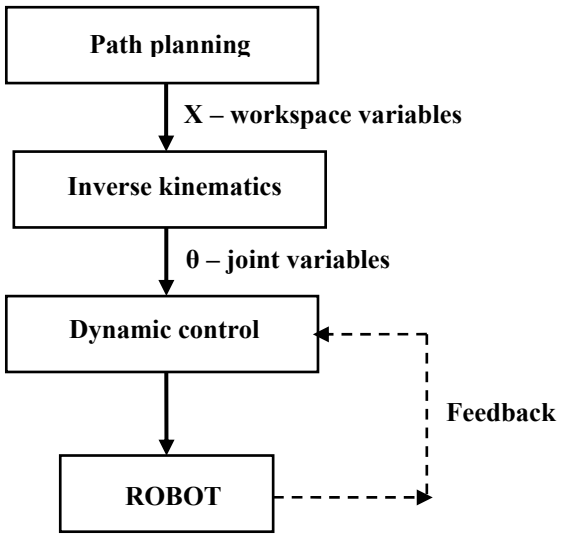
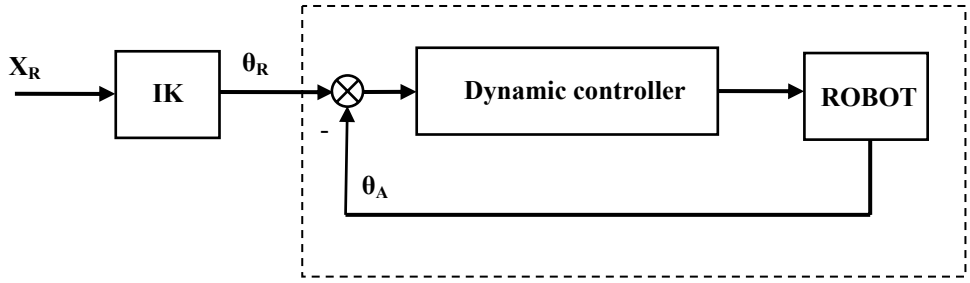


Figure 1.2 - The operational tasks of a robotic system.

This thesis focuses on solving the inverse kinematics problem which is used to transform a position of the end-effector in the Cartesian space to a set of joint angles. In Figure 1.2 the inverse kinematics block operates as a transformation function to provide the joint angle references for the dynamic control system which operates as a closed-loop controller in the joint space. Figure 1.3 shows the general control diagram using the inverse kinematics transformation to implement a position control task in the workspace. The combination of the dynamic controller and the robot itself can be modelled as the forward kinematics of the robotic system.

There are several different procedures available, which are based on the known geometry of the manipulator, to solve the inverse kinematics problem. These include the geometric, algebraic and numerical iterative methods [1.3], [1.4]. However, these solutions become more difficult, or impractical, when the manipulator geometry cannot be determined exactly. Therefore, this poses a question as to whether any alternative solution to determine the inverse kinematics transformation exists if the geometry of the

manipulator is unknown. A possible approach is to use neural networks to learn the inverse kinematics transformation.



IK – inverse kinematics transformation of the robot .

Dynamic controller – standard PID controller for joint angle control.

$\mathbf{X_R}$ – desired position in the world space.

$\boldsymbol{\theta_R}$ – desired joint angles setting for dynamic control system.

$\boldsymbol{\theta_A}$ – actual joint angles of the robot.

Figure 1.3 - A control approach using the inverse kinematics transformation to implement a position control task in the workspace.

Neural networks with their inherent learning ability have been widely applied in many fields of robotic control. They are seen as an intelligent control scheme because of their learning ability, as well as having the flexibility to cope with the uncertain and unstructured working conditions. The aim of this research is to develop a solution using neural networks to solve the inverse kinematics transformation of a robotic manipulator with unknown geometry. This is implemented though two sequential phases, training with data collected from the robotic system and then operating as a transformation function. Once trained, its generalisation ability allows responses to be produced for untrained data according to stored knowledge. The network performance is related to several factors, such as the network architecture, the learning method and training data. This performance is not entirely controllable and sometimes it can be inaccurate. The question of how to improve the performance of a neural network to solve the inverse kinematics problem is an interesting and important topic, especially for practical applications.

The main objective of this thesis is to develop a practical solution using a neural network to approximate the inverse kinematics of a robotic manipulator. The practical system includes a simple computer vision measurement system to acquire the position of the robot's end-effector. The system is shown in Figure 1.4.

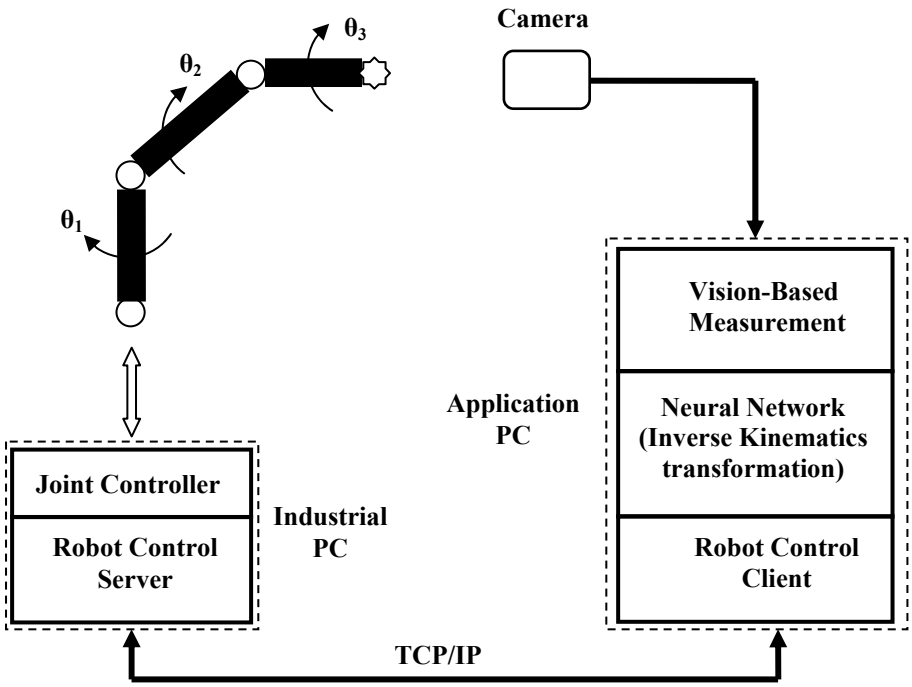


Figure 1.4 - Block diagram of the system used for practical work.

The vision-based measurement system, consisting of one camera and image processing software, determines the position of the end-effector in real-time. Recently, vision-based measurements have been used more extensively in the robotic field because of their benefits, such as accuracy, low cost and portability [1.5]. Any change in the structure of this system, either a displacement of the camera location or a change in the location of the manipulator base leads to a change in the configuration of the robotic system. Therefore, it is essential to develop a flexible and adaptable inverse kinematics solution so that it can deal with any disturbances affecting the structure of this system. A solution for this task is a neural network approach.

1.2 Manipulator kinematics

In robot kinematics there are two important problems, forward and inverse kinematics. Forward kinematics can be regarded as a one-to-one mapping from the joint space to the Cartesian coordinate space (workspace). From a set of joint angles, forward kinematics determines the corresponding location (position and orientation) of the end-effector. This problem can be solved by multiplying together the 4x4 homogenous transformation matrices using the Denavit & Hartenbergh representation for the manipulator [1.3], [1.4]. Inverse kinematics is used to compute the corresponding joint angles from the location of the end-effector in Cartesian space. Inverse kinematics is a more difficult problem than forward kinematics because of its multi-mapping characteristics.

There have been many techniques proposed to solve the inverse kinematics problem, e.g., the geometric, algebraic and numerical iterative methods. The geometric approach is useful for simple manipulators with revolute joints. It is based on the definitions of the link coordinate systems and human arm geometry which allows various arm configurations to be identified for the inverse kinematics problem [1.3]. The matrix algebraic approach is an inverse transform technique to obtain joint-angle solutions directly from the homogenous transformation matrices (forward kinematics representations) [1.6]. Both these approaches are regarded as analytical solutions which determine the exact mathematical formulae for the inverse kinematics problem. However, these are complicated and require intuition to select an appropriate case from the several possible solutions for a particular manipulator [1.4]. In contrast, more general approaches were developed using a numerical iterative solution based on the Jacobian matrix [1.4], [1.6]. A relationship between joint velocities and hand motion velocity is first derived from directly inverting the Jacobian matrix and then the inverse kinematics solution for joint angles and hand position follows. This solution can be applied for most of the common manipulator configurations in industry. However, it does not always guarantee to produce all the possible inverse kinematic solutions and involves significant computation. Furthermore, all the mentioned methods are termed traditional approaches because they have been developed from the geometric parameters of the manipulators. If the geometry cannot be exactly specified, these traditional approaches become more difficult or even impractical. For these reasons, it is of interest

to pursue other non-traditional approaches and the research described in this thesis proposes an alternative solution based on neural networks.

1.3 Approximating the inverse kinematics of robotic manipulators by neural networks

Artificial neural networks are a simple imitation of human brain behaviours, such as learning and responding to any stimuli from the environment. Due to its learning ability a neural network will establish its knowledge through updating the synaptic weights between interconnections of the network's neurons. This learning is implemented with training examples (sets of inputs and target outputs) which are collected from the desired process. Thus, a neural network can learn and approximate any complex function without any prior knowledge of that process [1.7].

Various neural networks have been used to solve the inverse kinematics problem. They include the multi layer perceptron network (MLPN), cerebellar model articulation controller (CMAC) and radial basis function network (RBFN) [1.7]. However, the radial basis function network seems to be more suitable for the inverse kinematics problem because its hidden-layer structure parameters (centres of radial basis functions) could be optimally selected from training data and the learning process is simple using the least squares approach [1.7], [1.8]. In this thesis, a novel approach using a radial basis function network with regularly-spaced position centres to approximate the inverse kinematics transformation of a manipulator is proposed and investigated. This solution has then been applied for position control of the robotic system as shown in Figure 1.4. Both computer simulation and practical work have demonstrated that the proposed approach is effective in solving the inverse kinematics problem.

1.4 Real-time visual measurement system

To measure the location of a movable object in the world space, distance sensors have normally been used including sonic or optical types. However, a vision-based measurement system is more convenient because it can measure absolute space coordinates of the object with respect to the camera base, instead of only the distance from the object to a reference point. In recent years there have been many solutions to

develop 3-D visual measurement systems using video cameras. The principle of these methods is to estimate the position and orientation of a known geometric object in the world space based on the object's image (in the image plane). This assumes that image coordinates of the object in the image plane can be determined and the camera intrinsic parameters (e.g., resolution, focal length, distortion coefficients) are known as well. This is called the camera calibration method [1.9] and only uses one camera accompanied with image processing software. This research field has made significant progress in aspects of efficiency, accuracy and reduction in cost. As a result, using cameras to supervise and control robots has become realistic and is widely applied in robotic systems [1.5].

This thesis presents a real-time visual measurement system to estimate the three-dimensional position of a manipulator in the workspace. It consists of a standard video camera mounted on a fixed pole to measure the position of a light sample board attached to the manipulator end-effector. The software is programmed based on the Intel Open Source Computer Vision Library, in C++, and uses a Graphical User Interface (GUI) to make this visual tool more convenient for practical applications.

1.5 Structure of the thesis

The thesis is presented in eight chapters.

Chapter 1 – Introduction. This chapter briefly introduces the problem background and structure of the thesis. The main topics include the inverse kinematics transformation of robotic manipulators, neural networks for approximating inverse kinematics and visual measurement. The objectives of the thesis are stated.

Chapter 2 – Traditional Approaches for The Determination of The Inverse Kinematics of Robotic Manipulators. This chapter introduces the background theory concerning the inverse kinematics problem of robotic manipulators. Various traditional solutions, such as the algebraic and geometric methods are discussed in detail. Advantages and disadvantages of each solution are presented, especially in terms of practical applications.

Chapter 3 – Background of Neural Networks for Inverse Kinematics Approximation of Robotic Manipulators. This chapter is the literature review concerning using neural networks to approximate the inverse kinematics transformation of robotic manipulators. It includes two of the popular neural network architectures, MLPN and RBFN. The fundamental aspects of the learning process and performance of each network are described to highlight the advantages and drawbacks of existing approaches. Finally, the reasons why the radial basis function network is adopted and the purpose of the proposed approach are stated.

Chapter 4 – Inverse Kinematics Approximation Using a Radial Basis Function Network. This chapter concentrates on investigating the possibility of using a radial basis function network to approximate the inverse kinematics transformation. A novel idea is proposed using hidden-layer centres which are regularly-spaced positions in the workspace and using constrained training data whose inputs are collected approximately around centre positions. Various simulations in MATLAB demonstrate the network performance and factors that affect the network performance are investigated.

Chapter 5 – Online Training to Modify The Inverse Kinematics Approximation. This chapter describes a solution to modify the inverse kinematics approximation using an additional online training process. It uses the delta rule to update the linear weights of the network which have been trained already with incorrect data. This online retraining can be applied to deal with the difficulty in collecting constrained data and to re-correct for operational errors due to variations in the visual measurement system. The simulation results are discussed in detail.

Chapter 6 – Development of a Three-dimensional Positional Measurement System. This chapter presents a solution using a webcam and image processing software to measure the position of a manipulator in a 3-D workspace. The background of camera calibration methods is described to explain the procedures in which the intrinsic and extrinsic camera parameters are estimated. The concept of developing a 3-D visual measurement system to determine the position of the manipulator end-effector is presented.

Chapter 7 – Practical Investigation of RBFN Performance. This chapter describes the set-up of the experimental system which includes a PA10-6CE manipulator, a visual

measurement system and application programmes. Two experimental systems using two different schemes are described for the two-dimensional and three-dimensional workspaces. The experimental results of the RBFN are then presented. A solution using online retraining to modify the RBFN to cope with a change in the structure of the visual measurement system is presented in detail. Conclusions about the effectiveness for the practical application are drawn.

Chapter 8 – Conclusions. This chapter summarises the author's main contributions including the benefits of this work. Finally, suggestions for future work are presented.

1.6 References

[1.1] C. Bergren, *Anatomy of A Robot*. McGraw Hill, 2003.

[1.2] J. Angeles, *Fundamental of Robotic Mechanical System: Theory, Methods and Algorithms – Second Editor*. Springer, 2003.

[1.3] K. S. Fu, R. C. Gonzalez and C. S. G. Lee, *Robotics – Control, Sensing, Vision and Intelligence*. McGraw Hill, 1987.

[1.4] W. Khalil and E. Dombre, *Modelling, Identification & Control of Robots*. Hermes Penton Ltd., 2002.

[1.5] S. Florczyk, *Robot Vision Video-based Indoor Exploration with Autonomous and Mobile Robots*. Wiley-VCH, 2005.

[1.6] A. J. Koivo, *Fundamentals for Control of Robotic Manipulators*. John Wiley & Sons, 1989.

[1.7] G. W. Irwin, K. Warwick and K. J. Hunt, *Neural Network Applications in Control*. IEE Control Engineering Series 53, 1995.

[1.8] S. Haykin, *Neural Networks a Comprehensive Foundation – Second Edition*. Prentice Hall, 1999.

[1.9] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall Inc., 1998.

CHAPTER 2
TRADITIONAL APPROACHES FOR THE DETERMINATION OF THE INVERSE KINEMATICS OF A MANIPULATOR

2.1 Introduction

The kinematics of a manipulator deals with the geometry of manipulator motion with respect to a fixed reference coordinate system as a function of time without regards to the forces or moments that cause this motion [2.1]. Based on the spatial configuration of a manipulator, kinematics equations are established to represent relationships between the joint variable space and location of the end-effector in the world space (Cartesian coordinate system). Inverse kinematics is used to compute the corresponding joint angles from a given location of the end-effector. It is a nonlinear function where more than one set of joint angles for a desired location of the end-effector can exist, i.e., there are multiple solutions. Sometimes no solution can be found due to particular configurations of the manipulator, such as singular and degenerate cases [2.3]. Thus, this is a complex problem dependent on many factors such as complexity of configuration, specific operating conditions and constraints of joint variables.

There are many inverse kinematics techniques based on either analytical or numerical methods. These techniques are called traditional approaches, because they require knowledge of the manipulator's configuration parameters. An analytical approach attempts to produce an exact solution mathematically by directly inverting the forward kinematics equations. However, it is only possible for some relatively simple geometric manipulators. A numerical solution uses approximate optimal techniques to solve the inverse kinematics of a general manipulator. It can mutually transform all motion characteristics, such as position, velocity and acceleration, from the Cartesian space to the joint space by iterative computation based on the inversion of the Jacobian matrix [2.3]-[2.5]. The principle of this approach is taken from the relationship between the motion velocity (translation and rotation) of the end-effector in the Cartesian space and the joint angle velocities to approximately calculate corresponding joint angles from a given location of the end-effector. However, this involves an iterative numerical algorithm with a high computational demand and does not guarantee convergence.

This chapter firstly describes the general structure of a robotic manipulator. The Denavit & Hartenbergh procedure to systematically establish the kinematic equations of a manipulator is then presented. Two analytical approaches, geometric and algebraic, are presented to describe traditional algorithms to solve the inverse kinematics of robotic manipulators. Each approach is applied to a three-link manipulator formed from the structure of a Mitsubishi PA10-6CE used later in the experimental work [2.6]. Following this, the beneficial reasons of adopting an alternative solution using neural networks for the determination of the inverse kinematics are stated.

2.2 Kinematic equations of a robotic manipulator

2.2.1 General structure of a robotic manipulator

A manipulator is made of several links, connected together (usually in series) by the joints, to form an arm and/or wrist. A specific location (position and orientation) of the end-effector, attached to a manipulator, will be completely determined by six independent coordinates related to 6 Degrees-of-Freedom (D.O.F): three for position and three for orientation [2.1]. The general configuration of a manipulator can be split into two functional groups, the arm and wrist. The first three joints are commonly designed in order to perform gross motion of the end-effector as an arm and the remaining joints are used to accomplish orientation as a wrist. Thus, according to the first three joint types (revolute or prismatic) and how they combine together, five distinct and non-redundant structures can be described: Articulated (RRR), Spherical (RRP), SCARA (RRP), Cylindrical (RPP), and Cartesian (PPP) (Figures 2.1 to 2.5) [2.3], [2.7].

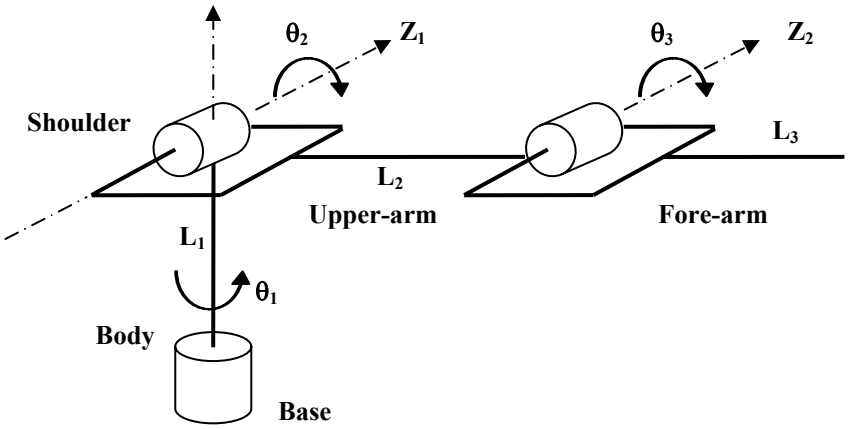


Figure 2.1- The articulated configuration (elbow manipulator) (RRR).

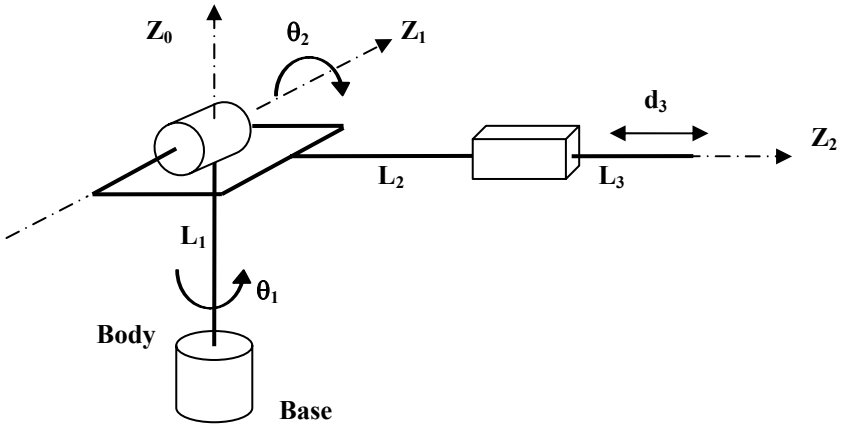


Figure 2.2 - The spherical configuration (RRP).

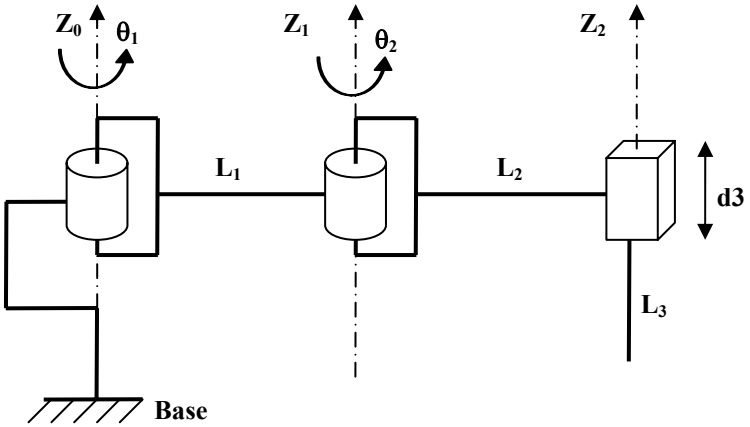


Figure 2.3 - The SCARA configuration (RRP).

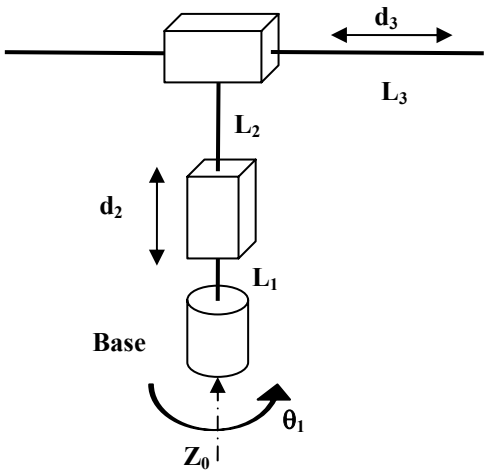


Figure 2.4 - The cylindrical configuration (RPP).

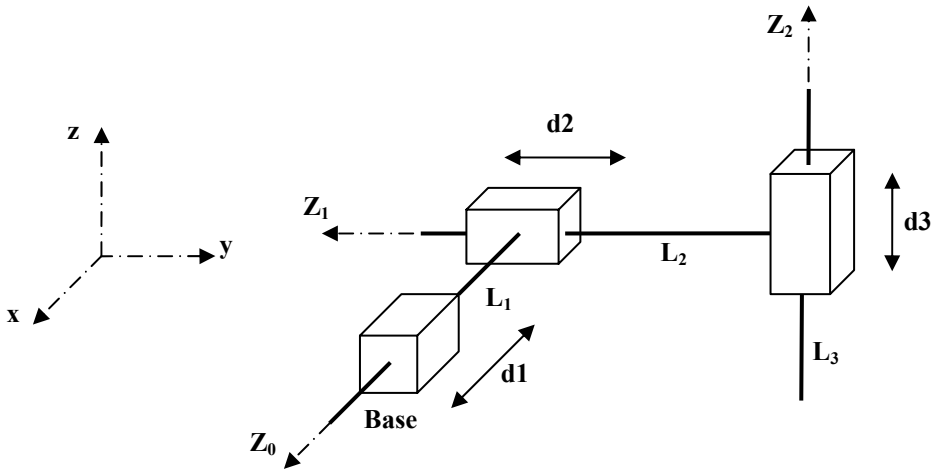


Figure 2.5 - The Cartesian configuration (PPP).

The wrist of a manipulator refers to the joints in the kinematic chains between the arm and hand. Almost all the joints of the wrist are revolute. The wrist is used to achieve the desired orientation of the end-effector. The most common type of wrist is the spherical configuration as shown in Figure 2.6. In practice, the spherical wrist greatly simplifies the kinematic analysis, effectively allowing decoupling of the position and orientation of an object [2.7].

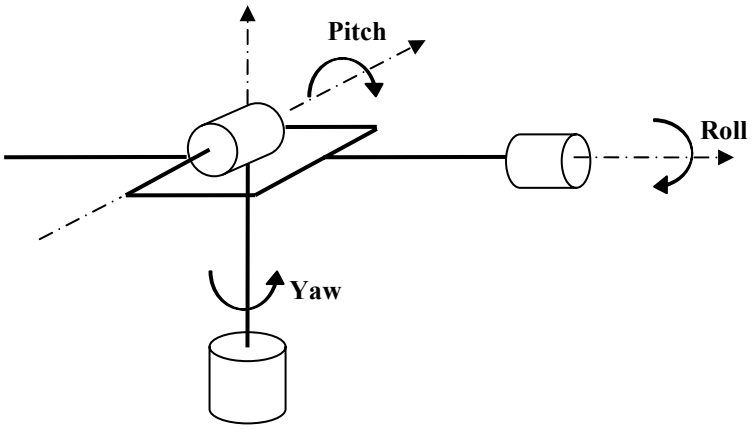


Figure 2.6 - The spherical wrist with Pitch, Yaw and Roll.

2.2.2 Denavit & Hartenbergh representation

The kinematic equations use matrix algebra to build relationship functions between the joint variables and the world coordinate location of the end-effector (position and orientation) based on the spatial configuration of a particular manipulator. A systematic and generalised approach to describe the kinematic equations of a serial link manipulator was proposed by Denavit & Hartenbergh (D-H) [2.1]. This expresses the rotation and translation of the coordinate frame attached to a link with respect to another reference coordinate frame by a homogenous transformation matrix. It is a 4x4 matrix in which the sub-matrix, a 3x3 rotation matrix, is used to describe the rotational operation. A 3x1 vector is used to describe the translational operation of the coordinate frame attached to a link with respect to the reference frame. The homogenous coordinate transformation matrix $\boldsymbol{A}_{i-1}^{i}$ can be written as

$$\boldsymbol{A}_{i-1}^{i} = \begin{bmatrix} \boldsymbol{R}_{3\times 3} & \boldsymbol{P}_{3\times 1} \\ 0 & 1 \end{bmatrix}. \tag{2.1}$$

The basic rules for establishing an orthogonal coordinate frame for each link and determining the geometric parameters of a serial manipulator are presented in [2.1]-[2.3] as a systematically closed-form procedure. Based on this algorithm, the D-H

coordinate systems can be established for all links and the homogenous transformation matrices $\boldsymbol{A}_{i-1}^{i}$ between adjacent coordinate frames can be expressed easily by

$$\boldsymbol{A}_{i-1}^{i} = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where a_i, d_i, α_i, θ_i are the geometric parameters of the relative i^{th} joint as shown in Figure 2.7 [2.1]. They can be defined as:

- α_i is the angle between z_{i-1} and z_i about x_i.
- d_i is the distance between origin O_{i-1} and the intersection of the z_{i-1} axis with the x_i axis along z_{i-1} (or the distance between x_{i-1} and x_i if they are parallel).
- θ_i is the angle between x_{i-1} and x_i about z_{i-1}.
- a_i is the distance between origin O_i and the intersection of the z_{i-1} axis with the x_i axis along x_i (or the distance between z_{i-1} and z_i if they are parallel).

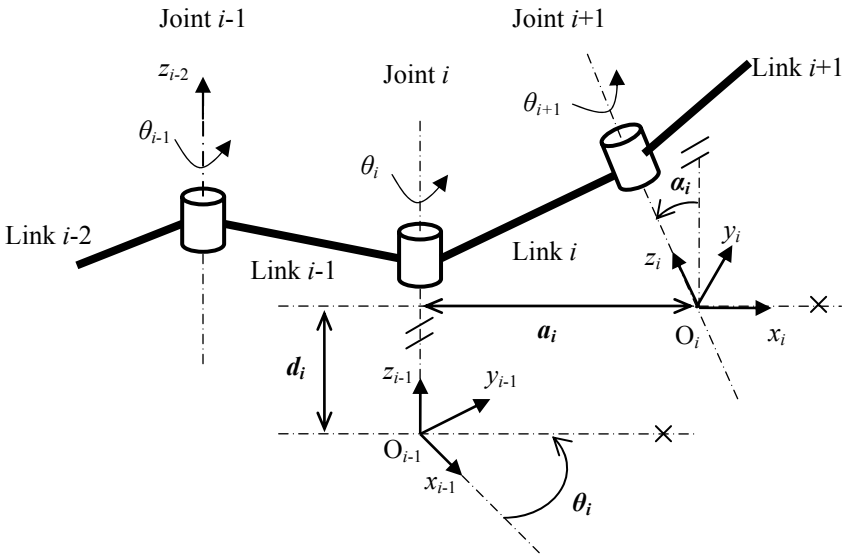


Figure 2.7- Structure kinematic parameters for a general link *i*.

If a position vector $\boldsymbol{p}_i$ is given in the i^{th} coordinate frame, then it can be expressed in the $(i\text{-}1)^{\text{th}}$ coordinate system as the vector $\boldsymbol{p}_{i\text{-}1}$ by

$$\boldsymbol{p}_{i-1} = \boldsymbol{A}_{i-1}^{i}\boldsymbol{p}_i \,. \tag{2.3}$$

Thus, the coordinate transformation matrix maps a position vector expressed in homogenous coordinates from one coordinate system to another coordinate system. As a result, through a sequential transformation, achieved by multiplying a series of the coordinate transformation matrices, the coordinate frame attached to the end-effector can be transformed and expressed in the base coordinate system. This systematic transformation is shown in Figure 2.8. Generally, the homogenous transformation matrix from the n^{th} coordinate frame to the base coordinate frame can be determined by multiplying $\boldsymbol{A}_{i-1}^{i}$ (i = 1, 2,…, n) together in sequence, such as

$$\boldsymbol{T}_0^n = \boldsymbol{A}_0^1\boldsymbol{A}_1^2 \ldots \boldsymbol{A}_{n-1}^n \,. \tag{2.4}$$

Consequently, if a vector $\boldsymbol{p}_n$ is known in the n^{th} coordinate frame, it can be determined with respect to base coordinate system by

$$\boldsymbol{p}_0 = \boldsymbol{T}_0^n\boldsymbol{p}_n \,. \tag{2.5}$$

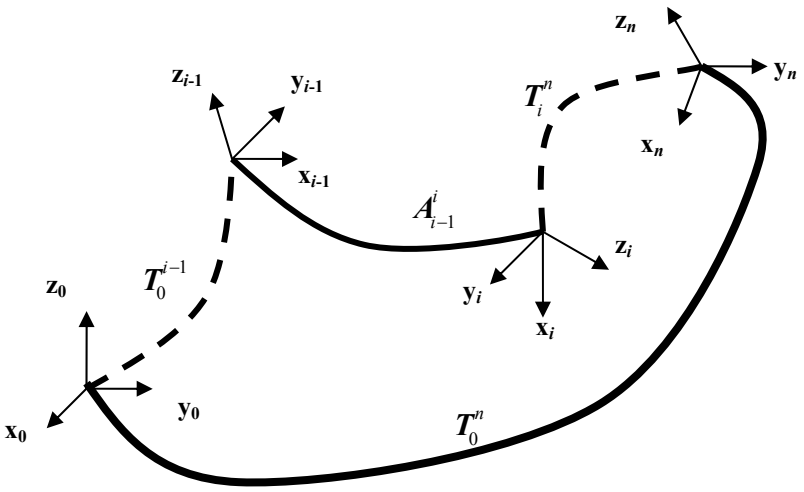


Figure 2.8-Coordinate transformation from the end-effector to a base coordinate system.

2.2.3 Kinematics equations of a three-link manipulator

Applying the D-H representation, the kinematics equations of a three-link manipulator illustrated in Figure 2.8 can be established as follows.

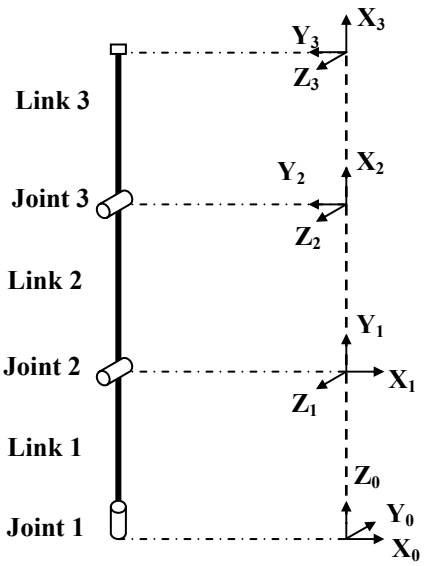


Figure 2.9 – Link coordinate systems of a three-link PA10 manipulator.

Using the D-H procedure, the coordinate frames are defined and the structural geometric parameters of the three-link manipulator are presented in Table 2.1. This structure is formed from the first three links of a Mitsubishi PA10-6CE manipulator [2.6].

Link	d_i **(mm)**	a_i **(mm)**	α_i **(degree)**	θ_i **(degree)**
1	317	0	90^0	θ_1
2	0	450	0	$\theta_2 + 90^0$
3	0	550	0	θ_3

Table 2.1- Structural geometric parameters of the three-link manipulator.

Therefore, the coordinate transformation matrices $\boldsymbol{A}_{i-1}^{i}$ (i = 1, 2, 3) relating the coordinates of the i^{th} frame to those of the $(i\text{-}1)^{\text{th}}$ frame can be written using equation (2.2) as follows

$$\boldsymbol{A}_0^1 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$\boldsymbol{A}_1^2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

$$\boldsymbol{A}_2^3 = \begin{bmatrix} C_3 & -S_3 & 0 & a_3C_3 \\ S_3 & C_3 & 0 & a_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

where $S_i = \sin \theta_i$ and $C_i = \cos \theta_i$.

The homogenous transformation matrix $\boldsymbol{T}_0^3$ from the coordinate frame attached to the end-effector to the base coordinate frame can represented as

$$\boldsymbol{T}_0^3 = \boldsymbol{A}_0^1\boldsymbol{A}_1^2\boldsymbol{A}_2^3 = \begin{bmatrix} C_1C_{23} & -C_1S_{23} & S_1 & C_1(a_3C_{23} + a_2C_2) \\ S_1C_{23} & -S_1S_{23} & -C_1 & S_1(a_3C_{23} + a_2C_2) \\ S_{23} & C_{23} & 0 & a_3S_{23} + a_2S_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

where

$C_{23} = \cos (\theta_2 + \theta_3) = C_2C_3 - S_2S_3$, and

$S_{23} = \sin (\theta_2 + \theta_3) = S_2C_3 + C_2S_3$.

Based on this transformation matrix, the forward kinematics can determine an exact location of the manipulator in the workspace according to the value of each joint angle variable.

2.3 Geometric approach

2.3.1 Principle of geometric approach

Based on the specific structural geometry of a manipulator this approach identifies inverse kinematics solutions for a revolute manipulator which is classified according to the first three joints and the remaining joints. The basic idea is that by analysing the trigonometry of the arm for a given position vector, the first three joint angles can be determined. The remaining joint angles can be calculated according to the orientation matrix and the known values of previous joint angles. For example, Lee [2.8] developed a geometric approach to solve the inverse kinematics for a Puma 560 manipulator. This algorithm first used the position vector pointing from the shoulder to the wrist that was determined from the homogenous matrix $\boldsymbol{T}_0^3$ to derive the inverse kinematics solution for the first three joints. The last three joints were sequentially calculated using previously known joint angles and the orientation sub-matrix of the corresponding matrices $\boldsymbol{T}_0^i$ (i = 4, 5, 6). In [2.6] a more general modified closed-form procedure was proposed for the inverse kinematics of a Puma 560. The position vector was first projected onto a x_i - y_i plane (i = 0, 1, 2) to derive the inverse kinematics solution for each joint (by solving trigonometric equations). The last three joint angles could then be determined from the previous known joint angles, the given orientation matrices and the trigonometric equations which were derived from projecting the link coordinate frames onto the following x_i - y_i planes (i = 3, 4, 5). These derived solutions were different for each specific arm configuration: LEFT or RIGHT hand and UP or DOWN elbow. Configuration indicators were therefore required to determine the appropriate result. The arm-configuration indicators can be predefined and chosen by the user according to the specific application. This approach can be generalised and extended to most industrial manipulators with revolute joints.

2.3.2 The geometric approach for a three-link manipulator

In this section, the geometric approach is applied to find the inverse kinematics of the three-link manipulator shown in Figure 2.9. The procedure is implemented as follows.

A point P representing a given position of the end-effector, which is the origin of the last coordinate frame with respect to the base coordinate system, can be expressed by

$$\mathrm{P} = (P_x, P_y, P_z)^T . \tag{2.10}$$

This corresponds to the translation vector of the homogenous coordinate transformation matrix $\boldsymbol{T}_0^3$ in equation (2.9)

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} C_1(a_3C_{23} + a_2C_2) \\ S_1(a_3C_{23} + a_2C_2) \\ a_3S_{23} + a_2S_2 + d_1 \end{bmatrix}. \tag{2.11}$$

By projecting the point P onto the x_0-y_0 plane as shown in Figure 2.10, the trigonometric equations to solve θ_1 can be obtained as:

$$P_x = \mathrm{OP}' \cos\theta_1 \tag{2.12}$$

$$P_y = \mathrm{OP}' \sin\theta_1 \tag{2.13}$$

$$\mathrm{OP}' = \sqrt{P_x^2 + P_y^2} . \tag{2.14}$$

Hence, θ_1 is calculated by

$$\theta_1 = \tan^{-1}\left(\frac{P_y}{P_x}\right). \tag{2.15}$$

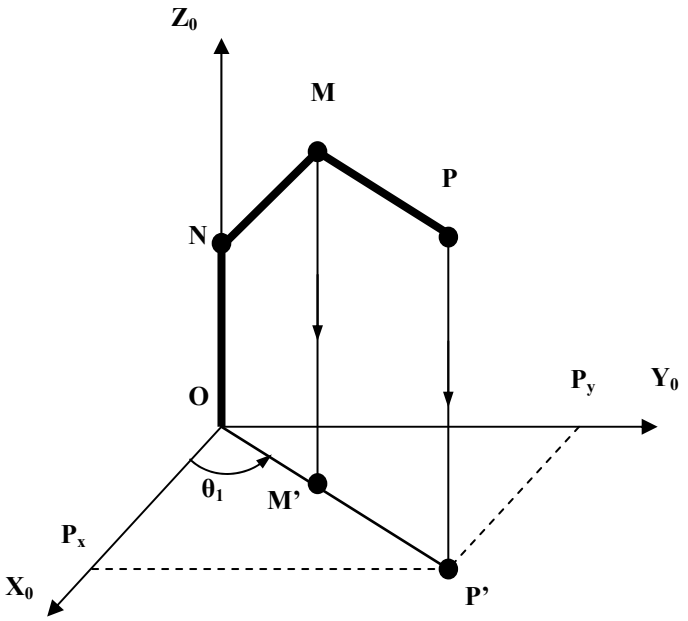


Figure 2.10 – Projecting the point P onto the x_0 - y_0 plane.

Similarly, as shown in Figure 2.11, by projecting the point P onto the x_1 - y_1 plane the geometric relationships for links 2 and 3 are written by

$$\mathrm{NP''} = \mathrm{NM''} + \mathrm{M''P''} = -a_2 \sin\theta_2 - a_3 \sin(\theta_2 + \theta_3) \tag{2.16}$$

$$\mathrm{NP'''} = \mathrm{NM'''} + M'''P''' = a_2 \cos\theta_2 + a_3 \cos(\theta_2 + \theta_3) \tag{2.17}$$

where

$$\mathrm{NP''} = \mathrm{OP'} = \sqrt{P_x^2 + P_y^2} \tag{2.18}$$

$$\mathrm{NP'''} = P_z - d_1 . \tag{2.19}$$

Squaring and summing both sides of equations (2.16) and (2.17), the solution to determine θ_3 then can be expressed as

$$\cos(\theta_3) = \frac{(\mathrm{NP''})^2 + (\mathrm{NP'''})^2 - a_2^2 - a_3^2}{2a_2a_3} = K \tag{2.20}$$

and

$$\sin(\theta_3) = \pm\sqrt{(1-K)^2}\ . \tag{2.21}$$

Therefore, θ_3 can finally be calculated as

$$\theta_3 = \tan^{-1}\left[\pm\frac{\sqrt{1-K^2}}{K}\right] \tag{2.22}$$

where

$$K = \frac{P_x^2 + P_y^2 + (P_z - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3}. \tag{2.23}$$

To derive the solution for the joint angle θ_2, equations (2.16) and (2.17) can be rewritten as

$$a_2 \sin\theta_2 = -\mathrm{NP''} - a_3 \sin(\theta_2 + \theta_3) \tag{2.24}$$

$$a_2 \cos\theta_2 = \mathrm{NP'''} - a_3 \cos(\theta_2 + \theta_3)\ . \tag{2.25}$$

Squaring and summing both sides of equations (2.24) and (2.25) then using some trigonometric transformations, the solution to calculate θ_2 is expressed by

$$\theta_2 = \tan^{-1}\left[\pm\frac{\sqrt{1-K_\Phi^2}}{K_\Phi}\right] - \phi - \theta_3 \tag{2.26}$$

where

$$\phi = \tan^{-1}\left(\frac{\mathrm{NP''}}{\mathrm{NP'''}}\right) = \tan^{-1}\left(\frac{\sqrt{P_x^2 + P_y^2}}{P_z - d_1}\right) \tag{2.27}$$

$$K_{\Phi} = \frac{P_x^2 + P_y^2 + (P_z - d_1)^2 + a_3^2 - a_2^2}{2a_3\sqrt{P_x^2 + P_y^2 + (P_z - d_1)^2}}. \tag{2.28}$$

The solutions for θ_2 and θ_3, as shown in Figure 2.11, corresponds to the upper-elbow configuration. It will change if another configuration is selected.

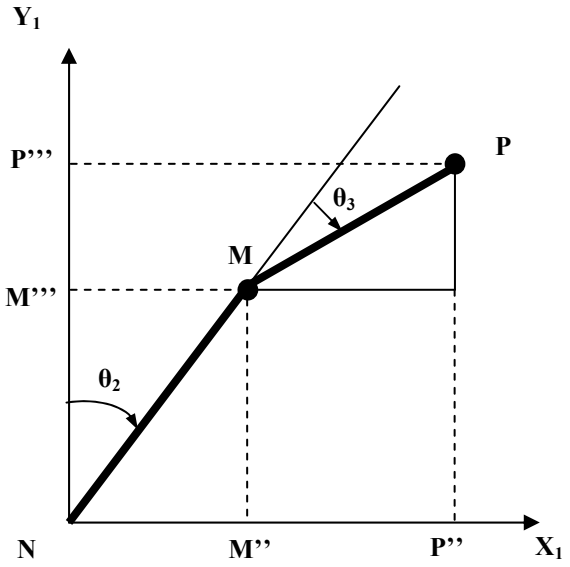


Figure 2.11 – Projecting the point P onto x_1 - y_1 plane.

The geometric approach is an exact mathematical solution for solving the inverse kinematics problem of simple configuration manipulators with revolute joints. It can also define various possible arm configurations based on the link coordinate systems and human arm geometry to analyse all the possible solutions. The appropriate solution is chosen by the user according to the specific configuration of the manipulator. However, this is a complex approach with many trigonometric transformations. For each specific configuration there is an individual way to derive the inverse kinematics

solution, so the geometric approach is dependent on user-defined solutions and is not a systematic method.

2.4 Algebraic approach

2.4.1 Principle of algebraic approach

When a manipulator has a simple geometry where at least one of the distances (a_i and d_i) are zero and most of the angles (θ_i and α_i) are zero or $\pm\pi/2$, the inverse kinematics solution can be analytically determined by an algebraic approach [2.3]. In [2.10] a solution was proposed to directly determine joint angles from the kinematics equations by using matrix algebraic techniques. This approach was applied to solve the inverse kinematics problem for a Puma manipulator [2.10], [2.11] and a Stanford/JPL manipulator [2.12].

In general, the description of the end-effector coordinate frame of a n-link manipulator with respect to the base coordinate frame is given by

$$\boldsymbol{T}_0^n = \boldsymbol{A}_0^1(\theta_1)\boldsymbol{A}_1^2(\theta_2)\ldots\boldsymbol{A}_{n-1}^n(\theta_n) \tag{2.29}$$

where $\boldsymbol{A}_{i-1}^i(\theta_i)$ is the homogenous transformation relating the coordinate frame of link i to the coordinate frame of link i-1.

The origin of the end-effector coordinate frame can be measured and represented by a given location (position and orientation) with respect to the base coordinate system as

$$\boldsymbol{U}_1 = \begin{bmatrix} \boldsymbol{n} & \boldsymbol{s} & \boldsymbol{a} & \bar{\boldsymbol{p}}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.30}$$

where $\bar{\boldsymbol{p}}_0$ is the position vector and $\boldsymbol{n}$, $\boldsymbol{s}$, $\boldsymbol{a}$ are unit vectors to define a coordinate frame for the end-effector [2.1].

Thus, the necessary equations representing the relationship between a given location of the end-effector in the workspace and corresponding joint angles can be determined by

$$\boldsymbol{U}_1 = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \boldsymbol{A}_0^1(\theta_1)\boldsymbol{A}_1^2(\theta_2)\ldots\boldsymbol{A}_{n-1}^n(\theta_n) \tag{2.31}$$

where $\boldsymbol{n}$, $\boldsymbol{s}$, $\boldsymbol{a}$, and $\bar{\boldsymbol{p}}_0$ are the known vectors representing the orientation and position of the end-effector in the workspace and the joint angles θ_i (i = 1, 2,.., n) are unknown variables. Hence, the inverse kinematics solutions for a manipulator can be obtained by solving equation (2.31).

To find the solutions of these joint angles, Paul [2.10] proposed separating each joint angle onto the left-hand side, one after another, by successively pre-multiplying equation (2.31) by $\boldsymbol{A}_i^{i-1}$ (i = 1, 2,.., n-1). These joint angles are then determined by equating appropriate elements on both sides of the matrix equation. For example, for a six D.O.F manipulator, the first matrix equation can be obtained by pre-multiplying equation (2.31) by $\boldsymbol{A}_0^1$

$$\left(\boldsymbol{A}_0^1\right)^{-1}\boldsymbol{U}_1 = \boldsymbol{A}_1^2\boldsymbol{A}_2^3\boldsymbol{A}_3^4\boldsymbol{A}_4^5\boldsymbol{A}_5^6 = \boldsymbol{U}_2\,. \tag{2.32}$$

In equation (2.32), the elements on the left-hand side are constants, or functions, of θ_1, but the elements on the opposite side are constants, or functions, of the remaining joint angles θ_2, θ_3,..., θ_6. By finding and equating appropriate elements from both sides, the joint angle θ_1 can be solved. If the implementation is continued by pre-multiplying equation (2.32) by $\boldsymbol{A}_1^2$, a new matrix equation can be obtained

$$\left(\boldsymbol{A}_1^2\right)^{-1}\left(\boldsymbol{A}_0^1\right)^{-1}\boldsymbol{U}_1 = \boldsymbol{A}_2^3\boldsymbol{A}_3^4\boldsymbol{A}_4^5\boldsymbol{A}_5^6 = \boldsymbol{U}_3 \tag{2.33}$$

Thus, once the joint angle θ_1 has been determined, the matrix elements on the left-hand side of equation (2.33) are functions of θ_2 only. The matrix elements of the right-hand

side are either constants, or functions, of the remaining joint angles (θ_3 to θ_6). Similarly, by equating appropriate elements on both sides of the matrix equation, the solution to determine the joint angle θ_2 can be sequentially obtained. The procedure is repeated to determine the solutions for the remaining joint angles.

2.4.2 The algebraic approach for a three-link manipulator

The algebraic approach is applied to the first three links of the PA10 manipulator to determine expressions for the joint angles θ_1 to θ_3 (Figure 2.9). From its geometric parameters, the transformation matrices related to the coordinate systems of links 1, 2, and 3 have been described by equations (2.6) to (2.8) respectively. The homogenous transformation matrix $\boldsymbol{T}_0^3$ from the coordinate frame attached to the end-effector to the base coordinate frame has also been derived as equation (2.9). If the location of the end-effector is given, all elements of the homogenous matrix $\boldsymbol{T}_0^3$ are defined in the matrix $\boldsymbol{U}_1$

$$\boldsymbol{U}_1 = \boldsymbol{T}_0^3 = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.34}$$

As proved in [2.1], the inversion of a general 4x4 transformation matrix, $\boldsymbol{T}_0^3$, can be determined by

$$\boldsymbol{T}_3^0 = (\boldsymbol{T}_0^3)^{-1} = \begin{bmatrix} n_x & n_y & n_z & -p'n \\ s_x & s_y & s_z & -p's \\ a_x & a_y & a_z & -p'a \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.35}$$

Applying equation (2.35) to $\boldsymbol{A}_0^1$, an inversion matrix $(\boldsymbol{A}_0^1)^{-1}$ is determined. Pre-multiplying both sides of equation (2.34) by $(\boldsymbol{A}_0^1)^{-1}$ gives

$$\left(\boldsymbol{A}_0^1\right)^{-1}\boldsymbol{U}_1 = \boldsymbol{A}_1^2\boldsymbol{A}_2^3. \tag{2.36}$$

The left-hand side of equation (2.36) contains functions of the joint angle θ_1 only as

$$\begin{bmatrix} C_1n_x + S_1n_y & C_1s_x + S_1s_y & C_1a_x + S_1a_y & C_1p_x + S_1p_y \\ n_z & s_z & a_z & p_z - d_1 \\ S_1n_x - C_1n_y & S_1s_x - C_1s_y & S_1a_x - C_1a_y & S_1p_x - C_1p_y \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.37}$$

The right-hand side of equation (2.36) is expressed as

$$\boldsymbol{A}_1^2\boldsymbol{A}_2^3 = \begin{bmatrix} C_{23} & -S_{23} & 0 & a_3C_{23} + a_2C_2 \\ S_{23} & C_{23} & 0 & a_3S_{23} + a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.38}$$

Therefore, after equating the elements in the third row, fourth column of equations (2.37) and (2.38), the solution to the joint angle θ_1 is given as

$$S_1p_x - C_1p_y = 0. \tag{2.39}$$

The joint angle θ_1 can be calculated by

$$\theta_1 = \tan^{-1}\left(\frac{p_y}{p_x}\right). \tag{2.40}$$

Having determined θ_1, the left-hand side of equation (2.36) is completely defined. If equations (2.37) and (2.38) are equated at elements in the first row, fourth column and elements in the second row, fourth column, two new equations which contain functions of θ_2 and θ_3 can be derived as

$$a_3 C_{23} + a_2 C_2 = C_1 p_x + S_1 p_y \tag{2.41}$$

$$a_3 S_{23} + a_2 S_2 = p_z - d_1 . \tag{2.42}$$

After squaring and summing both sides of equations (2.41) and (2.42), θ_2 is eliminated and θ_3 is finally determined by

$$\theta_3 = \tan^{-1}\left[\pm \frac{\sqrt{1-K^2}}{K} \right] \tag{2.43}$$

where

$$K = \frac{(C_1 p_x + S_1 p_y)^2 + (P_z - d_1)^2 - a_2^2 - a_3^2}{2 a_2 a_3} . \tag{2.44}$$

To derive the solution for the joint angle θ_2, equations (2.41) and (2.42) can be rewritten as

$$a_2 C_2 = C_1 p_x + S_1 p_y - a_3 C_{23} \tag{2.45}$$

$$a_2 S_2 = p_z - d_1 - a_3 S_{23} . \tag{2.46}$$

After several trigonometric calculation steps, the joint angle θ_2 is finally determined by

$$\theta_2 = \tan^{-1}\left(\pm \frac{\sqrt{1-K_\Phi^2}}{K_\Phi^2} \right) + \phi - \theta_3 \tag{2.47}$$

where

$$K_{\Phi} = \frac{(C_1 p_x + S_1 p_y)^2 + (p_z - d_1)^2 + a_3^2 - a_2^2}{2a_3\sqrt{(C_1 p_x + S_1 p_y)^2 + (p_z - d_1)^2}} \tag{2.48}$$

$$\phi = \tan^{-1}\left(\frac{p_z - d_1}{C_1 p_x + S_1 p_y}\right). \tag{2.49}$$

Note that θ_2 and θ_3 are multi-solution functions according to the plus or minus sign adopted in equations (2.43) and (2.47).

This approach uses matrix algebra to find the inverse kinematics solution for all joints sequentially. These solutions can be directly determined from the kinematics equations by using matrix algebraic techniques and trigonometric transformations. It can be applied to most industrial manipulators. However, the decision of how to choose suitable elements to determine the necessary equations when equating both sides of a matrix is largely based on intuition. It also involves a complex procedure of trigonometric transformation and the number of computations is higher than the geometric approach as well. However, the algebraic approach is simpler and more systematic than the geometric approach which involves complex geometric projections.

2.5 Conclusion

This chapter has presented an overview of traditional approaches to solve the inverse kinematics of a robotic manipulator. Analytical approaches, geometric and algebraic, attempt to mathematically produce an exact solution for the inverse kinematics problem by analysing the link geometry and using trigonometric transformation techniques. They can be used in real-time computer-based control applications. However, these approaches are only possible for relatively simple robotic manipulator configurations and involve an intuitive way to decide an appropriate result from the several possible solutions available for a particular manipulator. The traditional methods develop their computational algorithms based on the knowledge of manipulator geometry. In some cases when manipulator geometry cannot be exactly specified, these solutions become difficult or impractical. Therefore, an alternative approach using artificial neural

networks can be used to overcome this problem. In the next chapter, neural networks are presented to solve the inverse kinematics problem of unknown geometry manipulators.

2.6 References

[2.1] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics – Control, Sensing, Vision and Intelligence.* McGraw Hill, 1987.

[2.2] J. Denavit and R. S. Hartenbergh, "A Kinematic Notation for Lowe-Pair Mechanisms Based on Matrices", *Journal of Applied Mechanics*, June 1955, pp. 215-221.

[2.3] W. Khalil and E. Dombre, *Modelling, Identification & Control of Robots.* Hermes Penton Ltd., 2002.

[2.4] G. Z. Grudic and P. D. Lawrence, "Iterative Inverse Kinematics with Manipulator Configuration Control", *IEEE Trans. on Robotics and Automation*, vol. RA-9(4), August 1993, pp. 476-483.

[2.5] R. Featherstone, "The Position and Velocity Transformation between Robot End-Effector Coordinates and Joint Angles", *Int. Journal of Robotics Research*, vol. 2(2), 1983, pp. 35-45.

[2.6] *General Purpose Robot PA10-6CE – Instruction Manual for Installation, Maintenance and Safety*, Mitsubishi Heavy Industries Ltd.

[2.7] C. Bergren, *Anatomy of a Robot*. McGraw Hill, 2003.

[2.8] C. S. G. Lee, "Robot Arm Kinematics, Dynamics, and Control", *Computer*, vol. 15(12), December 1982, pp. 62-80.

[2.9] C. S. G. Lee and M. Ziegler, "Geometric Approach in Solving Inverse Kinematics of Puma Robots", *IEEE Trans. on Aerospace and Electronic Systems*, vol. AES-20(6), November 1984, pp. 695-706.

[2.10] R. Paul, B. Shimano, and G. E. Mayer, "Kinematic Control Equation for Simple Manipulators", *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-11(6), June 1981, pp. 449-455.

[2.11] R. Paul, *Robot Manipulator: Mathematics, Programming and Control*. MIT Press, 1981.

[2.12] A. J. Koivo, *Fundamentals for Control of Robotic Manipulators*. John Wiley & Sons, 1989.

[2.13] J. Angeles, *Fundamental of Robotic Mechanical System: Theory, Methods and Algorithms – Second Editor*. Springer, 2003.

[2.14] B. Z. Sandler, *Robotics - Designing the Mechanism for Automated Machines.* Academic Press, 1999.

CHAPTER 3
BACKGROUND OF NEURAL NETWORKS FOR INVERSE KINEMATICS APPROXIMATION OF ROBOTIC MANIPULATORS

3.1 Introduction

The previous chapter described traditional methods for solving the inverse kinematics problem, which plays an important role in robotic control. However, these solutions are based on the geometric parameters of robotic manipulators to develop their computational algorithms. Using the traditional methods becomes difficult or impractical if the geometric parameters are unknown. One such case is a robot-vision system in which the position of a robot is measured by a vision-based measurement system. If there is any change in the visual measurement system or the robot's base is moved in the workspace, the traditional methods will fail. This chapter presents a review of alternative solutions using artificial neural networks to solve the inverse kinematics problem assuming no prior knowledge of the manipulator's geometric configuration.

There have been many solutions proposed using neural networks to solve the inverse kinematics problem of an unknown geometry manipulator. One solution followed a closed-loop control scheme where a neural network is used to directly learn the nonlinear relationship between the displacement in the workspace and control signal in the joint angle space to achieve a desired position [3.1]-[3.5]. Another solution was proposed to learn the inverse mapping of both the position and velocity from the workspace to joint space by using a numerical solution based on the Jacobian matrix [3.6]-[3.12]. Other schemes used a self-organized network system [3.13]-[3.15], a multi-layer perceptron network [3.21]-[3.29] and a radial basis function network [3.32]-[3.35] to learn a mapping function from the world space to joint angle space.

This chapter firstly discusses the background to neural computing. A literature review of two of the most popular networks, the multi-layer perceptron networks (MLPNs) and radial basis function networks (RBFNs), to solve the inverse kinematics problem are

then introduced. The specific characteristics of both network types are analysed and compared. Finally, the reasons why RBFNs have been adopted in this research are discussed in the conclusion.

3.2 Background of neural computing

In general, a neural network consists of a number of simple processing nodes called artificial neurons which are connected together to form functional layers (input, output, and hidden layer) [3.16]. A neuron model represents a biological neuron that fires when its inputs are significantly excited (i.e., high enough) and an activation function (linear or nonlinear) is used to measure that excitation. There are many kinds of activation functions, such as threshold, sigmoid and hyperbolic tangent, that can be used in a neuron model. For example, Figure 3.1 presents a generic artificial neuron with the threshold function.

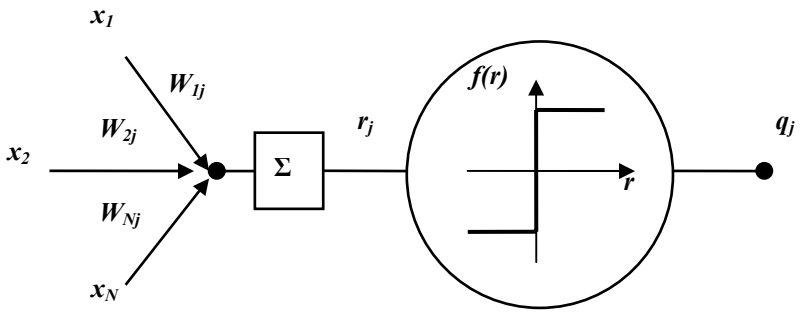


Figure 3.1 - A generic artificial neuron.

In this basic model, each neuron accepts a weighted set of inputs and responds by an output according to excitation strength and activation function characteristics. A neuron first forms the sum of weighted inputs to produce a stimulus signal given by

$$r_j = \sum_{i=1}^{N} W_{ij} x_i \tag{3.1}$$

where W_{ij} is a synaptic weight between an input vector x_i and the j^{th} neuron.

Synaptic weights represent the strength of interconnections between inputs and a neuron (or between neurons of other layers) and can be adjusted through a learning process. The stimulus signal r_j is the sum of the weighted inputs which take the form of data items either from the environment or from other network elements, possibly from the outputs of nodes in a previous layer. To expand the controllability of neurons, a bias is added to the stimulus signal r_j. It is equivalent to an additional input with a synaptic weight of b_j added to the neuron's inputs [3.16]. The output is then calculated as a function of the stimulus signal by

$$q_j = f(r_j + b_j) \text{ or} \tag{3.2}$$

$$q_j = f(\sum_{i=1}^{N} W_{ij} x_i + b_j). \tag{3.3}$$

This output is a result of the neuron's response according to inputs, synaptic weights and activation function characteristics. Generally, knowledge stored in neural networks (combination of many generic neurons) is represented by their structure (i.e., topology, number of hidden layers and number of neurons in each layer), the specific features of neurons (activation function) and the values of the interconnection weights between neurons. Figure 3.2 presents a simple feed-forward neural network.

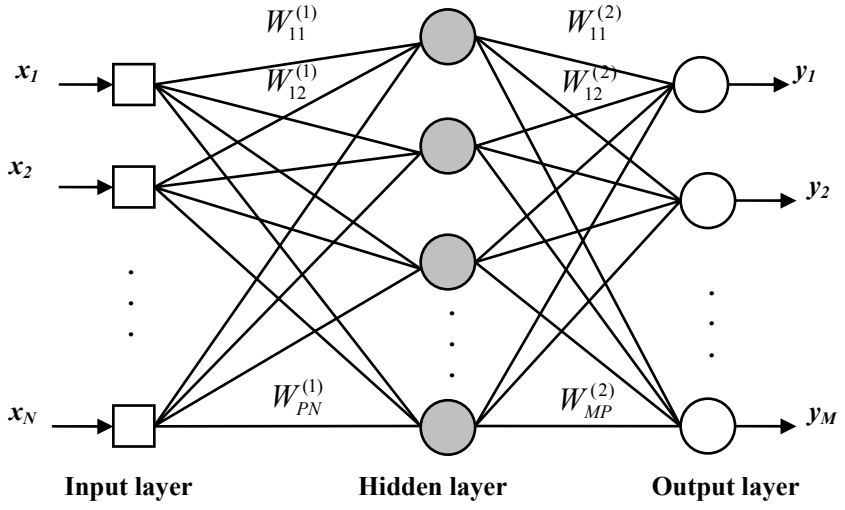


Figure 3.2 - General structure of a neural network.

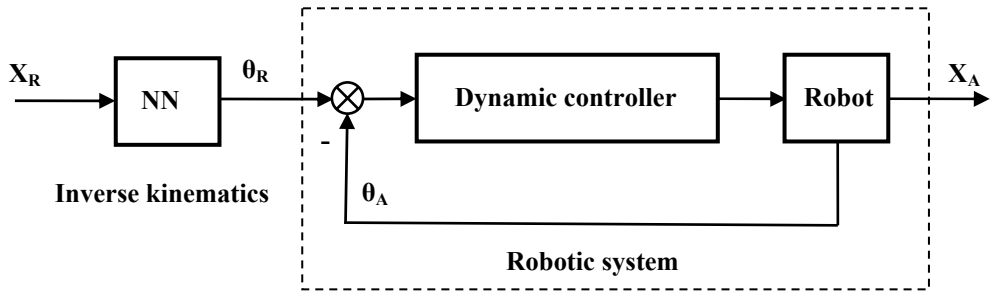
It contains an input layer which is used to receive stimuli from the environment, one or more hidden layers whose computation nodes intervene between the external inputs and the network output layer in some useful manner and an output layer which creates network responses according to inputs presented.

If a neural network is used to approximate a system model, the number of inputs and outputs of this network can be determined from the physical characteristics of this system. They are often equal to the number of input and output parameters of the system model (e.g., three inputs and three outputs for the inverse kinematics problem of a 3-link robot). However, other important features, such as the number of hidden layers and the number of neurons in each layer, the connection type (feed-forward or recurrent) between the layers and the neuron model (perceptron, radial basis function) need to be tailored (or investigated) according to the individual problem.

A common problem in neural network applications is to reproduce a function by learning from a set of examples (i.e., pairs of input-output data) without any knowledge of that function. This learning process is called supervised training [3.18]. The supervised training process aims to adjust the association weights between the processing nodes (neurons) so that errors between the actual outputs and desired responses of a network are minimised by an optimisation process. This training process ceases when the error falls below an expected goal or the maximum number of epochs is exceeded. As a result, a neural network can learn to approximate any nonlinear function by constructing the input-output mapping from information provided by training examples. After training, the neural network is able to produce a reasonable response for a new input according to its generalisation capability. The network operation is fast because it just comprises of simple arithmetical calculations. Neural networks also have the capability to adapt their association weights to changes in the surrounding environment, even in real-time through an online training process. Properties such as learning ability, generalisation and adaptability make neural networks distinctly different from conventional methods based on mathematical algorithms. For that reason, neural networks can be applied to a wide range of complex problems in many scientific fields including robotics control [3.16], [3.17].

3.3 General training schemes of neural networks used for inverse kinematics approximation

In this specific application, a neural network is applied to solve a problem called functional approximation in which the network should initially be trained through an offline process to approximate the inverse kinematics transformation of the manipulator. The network can then work as an independent controller in the operational phase as shown in Figure 3.3. This is called a feed-forward controller and is the most popular scheme in neural network applications. The success of the feed-forward scheme is closely related to the generalisation ability of the neural network to produce (or generalise) an appropriate response to any input that it has not been trained with, based on the experiential knowledge stored in its structure.



NN – Inverse kinematics approximation using a neural network.

Dynamic controller – Standard PID controller for joint angle control.

$\mathbf{X_R}$, $\mathbf{X_A}$ – Desired and actual positions of the robot in the world space.

$\boldsymbol{\theta_R}$, $\boldsymbol{\theta_A}$ – Desired and actual joint angle references.

Figure 3.3 – General structure of inverse kinematics approximation using a neural network.

To train the network in this feed-forward scheme, a general learning architecture can be used as shown in Figure 3.4 [3.19]. The training data is a set of reference joint angles θ_R and actual positions X_A**.** The training process can be implemented by two independent steps, collection of the training data and updating the network weights**.** To collect training data a set of arbitrary reference joint angles is sent to the robot's control system and the robot then moves to specific positions in the workspace corresponding to this

command set. The neural network then uses this pair, actual position and reference joint angle, as a training sample in the supervised learning procedure. The error between the actual response of the network output and the reference joint angle is used to update the network's weights. This is the most common training scheme in neural computing. However, this training architecture has a major drawback that may lead to poor generalisation in some operational regions when operating as a feed-forward controller. This is because the training region, where data was collected, is sometimes far from the actual operating region. This problem is due to the fact that training data is collected in the network output space, instead of the network input space. For example, training data can be sampled regularly for a range of joint angles (the network output space) in Figure 3.3, but they will be arbitrary positions in the Cartesian space (the network input space). The collected data may not reflect the full characteristics of the desired function (the inverse kinematics) and the network after training may not work well in this operational phase. To overcome this difficulty, training patterns could be collected over a wide region producing a significant amount of data. This may be unreasonable and inefficient since the neural network has to learn more training examples than is actually necessary.

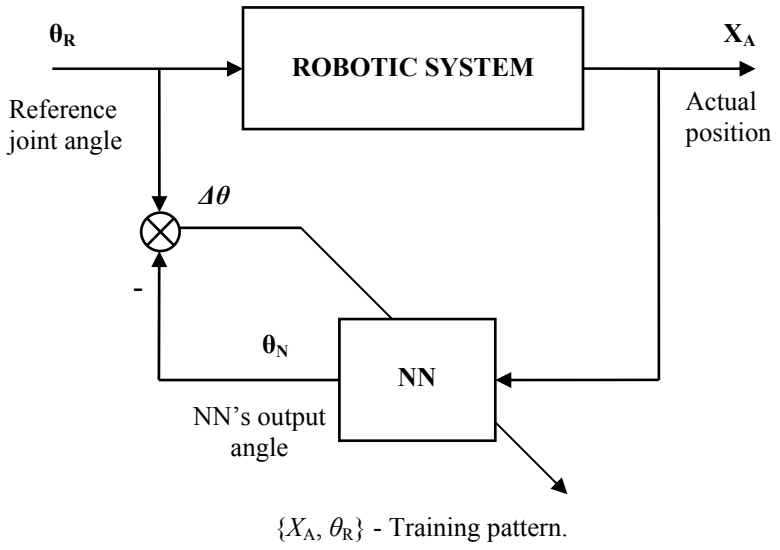


Figure 3.4 – The general training architecture.

Another training scheme called the specialised learning architecture [3.17], [3.19] can be applied as shown in Figure 3.5. This architecture directly uses errors between the desired and actual outputs of the robotic system to adjust the weights so that the errors

decrease. In this training architecture the training data can be collected in the network input space to approximate the desired function over the whole operating region. However, because the training algorithm criteria are in terms of the network output (actual joint angles) instead of the system output (actual positions), the error ΔX in the Cartesian space should be projected back into the error $\Delta\theta$ in the joint angle space. Thus, it requires knowledge about the Jacobian matrix of the robot kinematics. The Jacobian matrix contains partial derivatives of the position elements of each joint.

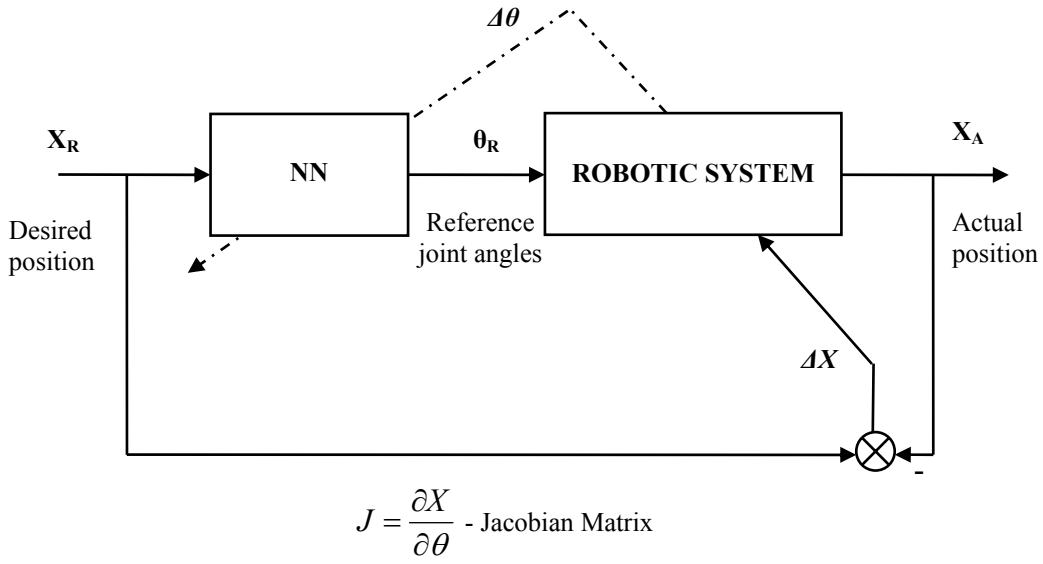


Figure 3.5 - Specialized learning architecture.

As the training and operational phases of the network are the same structure (Figures 3.3 and 3.5), the specialised learning architecture is able to update the network weights online while operating in regions of interest.

In this research, the training process was performed using this general learning architecture. However, the idea of collecting constrained data was proposed to overcome the drawback of poor generalisation in the operational phase due to the arbitrary collection of data.

3.4 Multi-layer perceptron networks

Multi-Layer Perceptron Networks (MLPNs) are the most widely encountered neural networks in control problems. They are a powerful solution to solve problems such as function approximation or process modelling. This is because of their inherent nonlinear mapping capabilities which can deal with a wide range of process features [3.17]. This section presents the general characteristics of MLPNs and a literature review of MLPNs used for inverse kinematics approximation.

3.4.1 Structure of MLPNs

MLPNs, one of the most popular neural networks, have been successfully applied to solve many complex problems by using a supervised training method known as the back-error propagation algorithm (back propagation) [3.16], [3.20]. An MLPN typically consists of an input layer, one or more hidden layers and an output layer of computational nodes as shown in Figure 3.6.

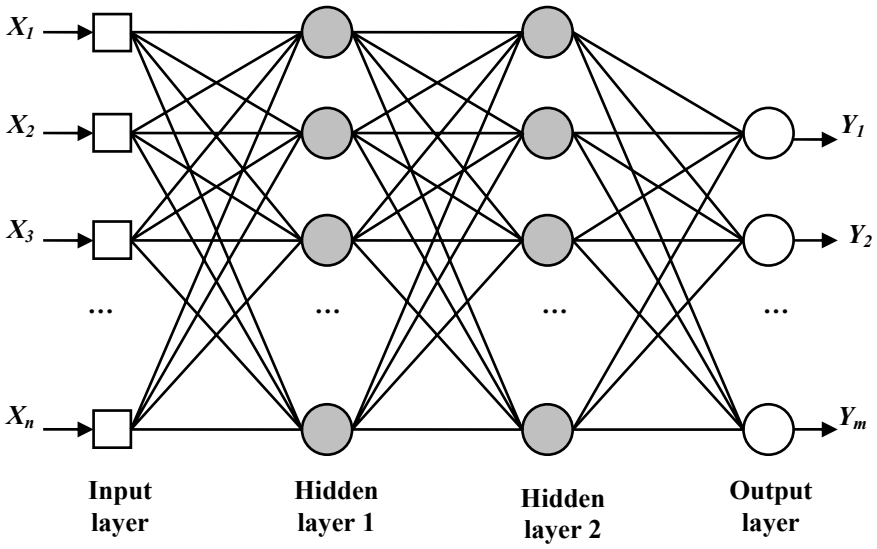


Figure 3.6 – Architecture of an MLPN with two hidden layers.

In an MLPN, each node includes a nonlinear activation function to present the nonlinear relationship between input-output pairs similar to the biological motivation of brain

neurons. Figure 3.7 presents a common form of the activation function called the sigmoid expressed as

$$f(r) = \frac{1}{1 + \exp(-kr)} \tag{3.4}$$

where r is the stimulus signal of the neuron which is a weighted sum of all synaptic inputs calculated by equation (3.1) and k is the slope parameter of the sigmoid function.

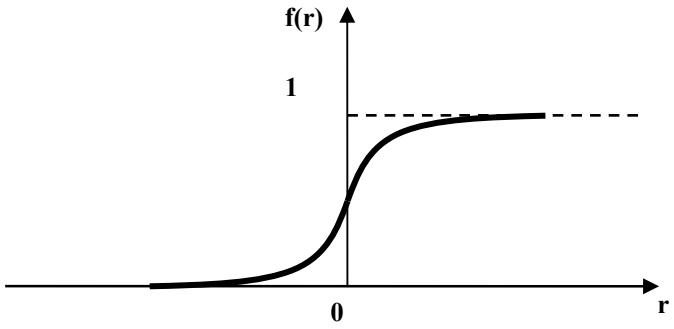


Figure 3.7 - Sigmoid function.

Basically, knowledge in an MLPN is represented by the network configuration (i.e., the number of hidden layers and number of neurons at each hidden layer) and the value of association weights between neurons in related layers as well. Thus, in order to develop a functional approximation based MLPN for a system, some important aspects related to the selection of the network structure parameters should be investigated in accordance with the system's characteristics. This includes the number of hidden layers, numbers of hidden neurons at each layer and the type of activation function. Specifically, the question of how to select an appropriate network structure is always considered and needs to be resolved at the first stage [3.17]. In most cases, the structure of the MLPN is often chosen in a heuristic way. Thus, for a specific problem, a reasonable number of hidden layers and neurons at each hidden layer are initially selected based on experience. Adjustments then can be made on a trial and error basis, if the chosen structure appears unsatisfactory. Once the selection of network structure has been decided, the training process is implemented to update the network weights so that the

network can perform as desired. A famous supervised learning method called the back propagation algorithm is presented in the next section.

3.4.2 Back propagation training algorithm

The back propagation algorithm was presented in [3.16], [3.17] to train MLPNs with more than one hidden layer. It is based on an idea that the error signal at the output layer can be propagated backwards through the network to update the hidden layer weights. This can be briefly described as follows.

At the output layer, the error signal of neuron j is defined by

$$e_j = t_j - y_j \tag{3.5}$$

where t_j is the target output of neuron j and y_j is an actual output of neuron j.

The purpose of the training process is to adjust the weights between neurons, so that this output error signal e_j decreases to a minimum. An optimisation algorithm can be applied with the constraint as an error energy function, which depends on the network weights, defined by

$$\varepsilon_j = \frac{1}{2}e_j^2 = \frac{1}{2}(t_j - y_j)^2 . \tag{3.6}$$

If L is the total number of training samples presented to the network, the average error energy of all outputs throughout all training samples can be obtained by equation (3.7)

$$\mathrm{E} = \frac{1}{2L}\sum_{l=1}^{L}\sum_{j=1}^{M}(t_j(l) - y_j(l))^2 \tag{3.7}$$

where $t_j(l)$ is the l^{th} target output of neuron j, $y_j(l)$ is the actual output of neuron j corresponding to stimulus of the l^{th} input and L and M are the number of training samples and output neurons respectively.

This average error energy E (called mean square error) is a function of all adjustable parameters (synaptic weights and/or bias levels) and could be used as the cost function to measure the learning performance. A training process is then used to adjust the network weights in order to minimise that cost function. It is an optimisation problem and can be implemented by several popular methods such as the steepest descent (or gradient descent), Newton and Gauss methods [3.16]. Based on the gradient descent method the network weights are adjusted according to

$$w_{ji}^{New} = w_{ji}^{Old} - \eta \frac{\partial E}{\partial w_{ji}}. \tag{3.8}$$

The correction of synaptic weight Δw_{ji} can be expressed as

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}. \tag{3.9}$$

In order to calculate the correction Δw_{ji} at iteration k, the partial derivative of the energy function E with respect to the weight w_{ji} can be obtained by multiplying a range of the partial derivatives of related functions as shown in [3.17], [3.20]. Finally, it is determined by

$$\Delta w_{ji}(k) = \eta e_j(k) \dot{\varphi}_j(r_j(k)) y_i(k) \tag{3.10}$$

where η is the learning rate, $e_j(k)$ is the output error of neuron j, $\dot{\varphi}_j(r_j(k))$ is the derivative of the sigmoid function and $y_i(k)$ is the output of neuron i at the preceding layer.

From equation (3.10), a key factor involved in correcting the network weight Δw_{ij} is the output error signal $e_j(k)$ of neuron j. Consequently, it is necessary to consider two cases depending on whether the location of neuron j is in the output layer or hidden layers.

If neuron j is located in the output layer, the error $e_j(k)$ is directly calculated from the desired output (training data) and actual response as in equation (3.5). The correction weight $\Delta w_{ij}(k)$ can then be determined from equation (3.10).

If neuron j is located in one of the hidden layers, the error signal $e_j(k)$ cannot be directly calculated from the training data. However, it can be obtained recursively from the error signals of the output-layer neurons, or the following layer, that are connected to this. The error of the output-layer neurons are propagated backwards to determine the hidden-layer weights. From equation (3.10) a function of the output error called the local gradient $\delta_j(k)$ is defined by

$$\delta_j(k) = e_j(k)\dot{\varphi}_j(r_j(k)). \tag{3.11}$$

This $\delta_j(k)$ can be regressively determined from the local gradient of the following layer as

$$\delta_j(k) = \dot{\varphi}_j(r_j(k))\sum_p \delta_p(k) w_{pj}(k) \tag{3.12}$$

where $\delta_p(k)$ is the local gradient computed for neurons p in the following hidden layer (or output layer) that are connected to neuron j and $w_{pj}(k)$ is the weight associated with the connection between neuron p and neuron j.

The correction weight $\Delta w_{ij}(k)$ can then be determined by equation (3.10).

3.4.3 Using MLPNs to approximate the inverse kinematics

The MLPN is the most popular neural network applied to functional approximation problems. Thus, the use of MLPNs in the inverse kinematics problem has occurred to a greater extent compared to other networks. There have been many approaches using various structures based on MLPNs to approximate the inverse kinematics. This is not only for simple configurations (ordinary manipulators) but also some complex cases,

such as redundant or singular configurations. This section presents the research that reflects the most important aspects of MLPNs applied to the inverse kinematics problem.

An approach using MLPNs with two hidden layers trained by the back-error propagation algorithm was proposed to approximate the inverse kinematics function in two- and three-dimensional space [3.22], [3.23], [3.26]. Figure 3.8 shows the general structure of the MLPN where its inputs are the end-effector coordinates (Cartesian space) and its outputs are the joint angles.

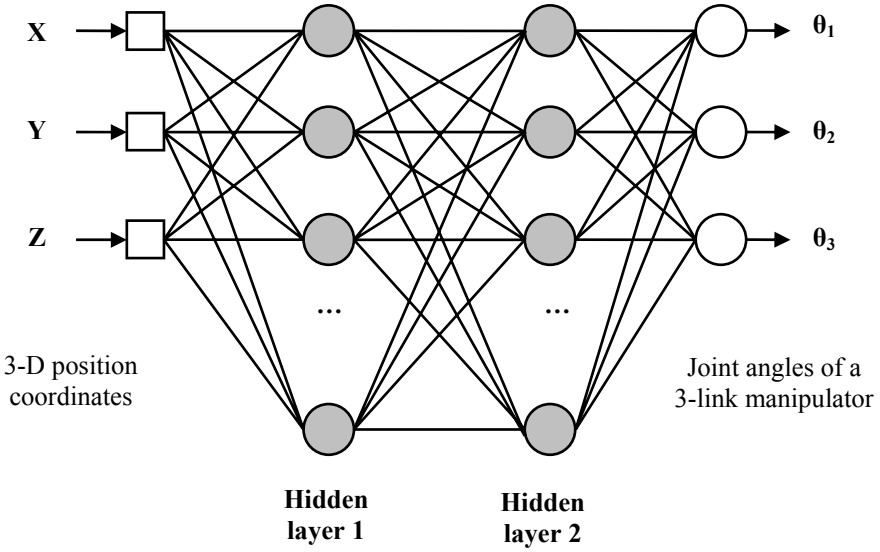


Figure 3.8 – A general structure of the MLPN to approximate the inverse kinematics of a manipulator.

In [3.23] the effect of structural parameters (i.e., the number of hidden layers and the number of neurons in each hidden layer), iteration steps and different numbers of training points on the performance of the inverse kinematics approximation was investigated. A more complex MLPN configuration is likely to produce a more accurate inverse kinematics approximation. However, it also leads to the number of iterations increasing significantly to satisfy the required training goal. Similarly, when increasing the number of training points (the size of training data) the network's performance (generalisation ability) seems to be improved, but it also requires many more iterations

to satisfy the training goal. The general trend of an MLPN training process is that the number of iterations increases exponentially with the number of training points and rapidly with the number of hidden neurons. For example, with the structure of 2-10-10-2 (two hidden layers), the training process needed 150,000 iterations to achieve a 1-percent RMS error training accuracy for 73 training points collected arbitrarily in the two-dimensional workspace [3.23]. In some cases when the number of hidden neurons or training points were too large, the training process cannot even converge to an expected error goal.

In [3.24] an MLPN with various structures of the input layer was proposed to solve the inverse kinematics problem of a 6 D.O.F manipulator. Three different forms representing the orientation of the end-effector with respect to the base were defined: a 3x3 rotation matrix (9 elements), a set of 3 Euler angles (3 elements) and one angle and a 1x3 unit vector (4 elements). According to the three ways of representing the orientation, there were three different network configurations established to perform the inverse kinematics approximation. The simulation investigated the effect of the each network configuration on the operational phase. The results showed that the first configuration with 9 elements representing the orientation produced the best performance. However, all three cases produced significant performance errors.

Other solutions were proposed by modifying the MLPN structure to improve the performance of complex configuration manipulators with redundancy and singularities. For example, a solution combining an MLPN and a lookup table to solve the inverse kinematics problem of a redundant manipulator was proposed in [3.25]. There are many solutions (sets of joint angles) that are available for a single position of the end-effector of a redundant manipulator. The MLPN should be trained to recognise different solutions corresponding to each common position according to the specific configuration. Therefore, lookup tables can be used to store the network knowledge corresponding to each configuration. After training, individual lookup tables of the network weights were created for each configuration (or orientation) of the redundant manipulator. These lookup tables can then be employed, depending on the required orientation of the manipulator, to obtain suitable joint angles for a given position of the end-effector in the workspace.

A new approach using an MLPN in parallel with a conventional inverse kinematics computation module was developed in [3.27]. The MLPN operated as a compensation component to improve the accuracy of the position control system. Using this scheme the MLPN can compensate for inaccuracies due to mismatch between the system model used to develop the inverse kinematics module and the real robotic system. Figure 3.9 shows the general diagram of this combination configuration.

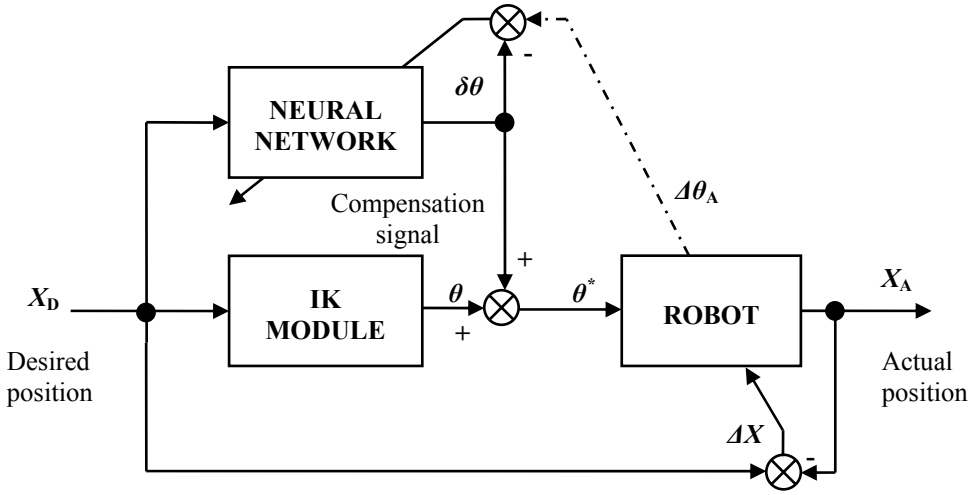


Figure 3.9 - Combination between a conventional inverse kinematics module and an MLPN to improve positioning accuracy of the manipulator.

The input layer of the MLPN in Figure 3.9 comprises of 9 inputs (P_x, P_y, P_z, a_x, a_y, a_z, s_x, s_y, s_z) to represent a location (position and orientation) of the end-effector. Training was implemented by an offline process following the specialised learning architecture in which the error ΔX between desired and actual location can be used to adjust the network weights. This error ΔX needs to be projected back to the target output $\Delta\theta$ of the MLPN in the joint angle space by using

$$\Delta\theta = J^{-1}\Delta X \tag{3.13}$$

where J^{-1} is the inverse of the Jacobian matrix which can be determined from the kinematic equations of the manipulator.

The solution using the MLPN operating in parallel with a conventional inverse kinematics computation module improved the positioning accuracy of a six D.O.F

manipulator when there were errors in the conventional inverse kinematics model. However, the training process is complex and based on prior knowledge of the manipulator kinematics.

In [3.28], [3.29] an approach using modular neural networks to solve the inverse kinematics problem was presented. The principle of this approach was that several MLPNs, each solving the inverse kinematics for one joint, were concatenated in order to find the set of joint angles from a given location of the end-effector in a sequential way. The modular neural network's configuration is shown in Figure 3.10.

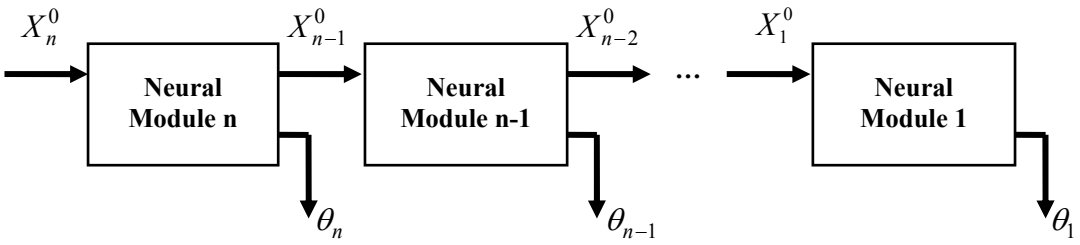


Figure 3.10 – Configuration of a modular network to solve the inverse kinematics problem.

As seen in Figure 3.10, each joint angle is sequentially derived directly from the corresponding homogenous transformation matrix. In general, the manipulator's kinematics equations can be expressed by a multiplication of the homogenous transformation matrices as follows:

$$\boldsymbol{T}_n^0 = A_1^0(\theta_1)A_2^1(\theta_2)...A_n^{n-1}(\theta_n) \tag{3.14}$$

$$= \begin{bmatrix} \boldsymbol{R}_n^0 & \boldsymbol{P}_n^0 \\ 0 & 1 \end{bmatrix} \tag{3.15}$$

where $\boldsymbol{R}_n^0$ and $\boldsymbol{P}_n^0$ are the rotation matrix and translation vector of the end-effector with respect to the base coordinate system respectively. By using the Euler angle type [φ, θ, ψ] for the orientation representation, a specific location of a frame relative to another can be described by a 6x1 vector. Thus, each neural module consists of 6 inputs

(vector X_i^0) and 7 outputs (vector $[X_{i-1}^0, \theta_i]^T$). The hidden layer's specification (number of hidden layers and number of neurons in each layer) is adjusted according to the application. The learning process is performed independently for each neural module by the back propagation algorithm. This learning aims to drive the manipulator to reach a given location by measuring corresponding input-output pairs (X_i^0, θ) [3.28]. Thus, the output error is defined by:

$$E_i = [BP(E_{i-1})^T(\theta_i^* - \theta_i)]^T \tag{3.16}$$

where $BP(E_{i-1})^T$ is the back propagation of error E_{i-1} through the neural module *i*-1.

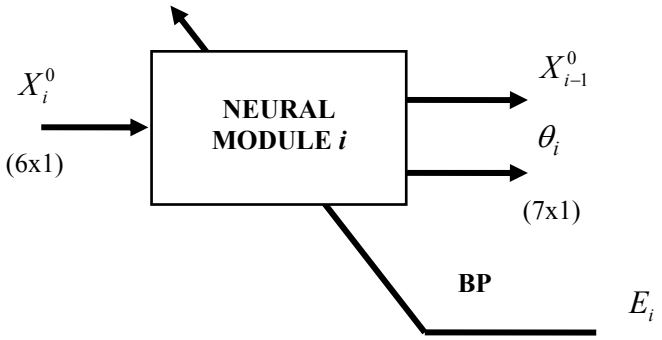


Figure 3.11 – Structure of a module (MLPN) in the modular network.

This training process is complicated due to the use of the back propagation method between different modules (individual networks) and is dependent on the forward kinematics expressions. Therefore, it does not seem suitable for practical applications.

MLPNs have been a popular solution for solving the inverse kinematics problem. It is due to their simple structure and their inherent nonlinear mapping capabilities which can deal with a wide range of process features. They are able to solve the inverse kinematics problem through interacting with input-output data using a variety of schemes. An MLPN is different from other radial basis function types because of its wider generalisation ability with the same training data set.

However, there is no reasonable mechanism to select a suitable network configuration (number of hidden layers and number of neurons at each layer) relating to the system characteristics represented by training data. Most of the structural choices are based on the user's experience and/or heuristic rules, so their performance is unpredictable. In addition, training MLPNs using the back-error propagation algorithm is complex and slow. It involves a nonlinear model formulated in the least squares manner (to build the energy function of the training process) which requires an iterative numerical procedure for optimisation. For a complex MLPN structure (multi-hidden layers and many neurons at each hidden layer) required for a complex configuration manipulator, or a large set of training data, the training process is slow to converge to a specific goal and sometimes gets stuck at a local minimum.

The performance accuracy as stated in [3.23] and [3.24] is likely to be unsatisfactory for application in a practical robotic system.

3.5 Radial basis function networks

Radial basis function networks (RBFNs) are feed-forward networks and are different from MLPNs and other networks because of the process performed at the hidden layer. The basic architecture of an RBFN is a three layer network consisting of an input layer, a single hidden layer and a linear output layer [3.16]-[3.18]. Rather than using the sigmoid activation function as in MLPNs, hidden units of RBFNs use the Gaussian function (or some basis kernel function) where each hidden unit acts as a local selector that computes a score for the match between the input vector and its centres. The basis function units are highly specialised pattern detectors. A network output is then produced by linearly combining the weighted outputs of all hidden units. Due to this specific structure, training RBFNs is simple and straightforward using the least squares approach. Recently, many solutions using RBFNs to solve the inverse kinematics problem have been developed as an alternative approach to MLPNs.

3.5.1 Structure of RBFNs

Figure 3.12 shows the typical architecture of an RBFN. The input layer is made up of the source nodes whose number is equal to the dimension N of input vector x. The

hidden layer is a group of nonlinear units which contains an activation basis function (N inputs, one output) as a ($R^N \rightarrow R$) mapping function. This basis function, where a Gaussian is the most common, has parameters such as centre and width. The centre of a Gaussian function is a vector whose size is the same as the dimension N of the input vector x. Each Gaussian function has its own centre point and the number of centre points in the workspace is the number of hidden units of the RBFN.

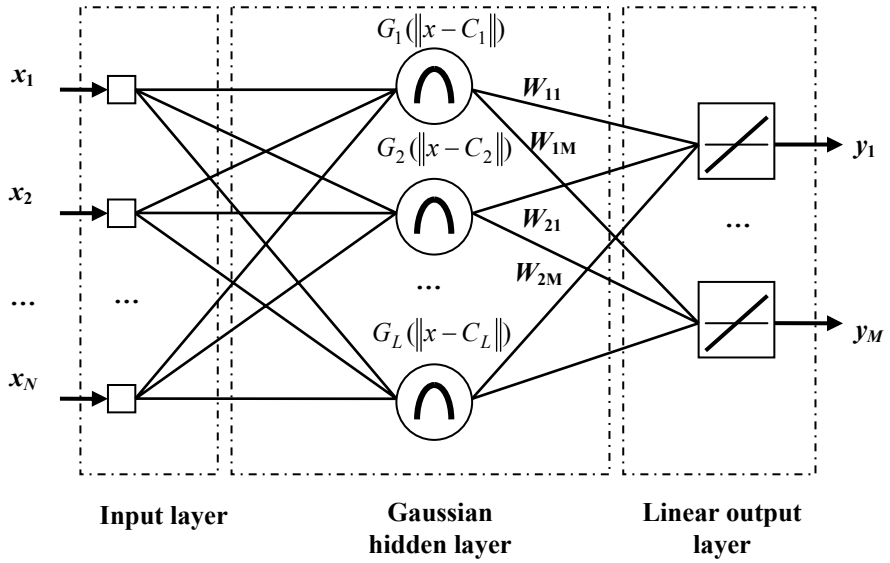


Figure 3.12 – The typical architecture of a radial basis function network, N inputs - L hidden units – M outputs.

If an input vector x (N dimensions) is sent to the RBFN, the output of the i^{th} Gaussian hidden unit can be expressed by

$$\Phi_i(x) = \exp\left(-\frac{\|x - C_i\|}{\sigma^2}\right) = \exp\left(-\frac{d_i^2}{\sigma^2}\right) \tag{3.17}$$

where d_i is the radial distance between the input vector x and centre C_i of the basis function i. It is computed by an Euclidian distance as

$$d_i = \sqrt{\sum_{k=1}^{N}(x_k - C_{ki})^2}. \tag{3.18}$$

As shown in Figure 3.13, the Gaussian function of a hidden unit is a curve which has a peak at zero distance (if the input vector coincides with the centre) and decreases as the distance from the centre is increased. The width σ defines the shape of the Gaussian function (thin or flat). The smaller the width, the thinner the shape of the Gaussian function. For convenience, the width of the Gaussian function can be represented by a spread value (SP). If the distance d_i is equal to the spread, the output of the Gaussian function is 0.5. Thus, equation (3.17) can be rewritten as

$$\Phi_i(x) = \exp\left(-d_i^2\left(\frac{0.8321}{spread}\right)^2\right). \tag{3.19}$$

The width and the spread are related to each other as

$$\sigma = \frac{spread}{\sqrt{-\ln(0.5)}} = \frac{spread}{0.8321}. \tag{3.20}$$

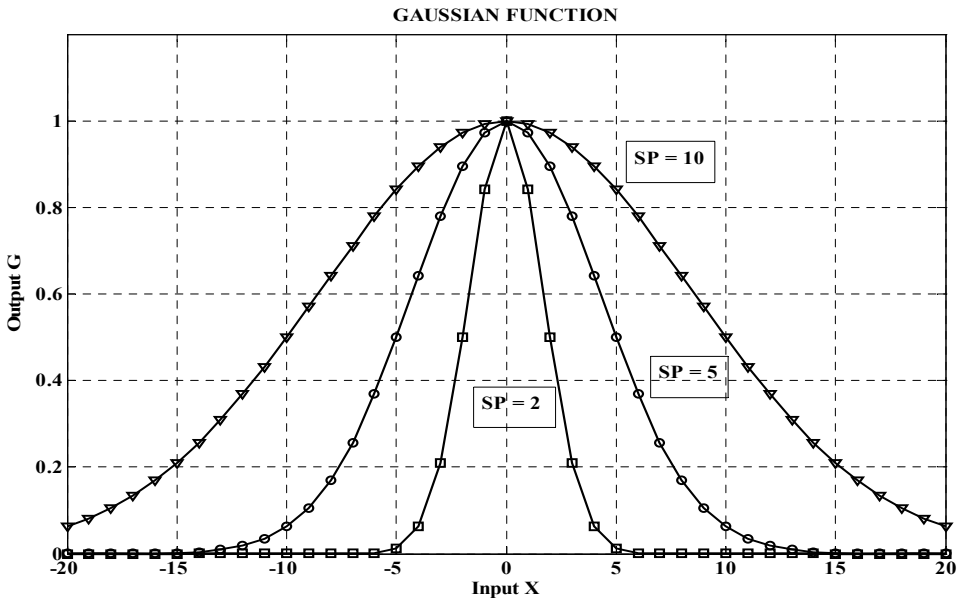


Figure 3.13 - Gaussian function with C = 0 and the SP = 2, 5 and 10.

Based on the particular structure of the RBFN, the transformation from the input space to the hidden unit space is nonlinear, whereas the transformation from the hidden unit

space to the output is linear. Therefore, for an RBFN with L hidden Gaussian units, an output j of the output layer can be computed by

$$y_j(x) = f(x) = \sum_{i=1}^{L} W_{ij} . \Phi_i(x) \tag{3.21}$$

where $\Phi_i(x)$ is the output of the i^{th} hidden unit and W_{ij} is the synaptic weight of interconnection between the i^{th} hidden unit and the output j.

The RBFN with N inputs, L hidden units and M outputs, as shown in Figure 3.12, is a mapping function $f : R^N \rightarrow R^M$ from the N-dimensional space to the M-dimensional space.

3.5.2 Training RBFNs

Knowledge of an RBFN is built based on the structure parameters of the hidden layer (centres and spreads of hidden Gaussian functions) and the linear weights. Therefore, in order to approximate a desired function, it is necessary to perform two independent steps: building the structure of the hidden layer (selection of centres and spreads) and supervised training of the linear weights. The first step is performed and then the training process is implemented.

3.5.2.1 Building the structure of the basis function hidden layer

The selection of structural configurations in terms of the number and position of basis function centres is important because it directly affects the quality of the functional approximation achieved by an RBFN. Normally, these centres can be determined with relation to the inputs of the training data by some unsupervised methods. These include Kohonen's self-organized maps and the K-means clustering technique [3.16], [3.30]. There are different ways to select these centres as a smaller subset of the training data. One is to choose a random subset from the training data and another is to incrementally select a point that minimises the training error the most, e.g., the orthogonal least squares technique [3.31]. However, the performance of an RBFN is referred to as a local mapping function where only a few of the hidden units will respond when an input is presented to the network. This is because each hidden unit has only a specific local

area in input space according to its centre position and the width of the radial basis function. Thus, it requires many hidden units to achieve a good performance within the input space and the number of centres tends to increase exponentially with respect to the input space dimension for a particular problem [3.17]. This number may possibly be higher than the number of hidden-layer nodes in an MLPN for the same problem.

To avoid increased complexity all hidden basis function units should have the same width (or spread) which can be a scaled factor of the average of centre distances.

3.5.2.2 Adjusting the linear weights

Once the hidden basis function units are set, the second phase of supervised training is used to adjust the linear weights. As there is a straightforward linear relationship between the linear weights and the network outputs, training the linear weights of RBFNs is simple. The optimal weights which minimise the cost function in the least mean squares manner can be calculated by a linear optimisation algorithm. This linear learning procedure is a significant advantage compared to the MLPN training process which requires a nonlinear optimisation algorithm [3.18].

Suppose that a training data set consisting of P training patterns, $\{(X_1, X_2,..., X_P); (T_1, T_2,..., T_P)\}$ (inputs; target outputs), is presented to the network, the training process aims to minimise the cost function

$$E = \sum_{k=1}^{P} (T_k - f(X_k))^2 \tag{3.22}$$

where $f(X_k)$ is the network output corresponding to the training pattern X_k. It is determined by equation (3.21) where the output of the i^{th} basis function under stimulus of X_k is calculated by

$$\Phi_i(X_k) = \exp\left(-\frac{\|X_k - C_i\|}{\sigma^2}\right). \tag{3.23}$$

Thus, after all training patterns are presented to the RBFN a (*P*x*L)* interpolation matrix is obtained. Every row corresponds to the responses of all hidden units for each pattern and every column to each hidden unit through all patterns

$$\boldsymbol{\Phi} = \begin{bmatrix} \Phi_1(X_1) & \Phi_2(X_1) & \cdots & \Phi_L(X_1) \\ \Phi_1(X_2) & \Phi_2(X_2) & \cdots & \Phi_L(X_2) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_1(X_P) & \Phi_2(X_P) & \cdots & \Phi_L(X_P) \end{bmatrix}. \tag{3.24}$$

In order to determine an optimal weight vector that minimises the cost function in the least squares manner, the interpolation method is used [3.16]. It can directly calculate the linear weights from target outputs and pseudo-inversion of the interpolation matrix from the following equation [3.18]

$$\boldsymbol{W} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{T}. \tag{3.25}$$

This solution produces an optimal weight vector that minimises the cost function derived from the sum-squared error as described in equation (3.22).

One of the simplest training algorithms is to use all the inputs of training patterns as the centres of hidden basis functions and to calculate the linear weights based on the interpolation method. This is called the strict interpolation method [3.16] because the RBFN performs an exact mapping of all observations in the training set. In this case, the interpolation matrix becomes square due to $L = P$. The generalisation of the RBFN after training is dependent on how appropriately the hidden-layer structure (centres and the spread of radial basis functions) has been selected. However, when there are too many data points in a training set, RBFNs trained by the strict interpolation method are likely to produce an over-fitted model and the size of the interpolation matrix is also too large to be able to compute its inverse. A modified method called the orthogonal least squares learning algorithm provides a simple and efficient means for fitting RBFNs [3.31]. This solution is used to select a suitable set of centres from a larger set of candidates (training data) by incrementally searching training points that minimise the training error the most.

The other training method known as the Least Mean Square (LMS) algorithm [3.17] is more general than the former and could be applied in an online training process. In this method, the structure of the hidden layer has to be chosen before and may not be related to the training data. This LMS algorithm is a typical training solution for feed-forward networks and is similar to the back propagation method for MLPNs. In the batch training mode, the linear weights are updated one time only at each epoch. At each epoch, the mean square error (MSE) through all patterns of the training set is calculated and then the weight adjustment can be determined by

$$\Delta W_{ji} = -\eta \frac{\partial MSE}{\partial W_{ji}(k)} = -\frac{\eta}{P}\sum_{k=1}^{P} e_j(k)\frac{\partial e_j(k)}{\partial W_{ji}(k)}. \tag{3.26}$$

Finally, it can be calculated from the network errors and the outputs of the Gaussian functions by

$$\Delta W_{ji} = \frac{\eta}{P}\sum_{k=1}^{P} e_j(k).\Phi_i(k) \tag{3.27}$$

where *L, M* and *P* are number of hidden units, network outputs and training patterns respectively. η is the learning rate ($0 \le \eta \le 1$).

In the incremental training mode, the linear weights are updated after each training pattern is presented to the network. As a result, in one epoch these linear weights are updated *P* times corresponding to *P* training patterns. This is also called the Delta rule and the weight adjustment can be calculated by

$$\Delta W_{ji}(k) = \eta.e_j(k).\Phi_i(k). \tag{3.28}$$

This LMS training algorithm is simple and is only related to a learning rate η and the size of the training data. As this is a gradient descent method, the convergence of this training process is highly dependent on the selected learning rate. If the learning rate η is small, the training process will take a long time to converge to a specific goal. In contrast, if a large learning rate is adopted, it could possibly lead to an unstable learning process in which the training may never converge to a goal.

3.5.3 Using RBFNs to approximate the inverse kinematics

MLPNs as discussed earlier have some significant disadvantages, such as the slow training phase due to the nonlinear training algorithm and a lack of reasonable methods to choose the network structures. Therefore, a trend towards using RBFNs which are conceptually simpler and possess the ability to model any nonlinear function conveniently have become more popular [3.17]. This section presents several approaches using RBFNs to approximate the inverse kinematics transformation of robot manipulators.

A solution using RBFNs for inverse kinematics approximation of robot manipulators was presented in [3.32]. This work tried to explore the effect of various network configurations on the performance of the network. A variety of network configurations were developed, e.g. a single 12 inputs-6 outputs network, two 12 inputs-3 outputs network modules and six 12 inputs-1 output network modules to solve the inverse kinematics of a six D.O.F manipulator. All consisted of 12 input elements, [$\boldsymbol{n}$, $\boldsymbol{s}$, $\boldsymbol{a}$, $\boldsymbol{p}$], to represent a location (position and orientation) of the end-effector with respect to the base coordinate system. Figure 3.14 shows the architecture of the 12 inputs–6 outputs network. Simulation results showed that the position errors of the three different architectures for the test data within the training data space were all similar. However, for the test data outside the training data space, the generalisation of the six 12 inputs-1output network modules was poorer than the other two. The errors increased smoothly with increased distances from the cubical volume of the training data space and almost negligibly changed with training size. This demonstrated that the generalisation of RBFNs has a localised characteristic in the effective area of training data. As a result, the generalisation will be poorer if the network operates outside training regions.

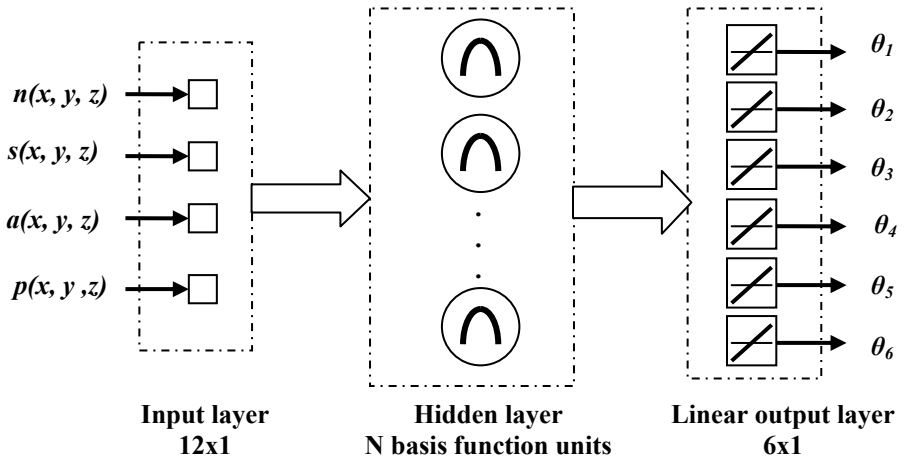


Figure 3.14 – Architecture of a 12 inputs-6 outputs network.

Another solution to implement an inverse kinematics approximation of a 6 D.O.F manipulator using RBFNs was presented in [3.33]. This solution developed a structure of six parallel RBFNs, each of which consists of six inputs, [P_x, P_y, P_z, φ, θ, ψ], which represent a location (position and orientation) of the end-effector and one output as the joint angle. Figure 3.15 presents the architecture of the 6 inputs–1 output network. Thus, the group of six parallel RBFNs (one for each joint angle) could perform an inverse kinematics approximation of a 6 D.O.F manipulator.

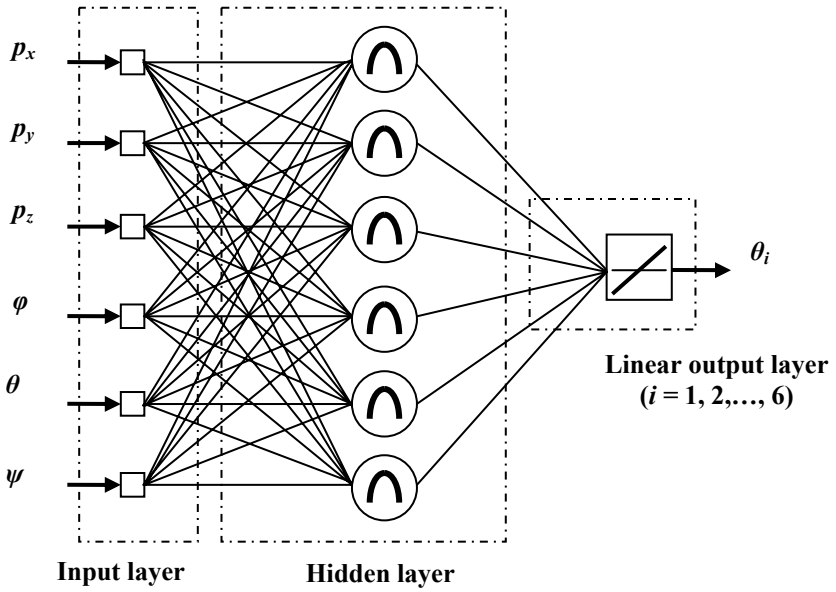


Figure 3.15 –The architecture of a 6 inputs–1 output network to approximate one joint angle of the inverse kinematics transformation.

The simulation was performed by using the Levenberg-Marquardt optimisation algorithm with 4,096 training data points created from the forward kinematics equations of the 6 D.O.F MOTOMAN manipulator. The result was compared with a similar approach using an MLPN and the back propagation training algorithm. It showed that this solution can be applied successfully to solve the inverse kinematics problem. However, it did not explain how to select the structure of hidden basis units (centres and the spread of Gaussian functions) which directly affects the generalisation of the RBFN. The network structure with only 6 basis function units seemed not enough to cover a large area of the workspace. This is contrary to the opinion that the number of hidden basis function units of an RBFN should be much higher than that of an MLPN to obtain the same performance quality [3.16], [3.17].

In [3.34] and [3.35], a comparison between an MLPN and an RBFN for the inverse kinematics problem of a 3-link manipulator was presented. In [3.34] both networks were established from the same structure, which was the 3-20-5-3 configuration shown in Figure 3.16 and used the same training data. Training of the RBFN was implemented by two independent phases, the first phase using the K-means clustering method to select cluster centres of the hidden radial basis units and the second phase using a supervised LMS algorithm to update the linear weights. However, the RBFN was a nonlinear model with more than one hidden layer which is different from the original structure of the RBFNs in [3.16]-[3.18]. As a result, the RBFN performance was poorer when compared to the MLPN due to the fact that the number of first hidden-layer centres (20 units) is too small for this complex problem. The second hidden layer made the training process slower and changed the generalisation characteristic of the RBFN. Therefore, it indicated that an RBFN often requires more hidden neurons than an MLPN to obtain equivalent performance for the same problem.

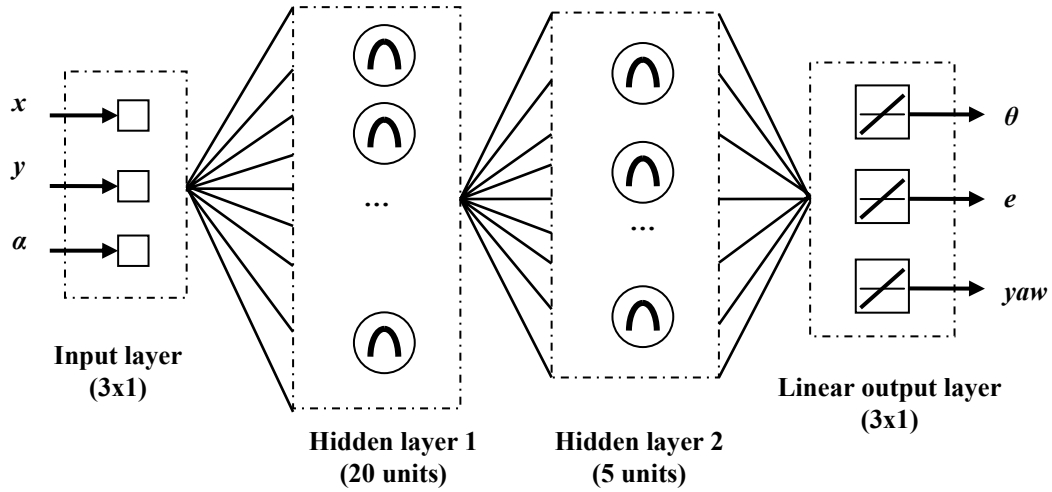


Figure 3.16 – An RBFN with two hidden basis function layers to approximate the inverse kinematics transformation of a 3-link manipulator.

Many solutions using RBFNs to solve the inverse kinematics problem have been presented as an alternative approach to MLPNs. Due to their particular structure, the training process of the RBFNs is simple. The optimal linear weights can be calculated by a linear optimisation algorithm (i.e., the interpolation method). Based on the relationship between the input space and the centres of hidden basis functions some solutions can be used to select a suitable set of centres related to the training data. This is a useful feature of RBFNs which can aid the optimal selection of the network structure and may possibly improve the network performance [3.36].

However, RBFNs have a localised generalisation characteristic in terms of centre positions and training data. Thus, for the same complex problem the required number of hidden-layer units is much higher than the number of hidden neurons of MLPNs.

3.6 Conclusion

This chapter has presented the general background of artificial neural networks and applications of neural networks to the inverse kinematics problem. The two most popular network types (MLPN and RBFN), used widely in inverse kinematics approximation, have been described. The MLPN is a universal and powerful solution for almost all functional approximation applications due to its inherent nonlinear

mapping capabilities which can deal with a wide range of process features. However, the training process is complex because it involves a nonlinear optimisation algorithm which requires an iterative procedure and there is no reasonable mechanism to select a suitable network configuration. In contrast, the RBFN training process is simple and straightforward by using a linear optimisation algorithm based on the least squares technique. There are several solutions to choose the optimal selection of the network configuration from information about the robotic system (training data) so that the generalisation can be improved. However, it seems that the performance of existing approaches (both MLPNs and RBFNs) described earlier is still insufficiently accurate and inefficient for practical applications.

For these reasons, a novel approach using an RBFN with regularly-spaced position centres has been proposed to solve the inverse kinematics problem. This solution produces an RBFN with a sufficiently small number of centres whilst achieving a satisfactory accuracy for the inverse kinematics approximation. In addition, in order to enhance the generalisation of RBFNs, the concept that the constrained training data should be collected closely to the position of centres has been suggested. The proposed approach is verified through simulations in Chapters 4 and 5 followed by practical work in Chapter 7.

3.7 References

[3.1] W. T. Miller, "Sensor-Based Control Of Robotic Manipulators Using A General Learning Algorithm", *IEEE Journal of Robotics And Automation*, vol. RA-3(2), April 1987, pp.157-165.

[3.2] K. T. Song and J. M. Chang, "Experimental Study on Robot Visual Tracking Using A Neural Controller", in *Proc. of the 22nd Int. Conf. on Industrial Electronics, Control, and Instrumentation*, 1996, pp.1850–1855.

[3.3] F.L. Lewis, "Neural Network Control of Robot Manipulators", *IEEE Expert*, vol. 11(3), June 1996, pp. 64-75.

[3.4] H. Hashimoto, T. Kubota, M. Kudou and F. Harashima, "Self-Organizing Visual Servo System Based on Neural Networks", in *Proc. of the 1991 American Control Conference*, Boston, April 1991, pp. 31-36.

[3.5] H. Hashimoto, T. Kubota, M. Baeg and F. Harashima, "A Scheme for Visual Tracking of Robot Manipulator Using Neural Network", in *Proc. of 1991 IEEE Int. Joint Conf. on Neural Networks*, vol. 2, Nov. 1991, pp. 1073 - 1078.

[3.6] D. H. Rao, M. M. Gupta and P.N. Nikiforuk, "On-Line Learning of Robot Inverse Kinematic Transformations", in *Proc. of 1993 IEEE Int. Joint Conf. on Neural Networks*, 1993, pp. 2827-2830.

[3.7] E. Oyama, N. Y. Chong and A. Agah, "Inverse Kinematics Learning by Modular Architecture Neural Networks with Performance Prediction Networks", in *Proc. of the 2001 IEEE Int. Conf. on Robotics and Automation*, Seoul, 2001, pp. 1006-1012.

[3.8] R. V. Mayorga and P. Sanongboon, "A Radial Basis Function Network Approach for Inverse Kinematics and Singularities Prevention of Redundant Manipulators", in *Proc. of the 2002 IEEE Int. Conf. on Robotics and Automation*, Washington DC, May 2002, pp. 1955-1960.

[3.9] F. Pourboghrat, "Neural Networks for Learning Inverse-Kinematics of Redundant Manipulators", in *Proc. of the 1991 Seattle Int. Joint Conf. on Neural Networks (IJCNN-91)*, vol. 2, Seattle, July 1991, pp. 1004-1007.

[3.10] G. Wu, J. Wang, "A Recurrent Neural Network for Manipulator Inverse Kinematics Computation", in *Proc. of the 1994 IEEE Int. Conf. on Neural Networks (IEEE World Congress on Computational Intelligence)*, vol. 5, Orlando, April 1994, pp. 2715–2720.

[3.11] Y. Kuroe, Y. Nakai and T. Mori, "A New Neural Network Approach to the Inverse Kinematics Problem in Robotics", in *Proc. of Asian-Pacific Workshop on Advances in Motion Control*, July 1993, pp.112-117.

[3.12] L. X. Wei, H. R. Wang and Y. Li, "A New Solution for Inverse Kinematics of Manipulator Based on Neural Network", in *Proc. of the second Int. Conf. on Machine Learning and Cybernetics*, Nov. 2003, pp. 1201-1203.

[3.13] M. Zeller and K. Schulten, "Vision-Based Motion Planning of A Pneumatic Robot Using a Topology Representing Neural Network", in *Proc. of the 1996 IEEE Int. Symposium on Intelligence Control*, Dearborn, Sept. 1996, pp. 7– 12.

[3.14] J. Barhen, S. Gulati and M. Zak, “Neural Learning of Constrained Nonlinear Transformations”, *Computer*, vol. 22(6), June 1989, pp. 67-76.

[3.15] G. Hermann, P. Wira and J. P. Urban, “Neural Networks Organisations to Learn Complex Robotics Functions”, in *Proc. of the 11th European Symposium on Artificial Neural Networks*, Bruges, Belgium, April 2003, pp. 33-38.

[3.16] S. Haykin, *Neural Networks a Comprehensive Foundation – Second Edition*. Prentice Hall, 1999.

[3.17] G. W. Irwin, K. Warwick and K. J. Hunt, *Neural Network Applications in Control*, IEE Control Engineering Series 53, 1995.

[3.18] M. J. L. Orr, *Introduction To Radial Basis Function Networks*. [Online]. Available: http://www.anc.ed.ac.uk/rbf/rbf.html. 1996.

[3.19] D. Psaltis, A. Sideris and A. A. Yamamura, “A Multilayered Neural Network Controller”, *IEEE Control Systems Magazine*, vol. 8 (2), April 1988, pp. 17-21.

[3.20] P. D. Wilde, *Neural Network Models - Second Editor*. Springer – Verlag London Limited, 1997.

[3.21] E. Watanabe and H. Shimizu, “A Study on Generalization Ability of Neural Network for Manipulator Inverse Kinematics”, in *Proc. of the 17th Int. Conf. on Industrial Electronics, Control and Instrumentation*, Kobe, Japan, vol. 2, Nov. 1991, pp. 957-962.

[3.22] A. Guez and Z. Ahmad, “Solution to The Inverse Kinematics Problem in Robotics by Neural Networks”, in *Proc. of 1988 IEEE Int. Conf. on Neural Networks*, San Diego, USA, vol. 1, July 1988, pp. 617-624.

[3.23] B. B. Choi and C. Lawrence, “Inverse Kinematics Problem in Robotics Using Neural Networks”, *NASA Technical Memorandum - 105869*, October 1992.

[3.24] Z. Binggul, H. M. Ertunc, and C. Oysu, “Comparison of Inverse Kinematics Solutions Using Neural Network for 6R Robot Manipulator with Offset”, in *Proc. of the 2005 Congress on Computational Intelligence Method & Application*, Istanbul, Turkey, Dec. 2005, pp. 1-5.

[3.25] A.S. Morris and A. Mansor, "Finding the Inverse Kinematics of Manipulator Arm Using Artificial Neural Network with Look-Up Table", *Robotica*, vol. 15, 1997, pp. 617-625.

[3.26] A. Guez and Z. Ahmad, "Accelerated Convergence In The Inverse Kinematics Via Multilayer Feedforward Networks", in *Proc. of 1989 IEEE Int. Conf. on Neural Networks*, Washington, USA, vol. 2, June 1989, pp. 341-344.

[3.27] N. Takanashi, "6 D.O.F. Manipulators Absolute Positioning Accuracy Improvement Using a Neural Network", in *Proc. of the IEEE Int. Workshop on Intelligent Robots and Systems*, Ibaraki, Japan, vol. 2, July 1990, pp. 635-640.

[3.28] P. J. Alsina and N. S. Gehlot, "Robot Inverse Kinematics: A Modular Neural Network Approach", in *Proc. of the 38th Midwest Symposium on Circuits and Systems,* Rio de Janeiro, Brazil, vol. 2, August 1995, pp. 631-634.

[3.29] B. L. Lul and K. Ito, "Regularization of Inverse Kinematics for Redundant Manipulators Using Neural Network Inversions", in *Proc. of 1995 IEEE Int. Conf. on Neural Networks*, vol. 5, Perth, USA, Dec. 1995, pp. 2726 -2731.

[3.30] J. A. Leonard, M. A. Kramer and L. H. Ungar, "Using Radial Basis Functions to Approximate a Function and Its Error Bounds", *IEEE Transactions on Neural Networks*, vol. 3(4), July 1992, pp. 624-627.

[3.31] S. Chen, C. F. N. Cowan and P. M. Grant, "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks", *IEEE Transactions on Neural Networks*, vol. 2(2), March 1991, pp. 302-309.

[3.32] J. A. Driscoll, "Comparison of Neural Network Architectures for the Modelling of Robot Inverse Kinematics", in *Proc. of the 2000 IEEE SOUTHEASTCON,* Tennessee, USA, vol. 3*,* April 2000, pp. 44-51.

[3.33] P. Y. Zhang, T. S. Lu and L. B. Song, "RBF Networks-Based Inverse Kinematics of 6R Manipulator", *Int. Journal of Advanced Manufacturing Technology*, Springer – Verlag London Ltd., vol. 26, 2004, pp. 144-147.

[3.34] S.S. Yang, M. Moghavvemi and John D. Tolman, "Modelling of Robot Inverse Kinematics Using Two ANN Paradigms", in *Proc. of TENCON2000 - Intelligent System and Technologies for the New Millennium,* Kuala Lumpur, Malaysia, vol. 3, Sept. 2000, pp. 173-177.

[3.35] A. S. Morris and M. A. Mansor, "Manipulator Inverse Kinematics Using an Adaptive Back Propagation Algorithm and Radial Basis Function With a Lookup Table", *Robotica*, vol. 16(4), 1998, pp. 433- 444.

[3.36] A. Ghodsi and D. Schuumans, "Automatic Basis Selection Technique for RBF Networks", *Neural Networks - Special issue: Advances in Neural Networks Research (IJCNN'03)*, vol. 16, June 2003, pp. 809-816.

CHAPTER 4
INVERSE KINEMATICS APPROXIMATION USING A RADIAL BASIS FUNCTION NETWORK

4.1 Introduction

In Chapter 3, several existing approaches using neural networks to approximate the inverse kinematics of robotic manipulators were discussed. However, it is likely that most of these approaches are still not suitable for practical applications because the accuracy of the networks' performances is limited. A radial basis function network (RBFN) may be more advantageous than other networks since its hidden layer structure can be chosen with regards to the training data. If appropriate training data can be collected throughout the whole input space, it is possible to select the optimal structure of the RBFN so that the network's performance could then be improved in the operational phase. Most of the solutions using RBFNs for determination of the inverse kinematics have used training data collected arbitrarily, or regularly, in the joint angle space (the output space of the networks) [4.1]-[4.3]. The training data has not reflected the full characteristics of the inverse kinematics function in the whole workspace. Consequently, the network's performance has not been optimal in the operational phase. This chapter presents a novel solution using RBFNs to approximate the inverse kinematics of robotic manipulators. This approach has some fundamental principles: centres of hidden-layer units are regularly distributed in the workspace, constrained training data is used where inputs are collected approximately around the centre positions in the workspace and the training phase is performed using either strict interpolation or the least mean square algorithm.

The chapter first describes the main concepts of the proposed approach. A simple example is presented to explain why regularly-spaced position centres can produce an acceptable approximation to the inverse kinematics function. Simulations for two-link and three-link manipulators are then presented to demonstrate the proposed approach. Finally, conclusions are presented following analysis of the simulation results.

4.2 Using RBFNs to approximate the inverse kinematics of robotic manipulators

The accuracy of a neural network function approximation depends on three main factors: the structure of the network, the training method and the training data. To enhance the performance of an RBFN for the inverse kinematics approximation, a new approach is proposed:

- using regularly-spaced position centres as a predefined structure of the RBFN,
- using constrained data for the training phase (this constrained training pattern is collected around centre positions with a reasonable degree of accuracy),
- using strict interpolation, or the least mean square (LMS) algorithm, to update the linear weights.

The main concepts of this proposed approach are described in the following sections.

4.2.1 Selection of the hidden layer parameters

To illustrate the idea that using an RBFN with regularly-spaced position centres can produce a better approximation for a desired function, an example is presented as follows.

Given a nonlinear function

$$f(x) = \sin(2x) + 0.1x \tag{4.1}$$

an RBFN is used to produce an approximation of this function by a linear model

$$y = \sum_{i=1}^{L} W_i . \Phi_i(x) \tag{4.2}$$

where $\Phi_i(x)$ is the output of the i^{th} hidden unit, W_i is the interconnection weight between the i^{th} hidden unit and the network output and L is the number of hidden units.

A training method called strict interpolation [4.7] is used in which centres of hidden-layer units are taken from inputs of a training set. This method creates as many hidden units as there are inputs of the training set. Thus, the generalisation of the trained network is dependent on the distribution of the training set and the spread of the Gaussian basis function. This spread value should be large enough compared to the distances between the centres of hidden units so that the active input regions of the radial basis function neurons overlap sufficiently. This makes the network function smoother and results in better generalisation for a new input occurring between centre positions. However, the spread should not be too large because it can produce a poor discrimination between radial basis functions in the effective area.

In this example, two different training data sets, an arbitrary and a regularly constrained distribution, are used in the training phase. The arbitrary set uses inputs which are randomly created by the function **rand** (in MATLAB) in the range [-3, 3] and consists of the input vector $\boldsymbol{P}_1$ and target output $\boldsymbol{T}_1$. These are:

$\boldsymbol{P}_1$ = [-2.46, - 2.29, -1.37, -1.33, -0.06, 0.53, 1.22, 1.85, 2.21, 2.38, 2.75, 2.77, 2.81];

$\boldsymbol{T}_1$ = [0.73, 0.76, -0.53, -0.60, -0.13, 0.92, 0.77, -0.34, -0.74, -0.76, -0.43, -0.41, -0.34].

Setting the spread value to 0.5 and using the strict interpolation method, the RBFN is built to approximate the function $f(x)$ according to the training set $\{\boldsymbol{P}_1, \boldsymbol{T}_1\}$. Figure 4.1 presents the network performance after training. The result shows that the generalisation of the network is not the same in all regions of the workspace because the set of hidden-layer centres is randomly distributed in the input space. When x is between zero and three, the function approximation is close to the desired function. However, in other regions, due to the lack of necessary neurons, the function approximation is poor. As the centres of the RBFN are randomly distributed, the network cannot perform well in the whole workspace.

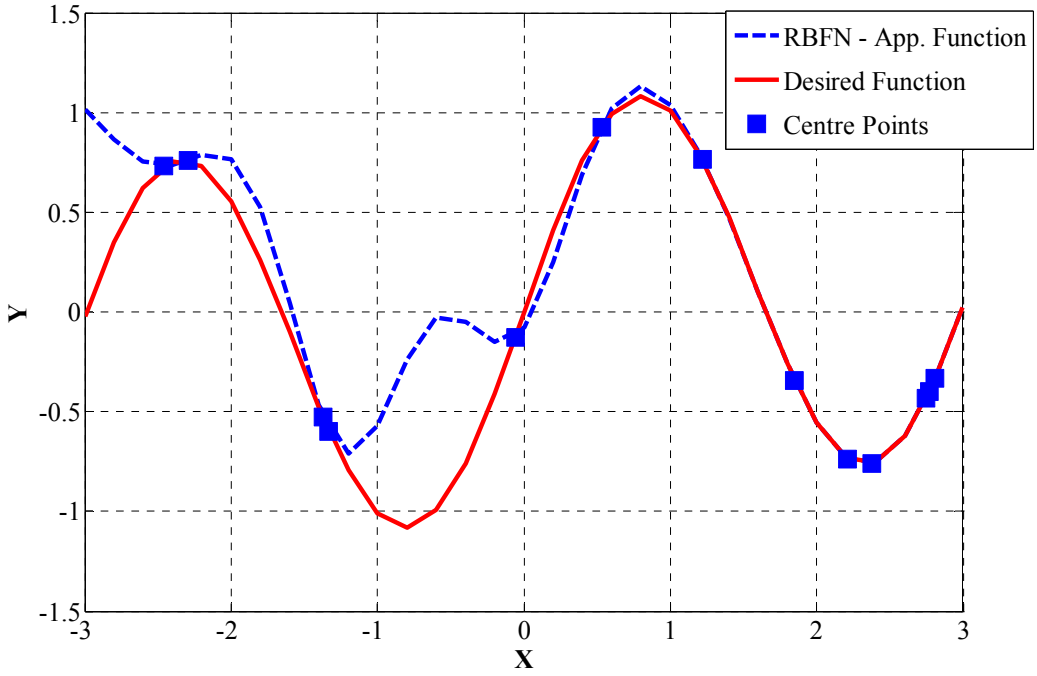


Figure 4.1- The network's performance with a set of randomly distributed centres.

Another training set is collected as regularly-spaced points in the input space. It contains:

$\boldsymbol{P}_2$ = [-3, -2.5, -2, -1.5, -1, - 0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3];

$\boldsymbol{T}_2$ = [-0.0206, 0.7089, 0.5568, -0.2911, -1.0093, -0.8915, 0, 0.8915, 1.0093, 0.2911, -0.5568, -0.7089, 0.0206].

The distance between elements of the input vector $\boldsymbol{P}_2$ is 0.5 and the spread of the Gaussian functions is chosen to be the same value. The strict interpolation method is then used to build an RBFN to approximate the function $f(x)$ according to the training set $\{\boldsymbol{P}_2, \boldsymbol{T}_2\}$. Figure 4.2 shows that the function approximation almost perfectly fits the desired function over the whole operating space. The generalisation of the network where the centres are regularly distributed in the input space is much better than the network where the centres are in randomly spaced positions.

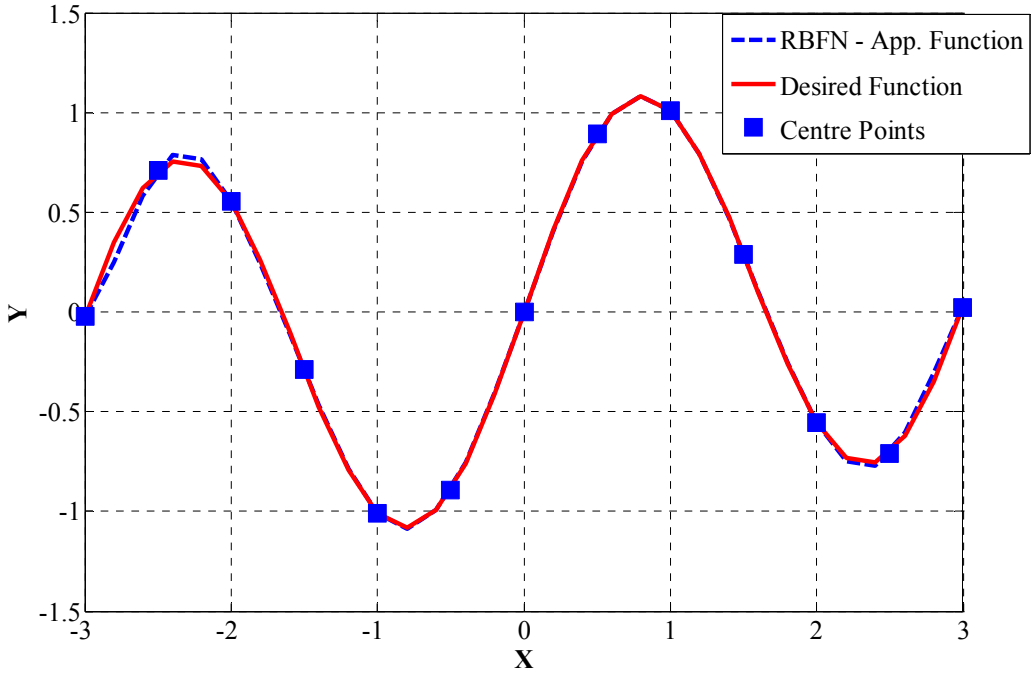


Figure 4.2 – The network's performance with a set of regularly distributed centres.

As demonstrated above, it is clear that an RBFN could produce an appropriate approximation of the inverse kinematics function if the hidden unit centres are regularly-spaced positions in the workspace. Figure 4.3 presents typical examples of regularly-spaced position centres in two- and three-dimensional spaces.

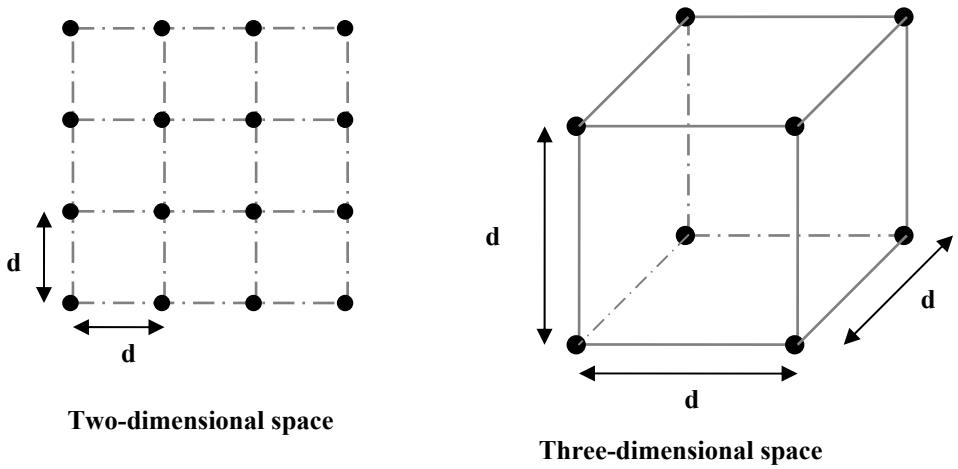


Figure 4.3 - Typical examples of regularly-spaced position centres in the workspace.

In this research, the distance between centres can be determined in a heuristic trial and error manner. The smaller the distance between centres, the better the performance of the RBFN. However, due to the limited computer memory capacity and the computational complexity, the number of hidden layer units must be limited to a sensible value so that a feasible training process can be implemented. Therefore, a reasonable choice for the distance between centres should be investigated carefully through trial and error experiments.

As the centres of the hidden layer units (Gaussian functions) are regularly distributed in the workspace, these functions should have the same spread. The spread value affects the smoothness of the network by varying the local-filter feature of the hidden units. For a specific application, the spread value should also be tailored and estimated through trial and error experiments.

4.2.2 Training methods

Once the structure of the hidden layer has been chosen, the second phase of supervised training is used to adjust the linear weights. As a linear relationship between the linear weights and the network outputs exists, the training process of an RBFN is simple and straightforward using the least squares approach. Therefore, the training of an RBFN is easier and faster compared to a multi-layer perceptron network for the same application [4.4], [4.5]. As presented in Chapter 3, there are two popular training methods, strict interpolation and LMS algorithms, to train the network.

When a training set with N patterns (input - target output), $\{(X_1, X_2,\ldots, X_N); (T_1, T_2,\ldots, T_N)\}$, is presented to an RBFN, using the strict interpolation method an optimal set of the linear weights is determined so that the cost function (i.e., sum of squared errors between target and actual outputs of the network) is minimised. The centres of hidden layer units can be either the same as, or different from, the inputs of the training data. However, their number must be the same to produce an exact mapping $f : X_k \rightarrow T_k$ for all training data presented to the RBFN [4.6]. The set of the linear weights, $\boldsymbol{W}$, is calculated by

$$\boldsymbol{W} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{T} \tag{4.3}$$

where $\boldsymbol{T}$ is the target output vector and $\boldsymbol{\Phi}(N\text{x}N)$ is the interpolation matrix, where each row corresponds to the responses of all hidden units for each pattern and each column corresponds to each hidden unit through all patterns.

The training phase using the strict interpolation method is simple and fast. It produces a unique set of the linear weights that minimises the cost function. However, the number of hidden units is limited due to the computational burden of the matrix inversion algorithm. For some cases when the amount of training data is higher than the number of hidden units, equation (4.3) is also used to calculate the linear weights. However, it cannot produce an exact mapping for all training data presented to the network [4.4].

The LMS algorithm uses the gradient descent technique to iteratively update the linear weights in a batch training mode [4.6]. At each training epoch, the linear weights are updated in a direction that reduces the MSE (mean square error of the network outputs) through all patterns of the training set. Assuming that a training set with N patterns is presented to the RBFN, the adjustment of linear weights can be calculated by

$$\Delta W_{ji} = \frac{\eta}{N}\sum_{k=1}^{N} e_j(k).\Phi_i(X_k) \tag{4.4}$$

where $\Phi_i(X_k)$ is the output of the i^{th} Gaussian function corresponding to stimulation of input X_k , $e_j(k)$ is the error at the network output j with the k^{th} target output and η is the learning rate ($0 \le \eta \le 1$).

The LMS algorithm can be used to train RBFNs with either arbitrary or constrained training data without any restriction in the number of hidden units and/or training patterns. This training process is simple and related to the value of learning rate η and the size of the training data set. If the learning rate η is small, the training process will take a long time to converge to a specific goal. In contrast, if a large learning rate is adopted, it could possibly lead to a divergent learning process.

4.2.3 Training data

Two kinds of training data were used to train the RBFN in the simulations. The first is called constrained data because their inputs coincide with the centres which are pre-defined as regularly-spaced positions in the workspace. The other is collected randomly around the centres' positions. This data is also limited by setting a maximum deviation from the centre position. For example, if a set of centres is predefined as in Figure 4.3, a random training data set can be collected around the centres' positions as

$$\begin{aligned} P_{\mathrm{x}} &= (2.\mathrm{rand} - 1).MaxDev + C_{\mathrm{x}} \\ P_{\mathrm{y}} &= (2.\mathrm{rand} - 1).MaxDev + C_{\mathrm{y}} \\ P_{\mathrm{z}} &= (2.\mathrm{rand} - 1).MaxDev + C_{\mathrm{z}} (\text{for the three - dimensional space}) \end{aligned} \tag{4.5}$$

where *MaxDev* is the maximum deviation, **rand** is a random distribution function (MATLAB) in the range [0, 1] and $\{C_{\mathrm{x}}, C_{\mathrm{y}}, C_{\mathrm{z}}\}$ are the coordinates of the centres.

For a two-link manipulator, the maximum deviation should not be higher than 30% of the centre distance to enable the training phase to produce an appropriate inverse kinematics approximation. For a three-link manipulator, due to the more complex configuration, the maximum deviation should not be higher than 20% of the centre distance.

4.3 Simulation for a two-link manipulator

Figure 4.4 presents the two-link manipulator used in this simulation. It consists of two revolute joints and two links that have the same length of 50 mm. Two coordinate values *x*, *y* describe the position of the tip of the manipulator with respect to the base coordinate frame. The forward kinematics is a mapping from a set of joint angles to the corresponding position in Cartesian space (workspace) and is defined by

$$x = l_1.\cos(\theta_1) + l_2.\cos(\theta_1 + \theta_2) \tag{4.6}$$

$$y = l_1.\sin(\theta_1) + l_2.\sin(\theta_1 + \theta_2) \tag{4.7}$$

where l_1 and l_2 are the lengths of link 1 and link 2 respectively.

The inverse kinematics can be described by

$$\theta_2 = \text{Atan}\,2\left(\pm\sqrt{1-\left(\frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2} \right)^2}, \left(\frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2} \right) \right) \tag{4.8}$$

$$\theta_1 = \text{Atan}\,2(y,x) - \text{Atan}\,2(l_2 \sin\theta_2, l_1 + l_2\cos\theta_2). \tag{4.9}$$

Atan2*(y, x)* is defined as

$$\text{Atan}\,2(\text{y},\text{x}) = \begin{cases} 0^0 \le \theta \le 90^0, & x, y \ge 0 \\ 90^0 \le \theta \le 180^0, & x \le 0, y \ge 0 \\ -180^0 \le \theta \le -90^0, & x, y \le 0 \\ -90^0 \le \theta \le 0^0, & x \ge 0, y \le 0. \end{cases} \tag{4.10}$$

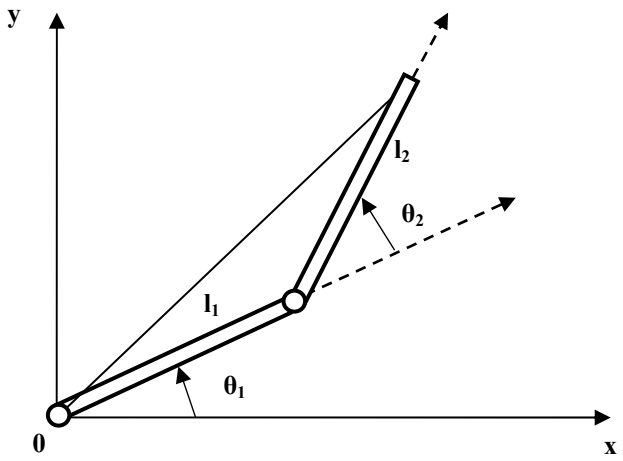


Figure 4.4 - Configuration of a two-link manipulator.

Using equations (4.8) and (4.9), training and test data were created for the simulation. Due to the periodic characteristic of the inverse kinematics function, there is a rapid change in the function θ_1, between -180^0 and 180^0. This singular region is in the area around $\{x = 0, y < 0\}$. However, because of the mechanical configuration limitations

(joint angle movement range) for a real manipulator, this singular region may not occur in the workspace. Furthermore, the plug sign in equation (4.8) is adopted for the expression of θ_2. This corresponds with the lower-elbow structure of the two-link manipulator. In this simulation, the workspace is limited to a quarter of the plane where $x > 0$ and $y > 0$, as shown in Figure 4.6.

4.3.1 Simulation description

An RBFN was used to approximate the inverse kinematics function of this two-link manipulator. Figure 4.5 shows the network configuration consisting of two inputs and two outputs to perform a transformation from the world space (x, y) to the joint angle space (θ_1, θ_2). The simulation was implemented according to the following procedure:

- The structure of the hidden layer was built with pre-defined centres regularly distributed in the workspace (e.g., 10 mm x 10 mm grids). The spread was experimentally selected so that the RBFN can produce an appropriate inverse kinematics approximation.

- Training patterns $\{(x, y); (\theta_1, \theta_2)\}$ were collected as either constrained or random data in the workspace. There were three sets of training data used for this simulation (e.g., Figure 4.7 presents the distribution of data with a centre distance of 10 mm). A set of constrained data whose inputs were coincident with the centres of hidden units was collected. Two others were randomly collected around centre positions with a maximum deviation of 3 mm and 4 mm.

- The linear weights were adjusted by one of two methods, strict interpolation or LMS.

- The RBFN performance was tested by presenting a set of new data that is different from the training data. At this stage, two independent test data sets, a trajectory inside (test trajectory 1) and a trajectory near the edge (test trajectory 2) of the workspace (Figure 4.6), were presented to the network.

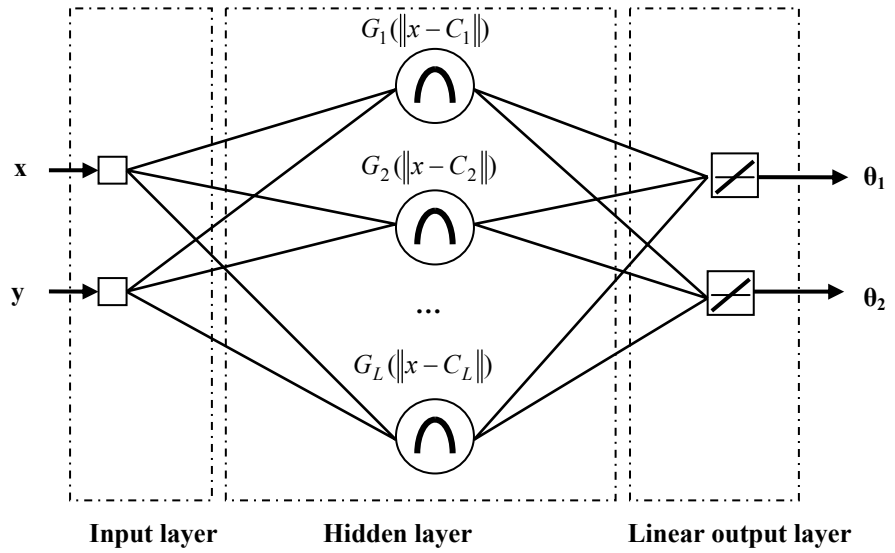


Figure 4.5 - Configuration of the RBFN to approximate the inverse kinematics of the two-link manipulator.

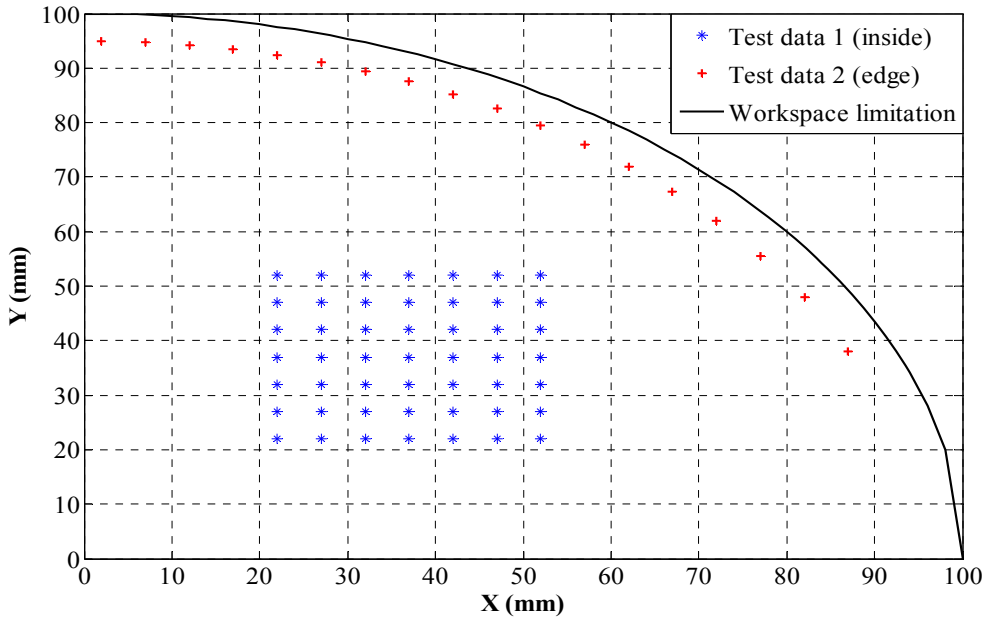


Figure 4.6 – Test trajectories for the two-link manipulator simulations.

The simulation investigated the network's performance for various conditions: two training methods with three different training data sets (constrained and random with a maximum deviation of 3 mm and 4 mm) and a variety of spread values (e.g., 6 - 28 mm). To verify the network's performance, the root mean square error (RMS) between

joint angles produced by the network and the desired inverse kinematics function (mathematical expressions) and the mean absolute errors (MAEs) between desired and actual positions in X and Y directions were calculated for each condition. The results of the three training cases with different training data are plotted in the same figure to compare the effect of training data on the network's performance. All simulation results are listed in tables (Appendix B) where the columns show performance criteria and the rows are the spread values.

4.3.2 Simulation results

Figure 4.7 shows the distribution of three different training data sets versus the position of the hidden unit centres in the workspace. This case corresponds to an RBFN with a centre distance of 10 mm (regularly-spaced distribution). The total number of hidden units is 111 nodes.

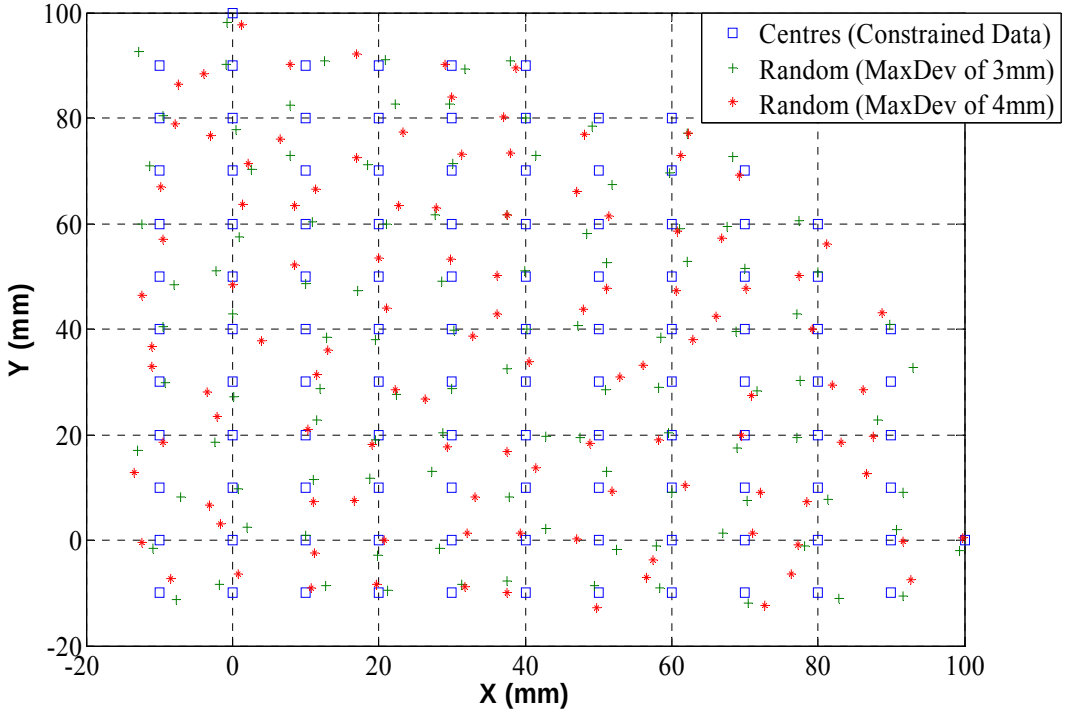


Figure 4.7 – Distribution of three training data sets in the workspace (10 mm distance).

Figures 4.8 and 4.9 present the network performance using test trajectory 1 after training by the strict interpolation method for various spread values (Appendix B.1).

Figures 4.10 and 4.11 present the performance of the same network using test trajectory 2.

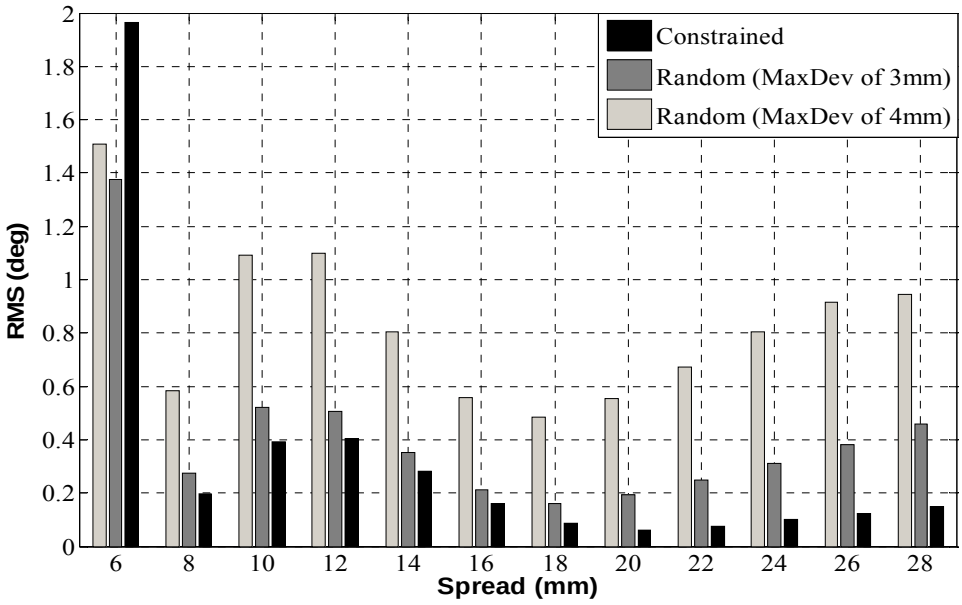


Figure 4.8 – Performance results for test trajectory 1 (inside).

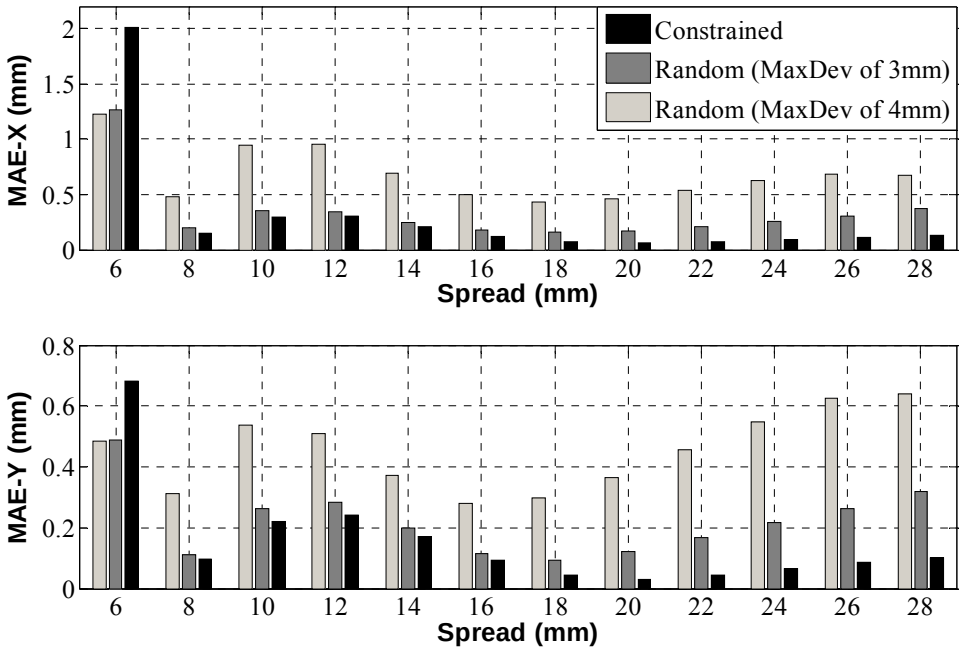


Figure 4.9 – MAEs for test trajectory 1 (inside).

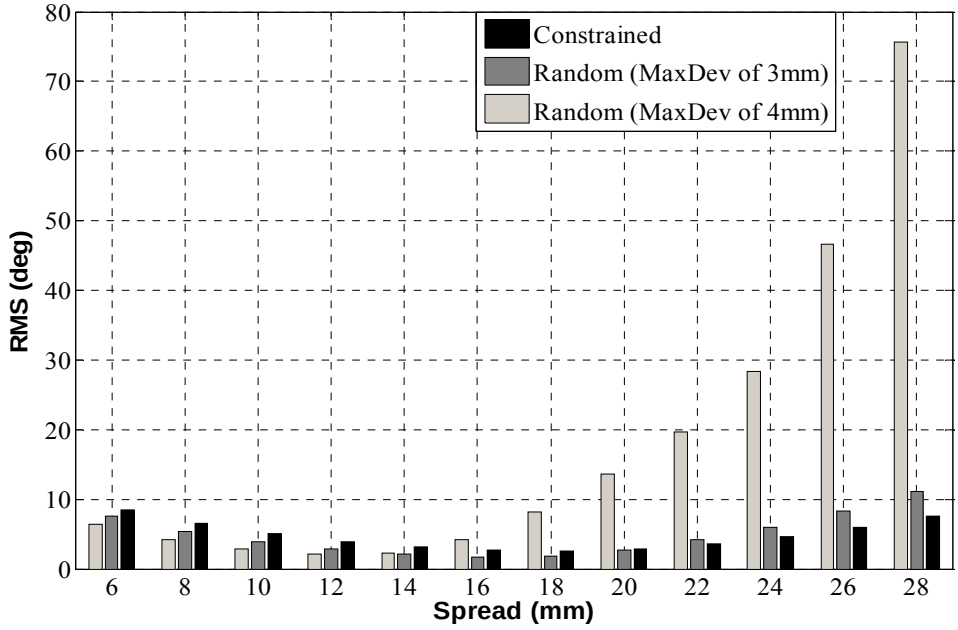


Figure 4.10 - Performance results for test trajectory 2 (edge).

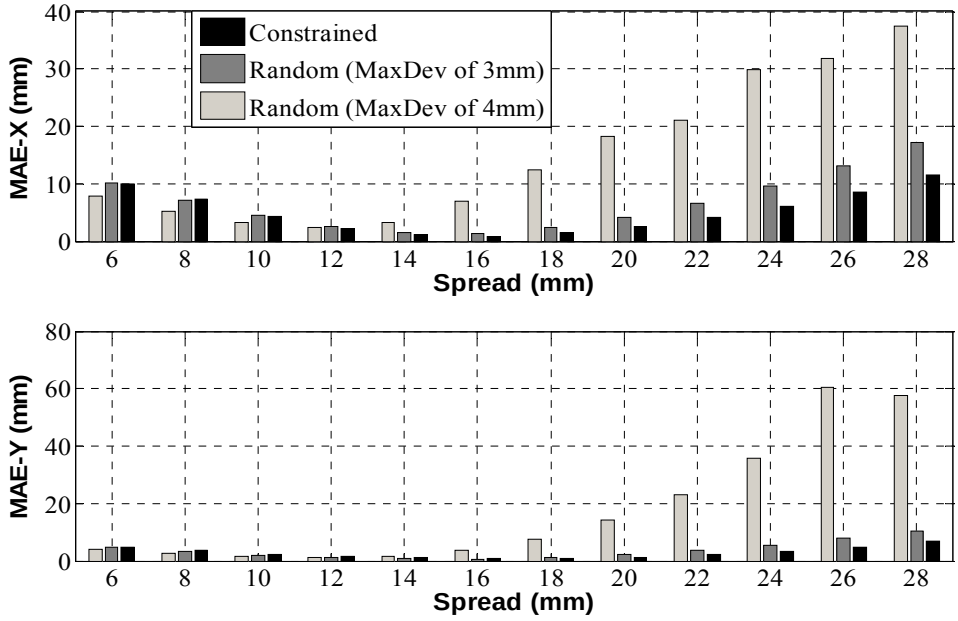


Figure 4.11 - MAEs for test trajectory 2 (edge).

The results show that the RBFN trained by the strict interpolation method produce an appropriate approximation of the inverse kinematics function. For test trajectory 1 (Figure 4.9), using the constrained data produces the best performance in which the average MAE (of MAE_X and MAE_Y) is approximately 0.1 mm for spreads between

16 and 22 mm. Using the random data with a maximum deviation of 3 mm also produces a good performance where the average MAE is approximately 0.2 mm for spreads between 16 and 22 mm. When using the random data with a maximum deviation of 4 mm, an average MAE of 0.5 mm is achieved for the same range of spread values. The performance of the RBFN is better for test trajectory 1 (inside the workspace). This is because at the edge of workspace, the network does not have enough hidden radial basis functions to be able to create appropriate responses for test trajectory 2. For example, the best performance is about 1 mm when using the constrained data with a spread between 12 and 16 mm as shown in Figure 4.11. This reflects the local generalisation characteristics of the RBFN. Varying the spread leads to differences in the performance. An increase in the spread value can improve the network performance (decrease in the RMS error and MAEs). However, the performance will become poorer if the spread is increased significantly, especially for test trajectory 2. There is a spread value between 16 and 22 mm that can produce an optimal inverse kinematics approximation for both test trajectories when training with any of the three training data sets.

Another simulation was performed corresponding to an RBFN with a centre distance of 15 mm. Figure 4.12 presents the distribution of three different training sets.

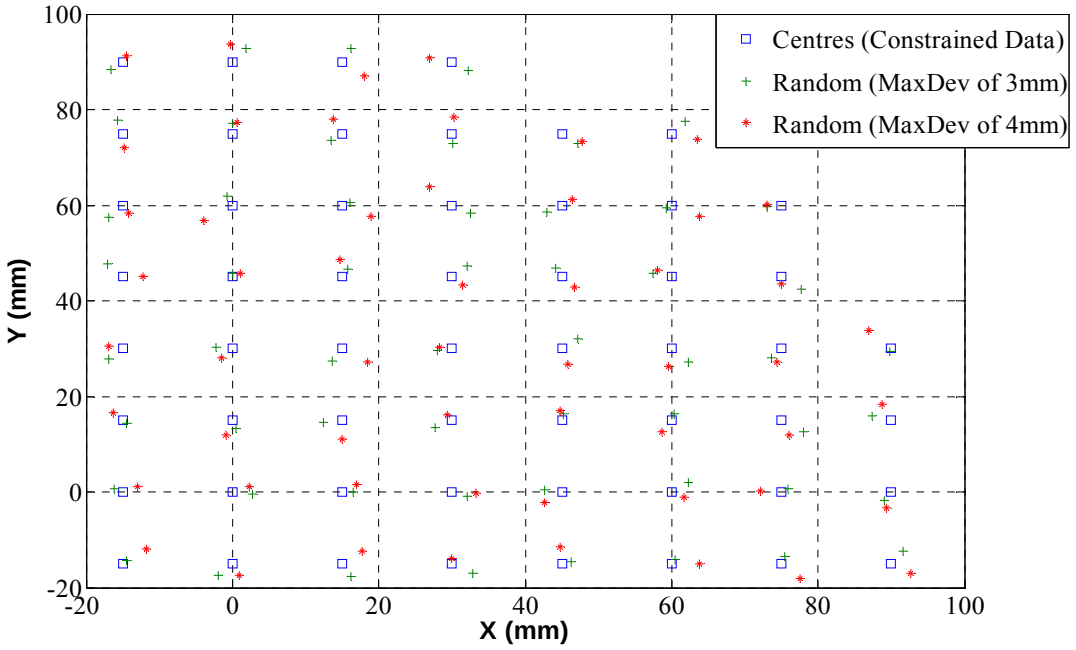


Figure 4.12 - Distribution of three training data sets in the workspace (15 mm distance).

Figures 4.13 and 4.14 present the network's performance for test trajectory 1 after training by the strict interpolation method corresponding to various spread values (Appendix B.3). Figures 4.15 and 4.16 present the performance of the same network for test trajectory 2.

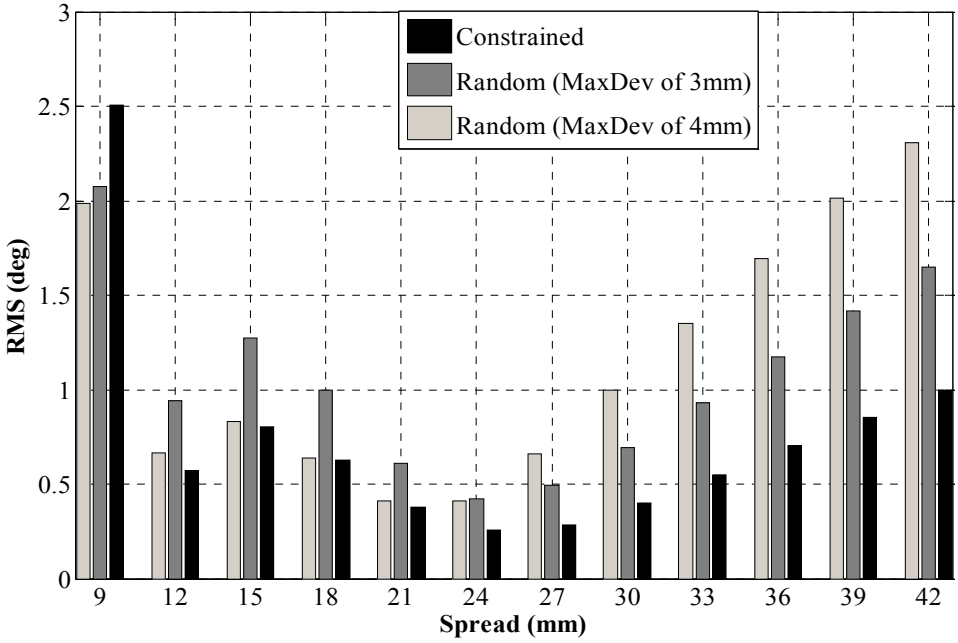


Figure 4.13 - Performance results for test trajectory 1 (inside).

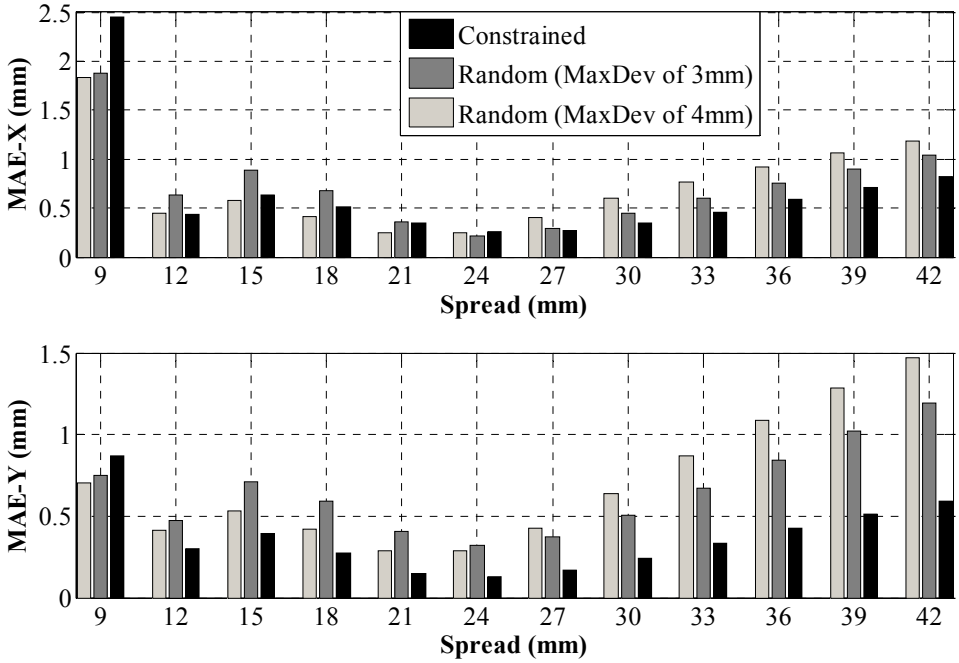


Figure 4.14 - MAEs for test trajectory 1 (inside).

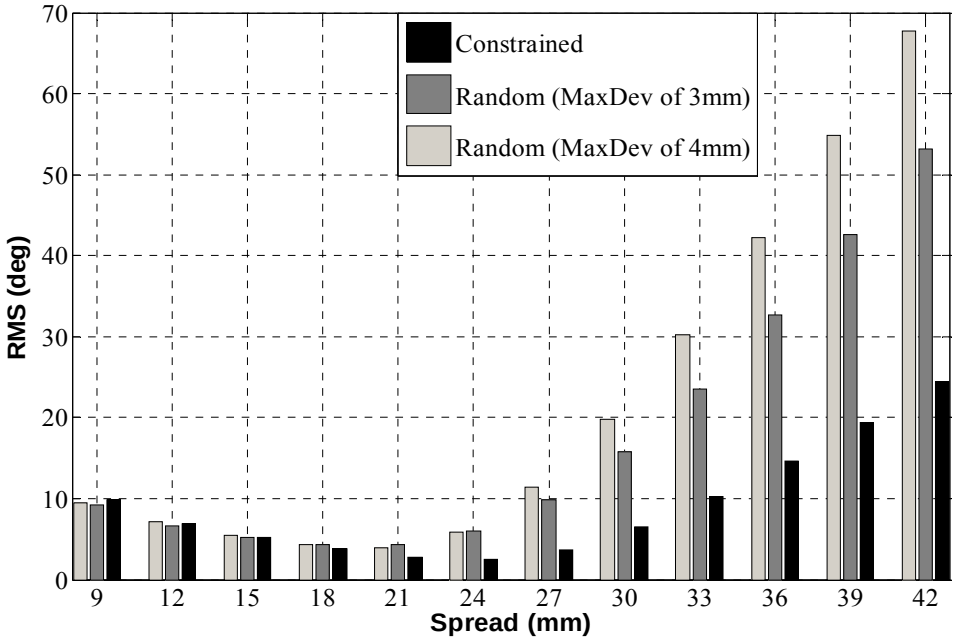


Figure 4.15 – Performance results for test trajectory 2 (edge).

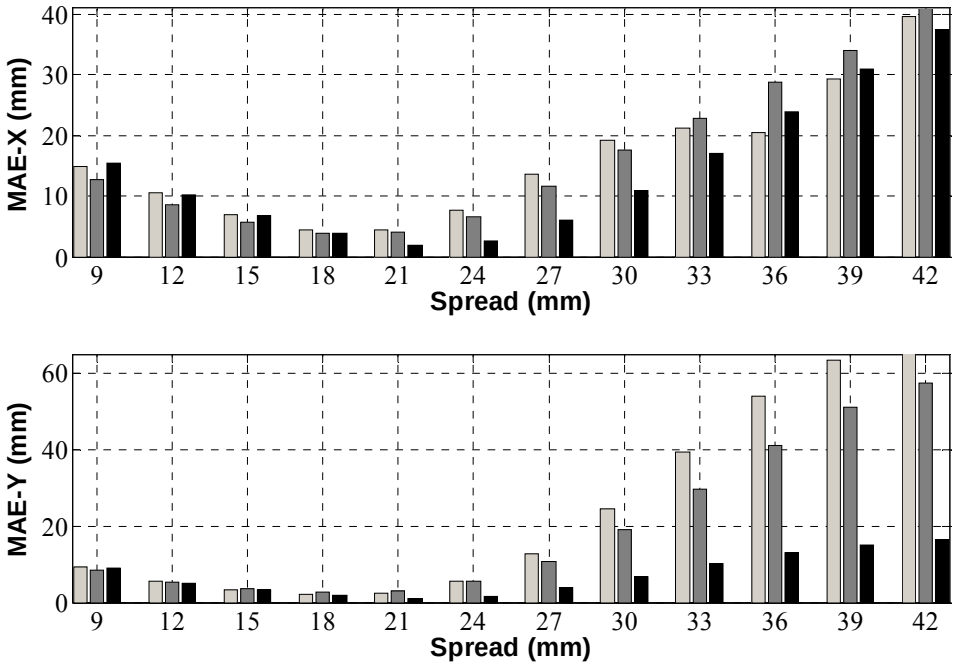


Figure 4.16 - MAEs for test trajectory 2 (edge).

Compared to the results using a centre distance of 10mm (Figures 4.8 to 4.11), it is clear that the generalisation in this case (centre distance of 15 mm) is poorer. However, this performance is still acceptable and the number of hidden units is significantly reduced (56 points compared to 111 points for the previous case). For test trajectory 1 (Figure

4.14), when using the constrained data, an average MAE of approximately 0.3 mm can be obtained for spreads between 21 to 30 mm. Using the random data produces a slightly poorer performance (average MAE of approximately 0.5 mm) for the same spread range. For test trajectory 2 (Figure 4.16), similar to the results using a centre distance of 10mm, the errors significantly increase if the spread increases. The best performance can be obtained with a spread value between 21 and 27 mm for both test trajectories when training with any of the three training data sets.

A further simulation was performed using the same network and training data as in the first case (centre distance of 10 mm) using the LMS algorithm. The training process was implemented with the following parameters:

- Maximum training epochs = 500000,
- Goal = 0.0001,
- Learning rate = 0.01 – 0.001.

As the LMS algorithm is an iterative gradient descent technique, the training time was significantly greater compared to the strict interpolation method. When the spread increases, the learning rate has to decrease correspondingly to keep the training process stable. The training time and training result (the final training performance at the maximum epoch) are thus slower and poorer for a larger spread value.

Figures 4.17 and 4.18 present the network's performance for test trajectory 1 after training by the LMS algorithm corresponding to various spread values (Appendix B.2). Figures 4.19 and 4.20 present the performance of the same network for test trajectory 2.

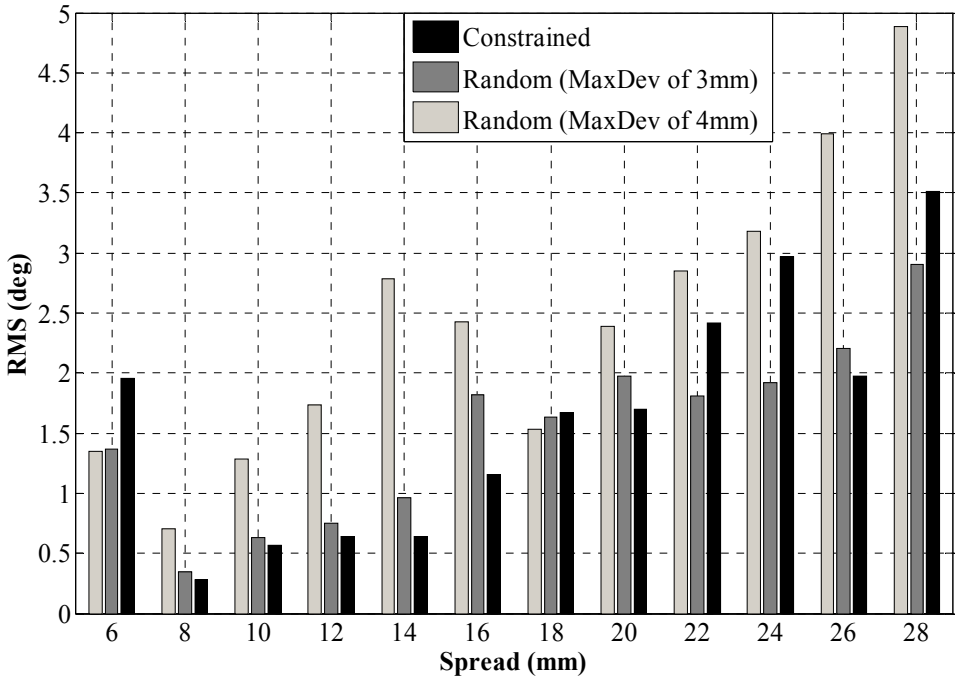


Figure 4.17 - Performance results for test trajectory 1 (inside).

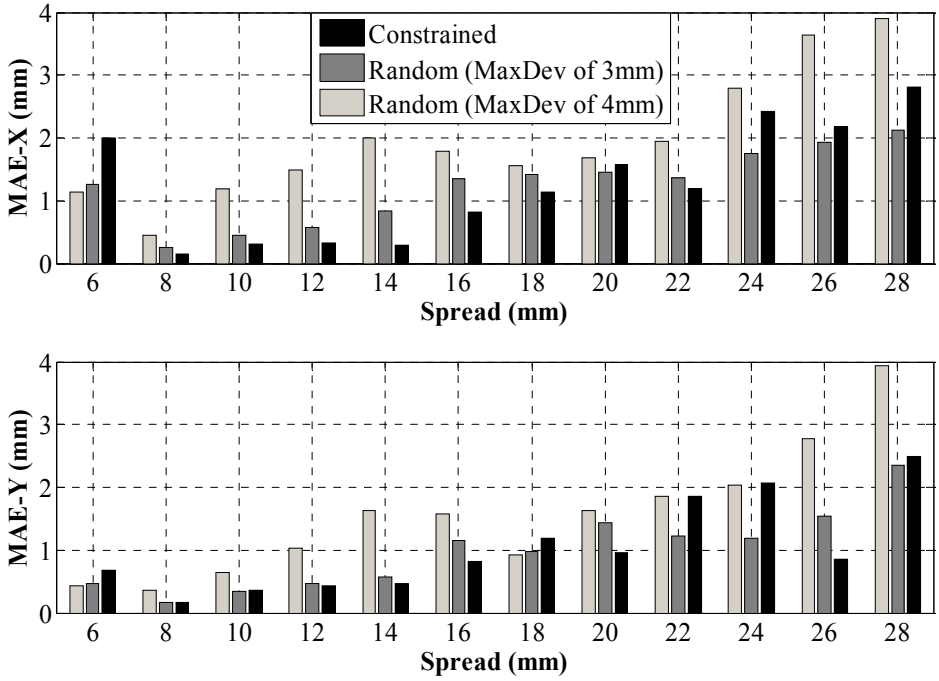


Figure 4.18 - MAEs for test trajectory 1 (inside).

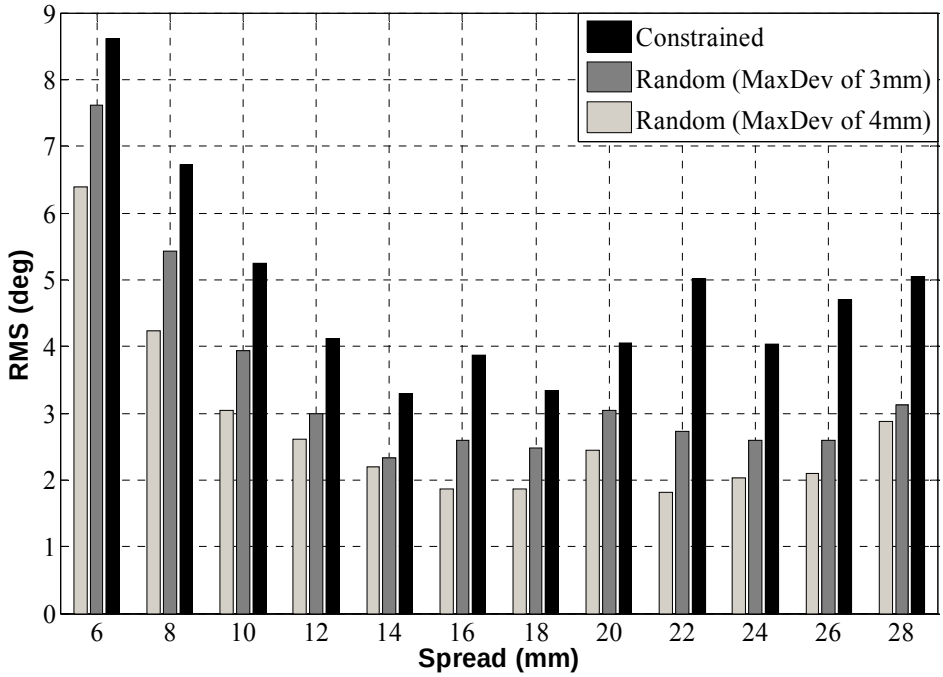


Figure 4.19 - Performance results for test trajectory 2 (edge).

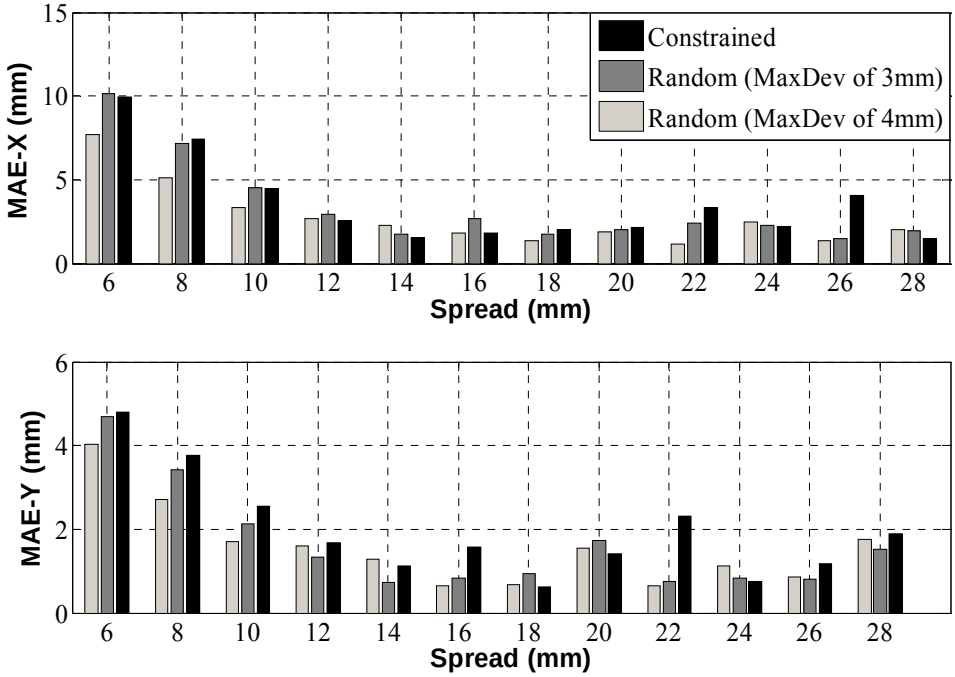


Figure 4.20 - MAEs for test trajectory 2 (edge).

For test trajectory 1 (Figure 4.18), the best performance is achieved with a spread of 8 mm (an average MAE of approximately 0.2 mm when using the constrained data).

When the spread is slightly increased (greater than 10 mm), the performance errors increase. In addition, the performance of the network trained with random training data does not significantly differ from the performance of the network trained with constrained data. In fact the network trained with constrained data has a poorer performance compared to the network trained with random data for test trajectory 2 (Figure 4.20). Thus, the effect of using constrained or random data on the network's performance is not significant when training using the LMS algorithm. For both test trajectories, using the LMS algorithm requires a smaller spread value to obtain the best performance compared to the strict interpolation method. In general, the generalisation capability in this case is poorer compared to the strict interpolation method.

4.3.3 Summary of results

The idea of using an RBFN with regularly-spaced position centres has produced an excellent approximation of the inverse kinematics function for a two-link manipulator. The average MAE (of X and Y directions) of the network with a centre distance of 10 mm is approximately 0.1 mm (or 1% of the centre distance) for a test trajectory inside the workspace. This corresponds to an approximate RMS error smaller than 0.1 degrees (Figures 4.8 and 4.9). The performance of the RBFN is poorer at the edge of the workspace. These results are significantly better compared to other relevant approaches [4.1], [4.2], [4.9], [4.10]. For example, in [4.1] the position errors of the best results varied from 2 to 7 (unknown units) for an RBFN with the test trajectory inside the trained area (cubical volume). Similarly, in [4.2] the network performance (RBFN) was even poorer with an approximate position error of 30 mm for a SCARA robot with the range of -300 mm $\leq$ x $\leq$ 300 mm and 0 mm $\leq$ y $\leq$ 700 mm. For an MLPN [4.10], the joint angle error of the best results was about 1.6 – 2 degrees (as shown in figure) for a two-link manipulator. Similarly, in [4.9] the best RMS error result for the simulation of a six D.O.F manipulator was about 8 degrees (average of all six outputs).

The two different methods (strict interpolation and LMS) used in the training process, in spite of the same network structure and training data, produce a different set of linear weights. Therefore, their performances are not the same and the RBFN trained by the strict interpolation method produces a better performance.

As the centres are regularly distributed in the workspace, when the distance between the centres decreases, the generalisation of the RBFN becomes better. However, this leads to a more complex training phase because the number of hidden layer units increases significantly. Using the strict interpolation method, the number of hidden units is limited due to the computational requirement of the matrix inversion algorithm. Therefore, a suitable choice of centre distance should be carefully considered.

A large spread can produce a smooth approximation function of the actual inverse kinematics function of the two-link manipulator. However, a large spread makes the training process using the LMS algorithm become extremely slow because it requires a small learning rate to allow convergence (gradually reducing the mean square error) of the gradient descent technique.

The closer the training data to the centre positions, the better the inverse kinematics approximation. The RBFN trained with a set of random data where the maximum deviation is no higher than 30% of the centre distance also produces good results. However, using constrained or random data does not significantly affect the network's performance when using the LMS algorithm.

4.4 Simulation for a three-link manipulator

Figure 4.21 presents the three-link manipulator used in these simulations. This is the same structure as the manipulator presented in Chapter 2.

As presented in Section 2.3, the inverse kinematics solutions of joint angles from the space coordinates (x, y, z) can be described as

$$\theta_1 = \text{Atan}\,2(y, x) \tag{4.11}$$

$$\theta_3 = \text{Atan}\,2\left(\pm\sqrt{1-K^2}, K\right) \tag{4.12}$$

$$\theta_2 = \text{Atan}\,2\left(\pm\sqrt{1-K_\Phi^2}, K_\Phi\right) - \text{Atan}\,2\left(\sqrt{x^2+y^2}, z-d_1\right) - \theta_3 \tag{4.13}$$

where

$$K = \frac{x^2 + y^2 + (z - d_1)^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{4.14}$$

$$K_{\Phi} = \frac{x^2 + y^2 + (z - d_1)^2 + a_3^2 - a_2^2}{2a_3\sqrt{x^2 + y^2 + (z - d_1)^2}}. \tag{4.15}$$

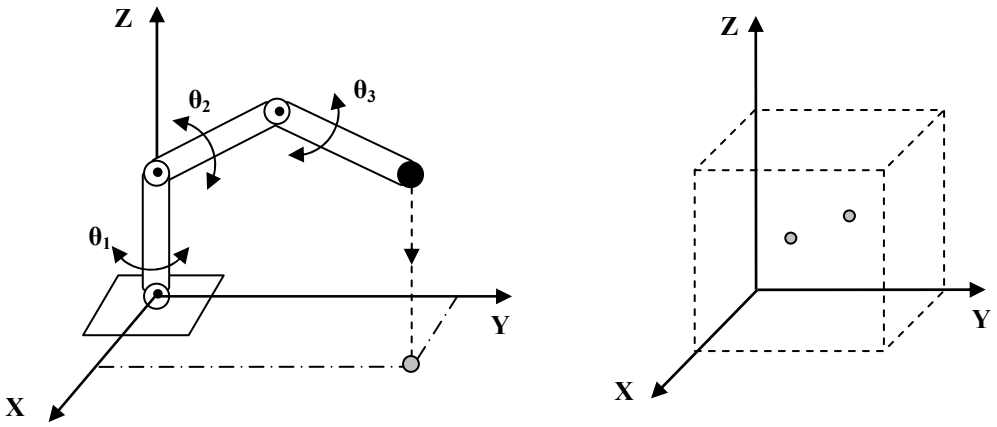


Figure 4.21 - A three-link manipulator and its workspace.

The world coordinates (*x, y, z*) represent a position of the manipulator end-effector. In equations (4.12), the joint angle θ_3 is not a unique value because it is expressed by two different functions. This reflects the multi-solution characteristics of the inverse kinematics problem. In this simulation, the plus sign in equation (4.12) corresponding to the upper-elbow structure was used. Hence, the third joint angle θ_3 was always a negative value. For this simulation, the three-dimensional workspace was limited to a specific region where the three space coordinates were all positive values {(x > 0), (y > 0), (z > 0)}.

4.4.1 Simulation description

To approximate the inverse kinematics function of the three-link manipulator, an RBFN as shown in Figure 4.22 was applied. Simulations were performed following this procedure:

- The structure of the hidden layer was built with pre-defined centres regularly distributed in the workspace (10 mm x 10 mm x 10 mm cubes). The total number of hidden units was 410 nodes. The spread was experimentally selected so that the RBFN can produce an appropriate inverse kinematics approximation.

- Training patterns $\{(x, y, z); (\theta_1, \theta_2, \theta_3)\}$ were collected as either constrained or random data in the workspace. There were three sets of training data: constrained and random with a maximum deviation of 2 mm and 3 mm (Figure 4.23).

- The linear weights were adjusted by one of two methods, strict interpolation or LMS.

- Two test trajectory sets (Figure 4.24), which did not occur in the training phase, were presented to verify the performance of the RBFN after training. The first one was a trajectory inside the workspace (5 mm x 5 mm x 5 mm cubes) and the second was a trajectory near the edge of the workspace.

`

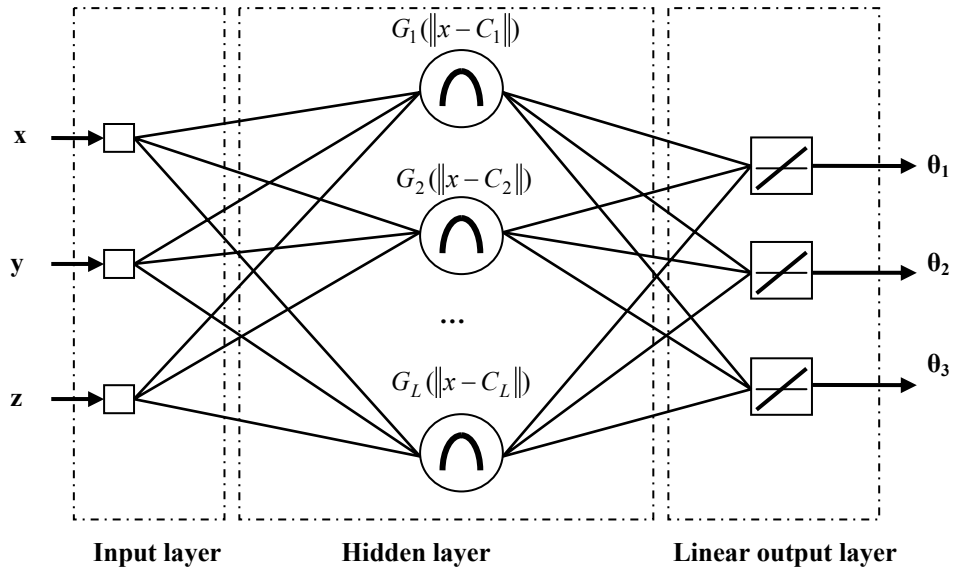


Figure 4.22 - Configuration of the RBFN to approximate the inverse kinematics of the three-link manipulator.

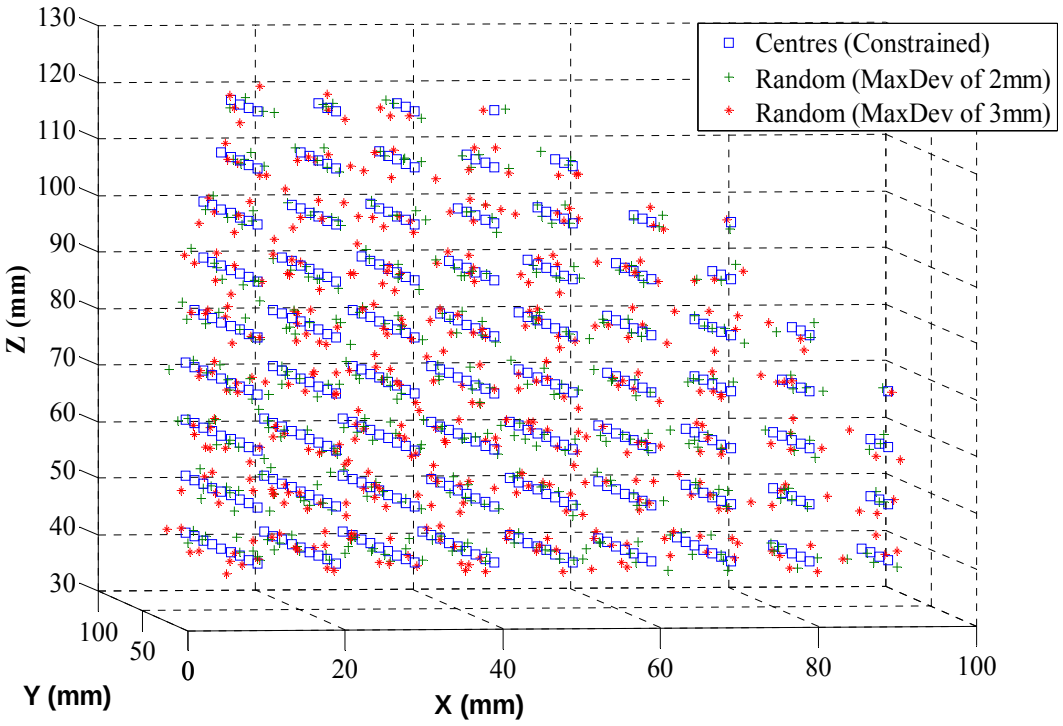


Figure 4.23 – Distribution of three training data sets used in the three-link manipulator simulations.

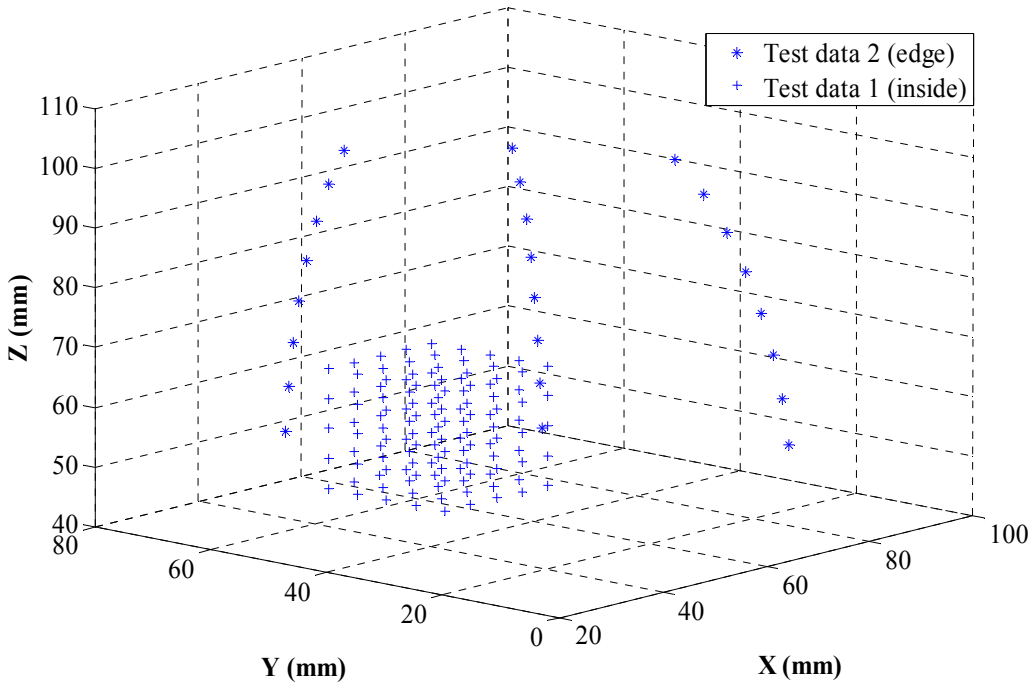


Figure 4.24 – Test trajectories for the three-link manipulator simulations.

Similar to the two-link manipulator case, the three-link manipulator simulations investigated the network performance for various conditions: two different training methods were used with three training data sets and a variety of spread values (6 - 28 mm) were chosen. The RMS error between the joint angles produced by the network and the desired inverse kinematics function (mathematical expressions) and MAEs (mean absolute errors) between the desired and actual positions in *X*, *Y*, *Z* directions were used to present the network performance. The results of the three training cases with different training data are plotted in the same figure to compare the effect of training data on the network's performance.

4.4.2 Simulation results

In the first case, the RBFN was trained by the strict interpolation method with the three different training data sets. Figures 4.25 and 4.26 present the network performance for test trajectory 1 (inside). Figures 4.27 and 4.28 present the performance of the same network for test trajectory 2 (edge). These simulation results are detailed in Appendix B.4.

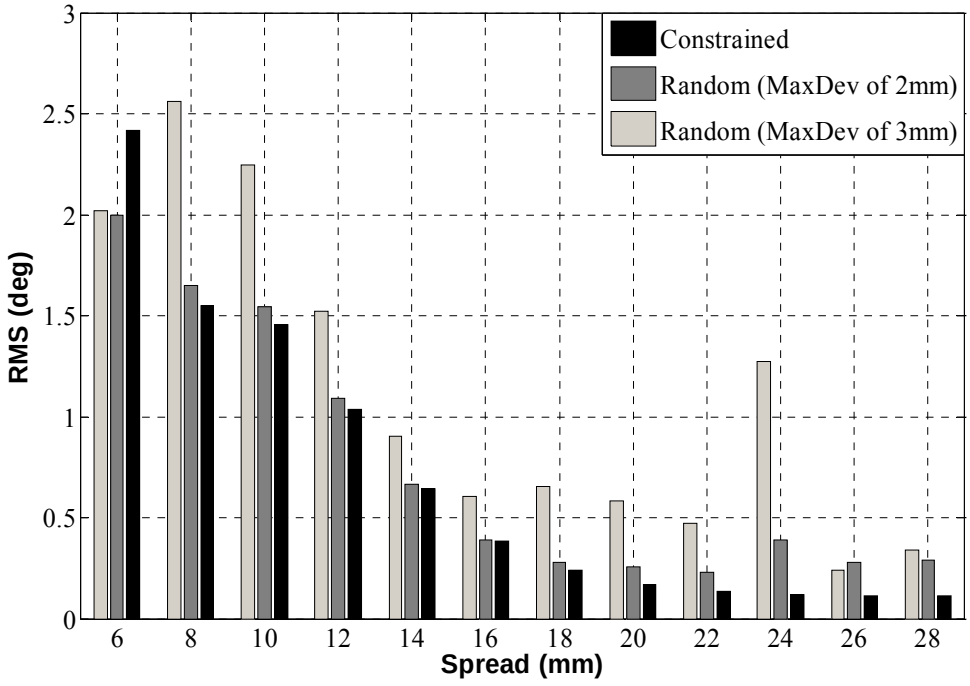


Figure 4.25 - Performance results for test trajectory 1 (inside).

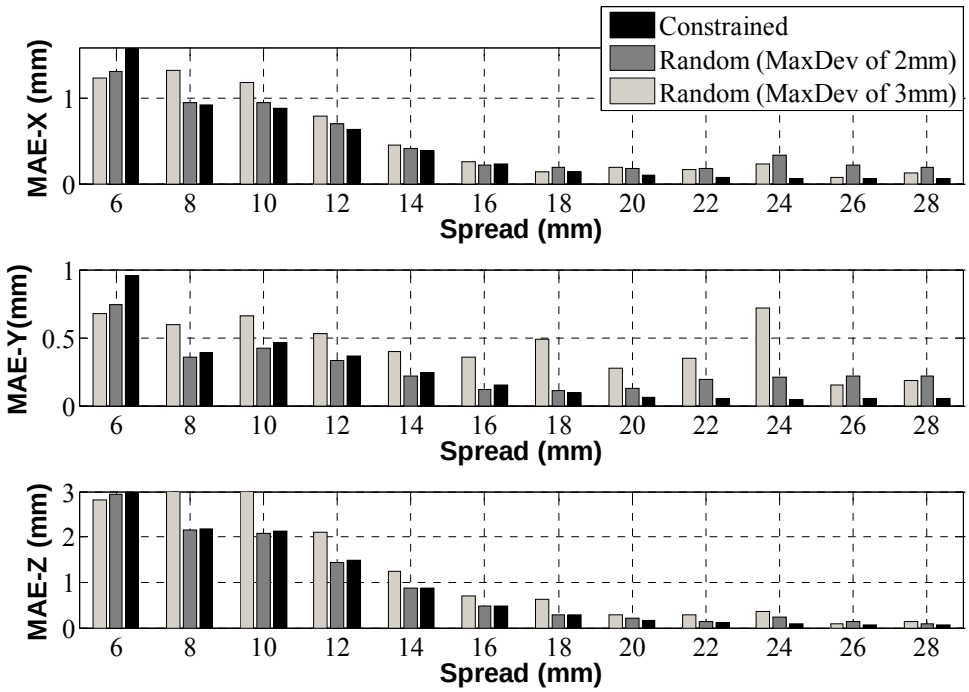


Figure 4.26 - MAEs for test trajectory 1 (inside)

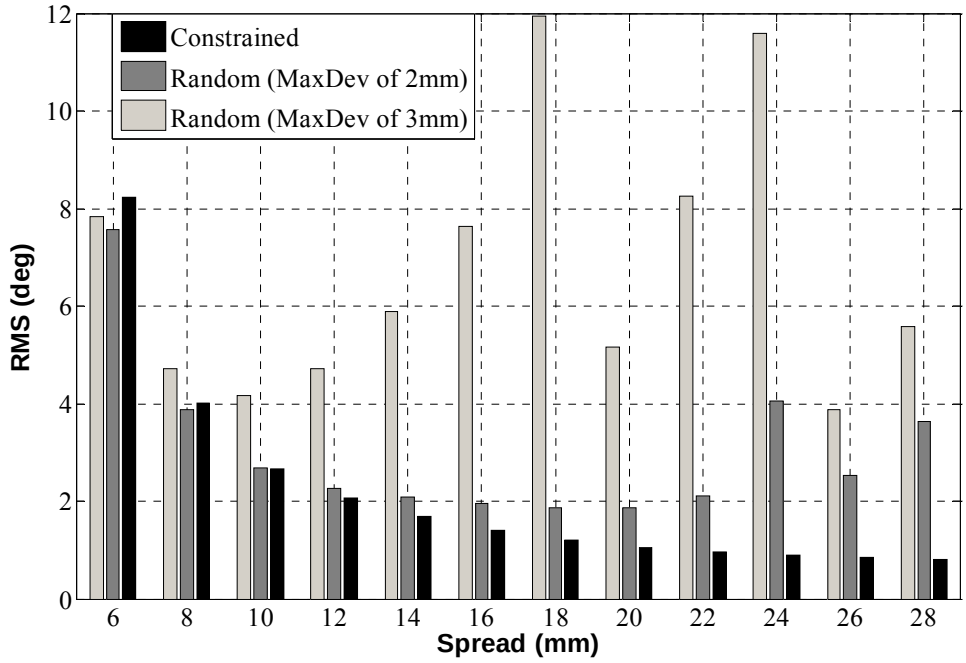


Figure 4.27 - Performance results for test trajectory 2 (edge).

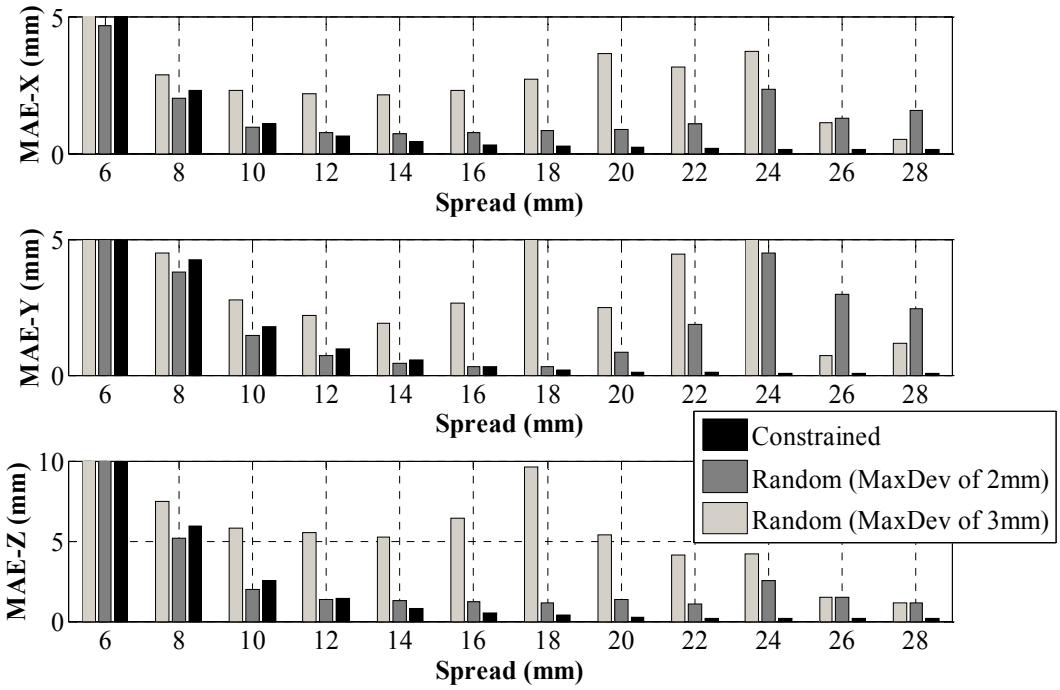


Figure 4.28 - MAEs for test trajectory 2 (edge).

The results show that the RBFN trained by the strict interpolation method produces an appropriate approximation of the inverse kinematics function for the three-link manipulator. For the test trajectory inside the workspace (Figure 4.26), using the constrained data produces the best performance where the average MAE (of X, Y and Z directions) is approximately 0.1 mm for spreads between 20 to 28 mm. The performance is also good when using the random data with a maximum deviation of 2 mm (average MAE of approximately 0.2 mm for spreads between 20 to 28 mm) and is poorer when using the random data with a maximum deviation of 3 mm. For test trajectory 2 (Figure 4.28), the performance is poorer with the same network compared to test trajectory 1. This is similar to the two-link manipulator simulation. The effect of the different training data sets (constrained or random) on the network's performance is not significant for test trajectory 1. However, for test trajectory 2, the network using the random data with a maximum deviation of 3 mm produced a significantly poorer performance compared to the other training data sets. The RBFN used in the three-link manipulator simulations requires a wider spread to achieve the optimal inverse kinematics approximation (20 to 28 mm) compared to the two-link simulations (16 to 22 mm).

The second simulation was performed with the same network and training data but used the LMS algorithm. The training process was implemented with the following parameters:

- Maximum training epochs = 500000,
- Goal = 0.0001,
- Learning rate = 0.01 – 0.001.

Similar to the two-link manipulator simulation, the training process using the LMS algorithm was significantly slower than the strict interpolation method. It is slower than the two-link case due to the greater number of hidden nodes and training points (410 nodes compared to 110 nodes) and a more complex network structure (3 inputs – 3 outputs). Figures 4.29 and 4.30 present the network's performance for test trajectory 1 after training by the LMS algorithm corresponding to various spread values (Appendix B.5). Figures 4.31 and 4.32 present the performance of the same network for test trajectory 2.

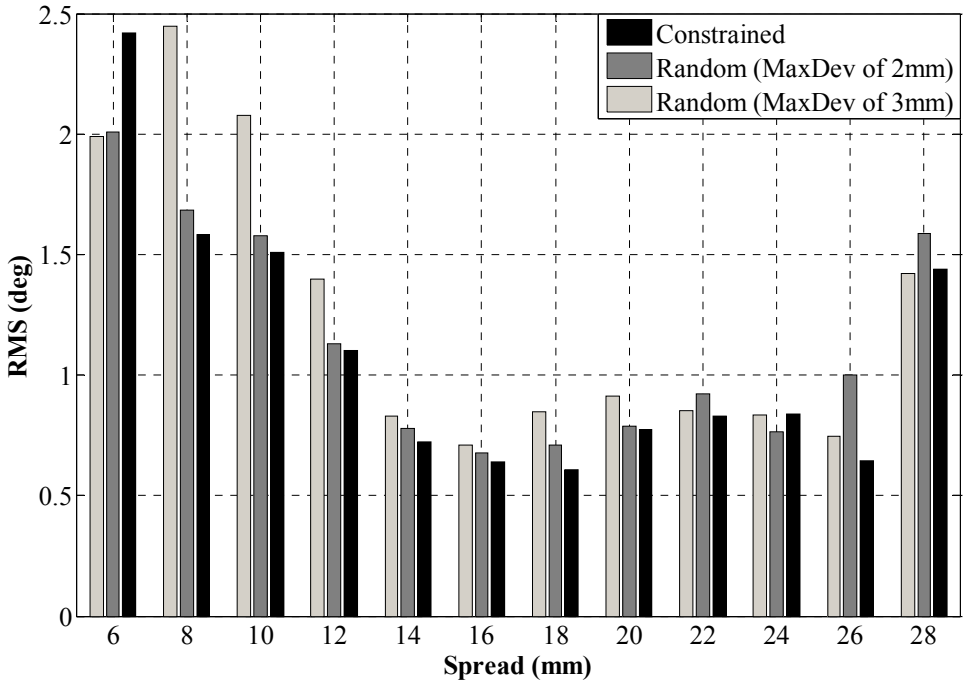


Figure 4.29 - Performance results for test trajectory 1 (inside).

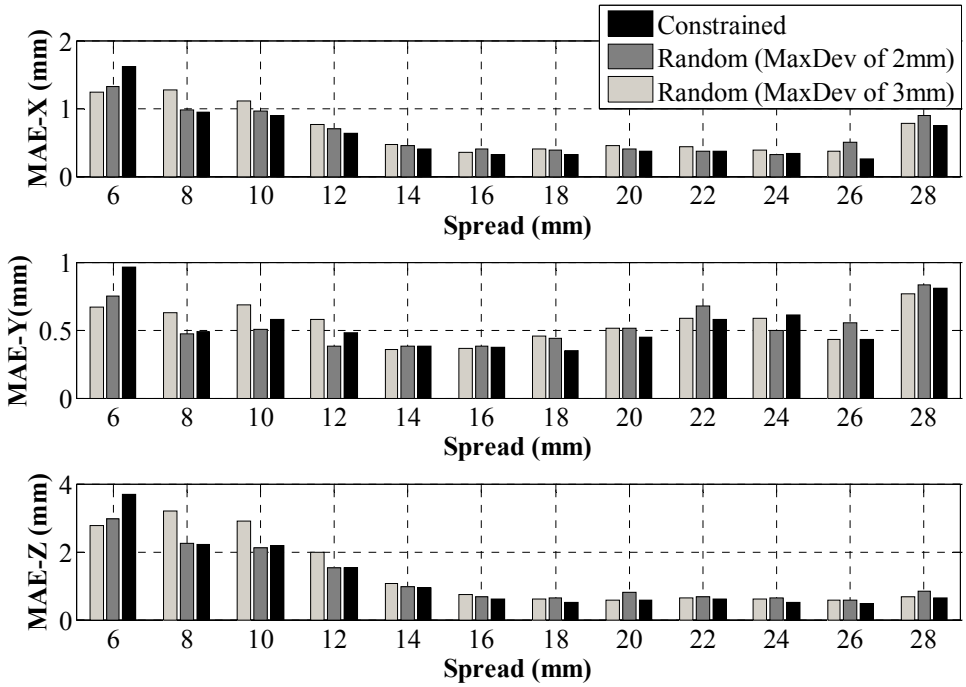


Figure 4.30 - MAEs for test trajectory 1 (inside).

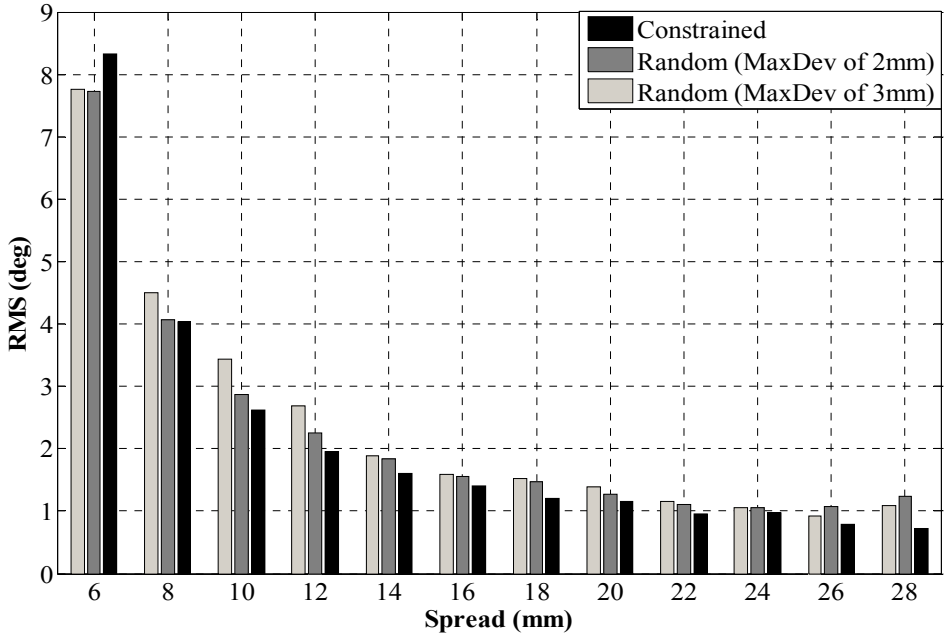


Figure 4.31 - Performance results for test trajectory 2 (edge).

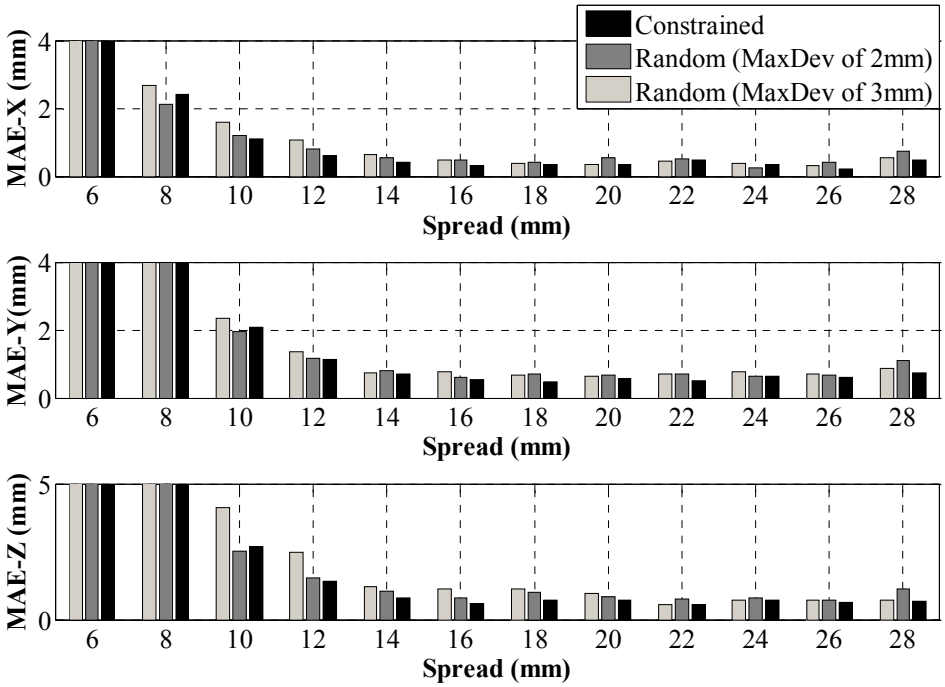


Figure 4.32 - MAEs for test trajectory 2 (edge).

The results show that the network trained by the LMS algorithm can produce a good inverse kinematics approximation for the three-link manipulator. For test trajectory 1 (Figure 4.30), using the constrained data an average MAE of approximately 0.5 mm can be obtained for spreads between 14 and 18 mm. The network's performance is not significantly different when using different training data (constrained or random). The performance of the same network is poorer for test trajectory 2. In general, the performance of the network trained by the LMS algorithm is poorer compared to the network trained by the strict interpolation method. For both test trajectories, using the LMS algorithm requires a smaller spread value to obtain the best performance compared to the strict interpolation method.

4.4.3 Summary of results

Similar to the two-link manipulator simulations, using an RBFN with regularly-spaced position centres has produced an excellent approximation of the inverse kinematics function for a three-link manipulator. The RBFN trained by the strict interpolation method, using constrained data produced an average MAE of approximately 0.1 mm (or 1% of the centre distance) for the test trajectory inside the workspace. However, it

requires a higher spread value compared to the two-link manipulator case to obtain the best performance when training by either the strict interpolation or LMS methods.

The training process using the LMS method is extremely slow with a large spread value. This situation is more serious compared to the two-link manipulator case due to the more complex structure of the three-link manipulator and the larger number of training points required.

The set of random data where the maximum deviation is not higher than 20% of the centre distance also produces good results.

4.5 Conclusion

The proposed approach using an RBFN to approximate the inverse kinematics function of robot manipulators has been presented in this chapter. Various simulations for two-link and three-link manipulators have been presented to demonstrate the effectiveness of the RBFN. Some conclusions can be stated:

- The selection of hidden unit centres as regularly-spaced positions in the workspace significantly improves the network performance.

- The training process using the strict interpolation method with training data collected closely to the centre positions enhances the network performance.

- The generalisation capability of an RBFN is closely related to the structure of the hidden layer (centre distance and spread). If the centre distance is fixed due to the limited number of hidden units, the spread value chosen affects the network's performance significantly, especially when using the strict interpolation method.

- The effect of using random training data (high deviation from the centre points) on the network's performance is not significant when the network is trained by the LMS algorithm. Thus, it would be a suitable alternative if the strict interpolation method was not chosen due to the significant number of hidden units required and the training data has a high deviation from the centre points.

4.6 References

[4.1] J. A. Driscoll, "Comparison of Neural Network Architectures for the Modelling of Robot Inverse Kinematics", in *Proc. of the 2000 IEEE SOUTHEASTCON,* Tennessee, USA, vol. 3*,* April 2000, pp. 44-51.

[4.2] S. S. Yang, M. Moghavvemi, and John D. Tolman, "Modelling of Robot Inverse Kinematics Using Two ANN Paradigms", in *Proc. of TENCON2000-Intelligent System and Technologies for the New Millennium*, Kuala Lumpur, Malaysia, Sept. 2000, vol. 3, pp. 173-177.

[4.3] P. Y. Zhang, T. S. Lu, L. B. Song, "RBF Networks-Based Inverse Kinematics of 6R Manipulator", *International Journal of Advanced Manufacturing Technology*, Springer – Verlag London Ltd., vol. 26, 2004, pp.144-147.

[4.4] M. J. L. Orr, *Introduction to Radial Basis Function Networks*. [Online], Available: http://www.anc.ed.ac.uk/rbf/rbf.html, 1996.

[4.5] G. W. Irwin, K. Warwick and K. J. Hunt, *Neural Network Applications in Control*, IEE Control Engineering Series 53, 1995.

[4.6] S. Haykin, *Neural Networks a Comprehensive Foundation – Second Edition*. Prentice Hall, 1999.

[4.7] A. Guez and Z. Ahmad, "Solution to The Inverse Kinematics Problem in Robotics by Neural Networks", in *Proc. of 1988 IEEE Int. Conf. on Neural Networks*, San Diego, USA, vol.1, July 1988, pp. 617-624.

[4.8] B. B. Choi and C. Lawrence, "Inverse Kinematics Problem in Robotics Using Neural Networks", *NASA Technical Memorandum - 105869*, October 1992.

[4.9] Z. Binggul, H. M. Ertunc, and C. Oysu, "Comparison of Inverse Kinematics Solutions Using Neural Network for 6R Robot Manipulator with Offset", in *Proc. of the 2005 Congress on Computational Intelligence Method & Application*, Istanbul, Turkey, Dec. 2005, pp. 1-5.

[4.10] E. Watanabe and H. Shimizu, "A Study on Generalization Ability of Neural Network for Manipulator Inverse Kinematics", in *Proc. of the 17th Int. Conf. on*

Industrial Electronics, Control and Instrumentation, Kobe, Japan, vol. 2, Nov. 1991, pp. 957-962.

CHAPTER 5
ONLINE TRAINING TO MODIFY THE INVERSE KINEMATICS APPROXIMATION

5.1 Introduction

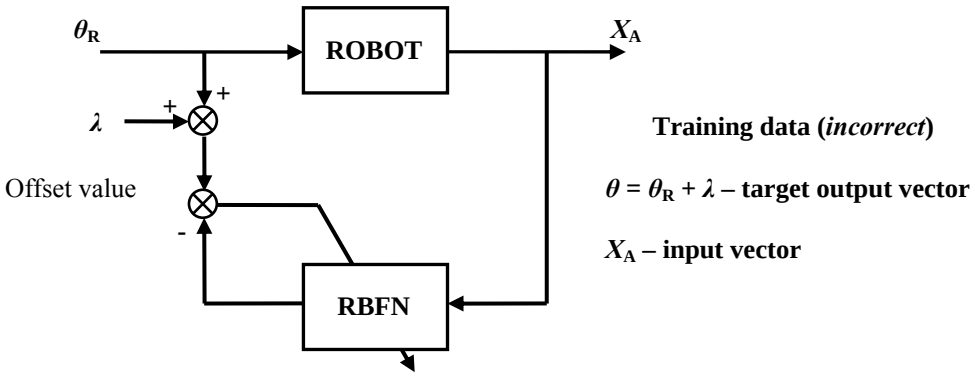
In Chapter 4, the idea of using a radial basis function network (RBFN) with the centres of hidden-layer units distributed regularly in the workspace to approximate the inverse kinematics problem was presented. Simulation results have shown that this approach can produce an appropriate approximation for the inverse kinematics transformation of a robotic manipulator. However, sometimes a well-trained network cannot work effectively in the operational phase because the initial network training occurs in an environment that is not exactly the same as the environment where the system is actually deployed. An online retraining approach can be effectively applied for systems whose characteristics change due to environmental variations. An example of this is a robot-vision system whose structure changes due to environment alterations between the initial training phase and practical deployment, e.g., different type of camera or variation in distance and view angle between the camera and robot.

This chapter presents an approach to modify the RBFN using an additional online retraining phase. The RBFN, which has been trained to approximate the inverse kinematics of a manipulator, can be modified through an online process during the operational phase. This is an additional phase using the delta rule to update the linear weights of the RBFN which has been initially trained. Simulations for two-link and three-link manipulators are then presented and discussed.

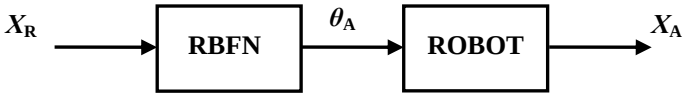
5.2 Using online training to modify the inverse kinematics approximation

The principle of this approach is that an RBFN which has been trained before is modified by the delta rule (also known as the Widrow–Hoff method [5.1]) through an additional online process during the operational phase. This chapter presents simulation work that consists of two stages: an incorrect inverse kinematics approximation is produced by the strict interpolation method and then is corrected through an online

retraining phase. The incorrect network reflects the reasonable assumption that the initial network training occurs in an environment that is not exactly the same as the environment where the system is actually deployed. Figure 5.1 presents an example where the RBFN is trained with data that does not reflect the correct characteristics of a robotic system in which the RBFN will actually operate. This situation is due to an offset value added to all target outputs of the training data.



(a) – Training phase with incorrect data.



(b) – Operational phase with incorrect IK approximation.

Figure 5.1- Block diagram of training and operational phases.

For the simulations in this chapter, data used in the initial training phase is created by the following procedure. A set of N training patterns is generated using mathematical expressions for the inverse kinematics functions

$$\mathrm{S}_{\mathbf{1}} = \{(X_i, \theta_i)\} \tag{5.1}$$

where X_i and θ_i are the i^{th} input and target output of training set (the position and the corresponding joint angle of the robot).

A constant error is added as an offset value to all target outputs. This produces a set of incorrect patterns called $\boldsymbol{S_2}$

$$\mathrm{S_2} = \left\{ (X_i, \hat{\theta}_i) \right\} \tag{5.2}$$

where

$$\hat{\theta}_i = \theta_i + offset\,. \tag{5.3}$$

Therefore, using the data set $\boldsymbol{S_2}$ in the training phase means that the RBFN produces an incorrect approximation of the inverse kinematics function.

The online retraining process is implemented using the Delta rule with recent data collected during the operational phase. The training criterion is the cost function expressed as a sum of square errors at each training pattern [5.1]. It is given by

$$E(X_k) = \frac{1}{2}\sum_{j=1}^{L} e_j^2(X_k) \tag{5.4}$$

where

$$e_j(X_k) = \theta_j - \sum_{i=1}^{M} W_{ji}\Phi_i(X_k)\,. \tag{5.5}$$

θ_j is the target output of a network output j, Φ_i is the output of the i^{th} hidden Gaussian function and W_{ji} is the interconnection weight between the network output j and the i^{th} hidden unit. The weight adjustment is derived using the Delta rule as

$$\Delta W_{ji} = \eta.e_j(X_k).\Phi_i(X_k) \tag{5.6}$$

where η is learning rate ($0 \le \eta \le 1$).

One drawback of an online training process is learning interference where the training effect of a current training point may upset some of the weights which were trained with other points, if they are close together [5.2]. Consequently, the RBFN at a retraining step can converge to the desired function in one area but diverge in other areas. This learning interference is more serious when the spread and/or the learning rate is large. Based on the characteristics of Gaussian functions, a simple rule is proposed to select appropriate patterns in order to avoid learning interference. Given an input x, the output of a Gaussian function can be calculated as

$$\Phi = \exp\left(-\left(\frac{0.832.D}{Sp}\right)^2\right) \tag{5.7}$$

where Sp is the spread of the Gaussian function and $D = \|x - C\|$ is the distance between training points and the centre of the Gaussian function.

From equation (5.7), the relationship between the ratio of the distance D to the spread value and the output of the Gaussian function can be determined as

$$\frac{D}{Sp} = \sqrt{\frac{\ln \Phi}{-0.693}}, \qquad 0 \le \Phi \le 1. \tag{5.8}$$

Thus, the effect of a training point on the output of a hidden unit is dependent on the distance between this training point and the centre of the hidden unit in proportion to the spread. It can be formulated as follows:

$$\Phi \ge 0.5 \Leftrightarrow D \le Sp \tag{5.9}$$

$$0.1 \le \Phi \le 0.5 \Leftrightarrow Sp \le D \le 1.822Sp \tag{5.10}$$

$$0.05 \le \Phi \le 0.1 \Leftrightarrow 1.822Sp \le D \le 2.079Sp \tag{5.11}$$

$$\Phi \leq 0.05 \Leftrightarrow D \geq 2.079Sp\,. \qquad (5.12)$$

The principle of the free interference rule is that if the position of a new training point is further than twice the spread value from the position of the previous training point used to train the network, then the training process (with this new point) will not interfere in the weights of hidden units whose centres are close to the previous point. Therefore, following the free interference rule, the RBFN may be modified gradually and smoothly in this online retraining process.

5.3 Simulation procedure

A MATLAB simulation was developed to demonstrate the proposed approach. This simulation used two manipulator structures: two-link and three-link, the same as the simulations in Chapter 4. It answered two questions: how an incorrect approximation can be improved by an online training process and which factors affect this retraining process. The simulation procedure can be described as below:

Step 1: Select the hidden layer parameters as presented in Chapter 4 where:

i/ centres of the hidden layer are predefined as regularly spaced positions in the workspace,

ii/ the spread of the Gaussian functions is heuristically selected as a proportion of the centre distance.

Step 2: Generate a constrained training set where its inputs are coincident with the centre positions in the workspace using the mathematical inverse kinematics functions presented in sections 4.3 and 4.4. A constant error (offset value) is then added to the target outputs. Thus, the new training set is incorrect because the inputs and target outputs no longer correspond to each other.

Step 3: Train the RBFN with the incorrect training set using the strict interpolation method. This produces an incorrect inverse kinematics approximation.

Step 4: Adopt a learning rate (0.1 - 0.5) and update the linear weights of the RBFN through the online training process. It is described by the following flowchart.

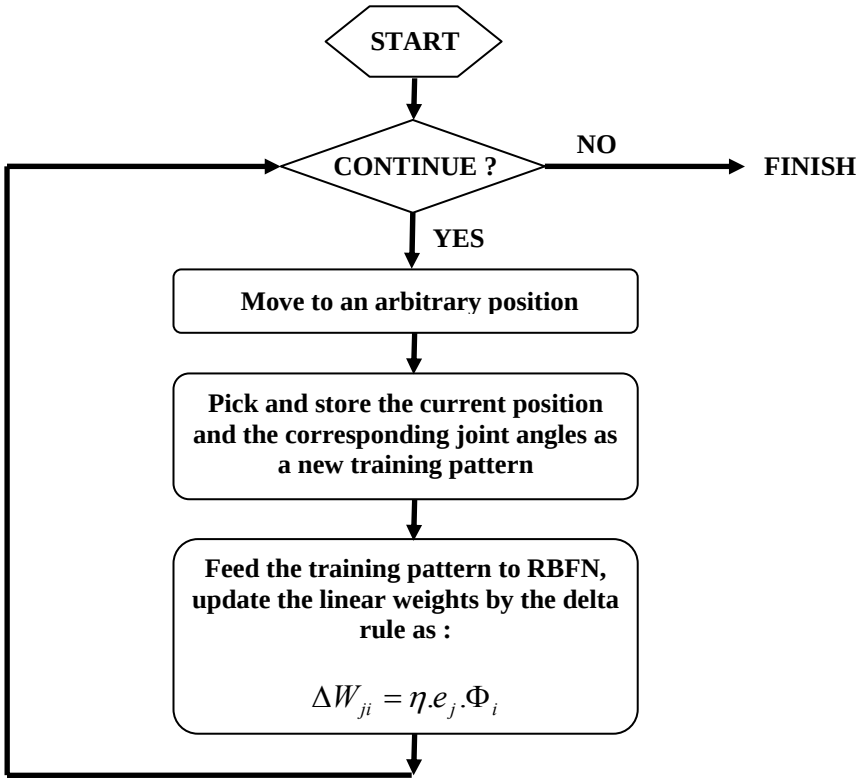


Figure 5.2 – Online retraining flowchart.

The effect of this retraining phase is dependent on three factors: learning rate, spread and the distance between centres and training points.

To examine the network performance a range of test data are presented after each retraining step. The performance criterion is *root mean square* (RMS) error between the approximation function (modified RBFN) and the desired function (mathematical expressions). In addition, *mean absolute errors* (MAEs) in X, Y, and Z (for the three-link manipulator case) directions between the actual positions and the desired positions are also used to demonstrate the network performance.

5.4 Two-link manipulator simulation

In this simulation, a set of regularly-spaced position centres were selected as 10 mm x 10 mm grids in the workspace. A training set was created where the inputs were coincident with pre-defined centre positions and an offset value of 10 degrees was added to the target outputs. As a result, the RBFN which was trained by this data set using the strict interpolation method produced an incorrect inverse kinematics approximation. To examine the network's performance, a range of test data distributed as 5 mm x 5 mm grids in the square area {x = 22 – 52 mm; y = 22 - 52 mm} were sent to the RBFN. Figure 5.3 presents the outputs (joint angles) of the desired inverse kinematics function and the incorrect approximation (the RBFN) in this test area. It shows that the surface of the approximation functions ($\theta_1 = f_1(x, y)$ and $\theta_2 = f_2(x, y)$) are parallel to the surface of the desired functions but differ by a 10 degree offset. The network performance is obtained as: RMS error = 9.91 degrees, MAE_X = 14.39 mm and MAE_X = 3.21 mm.

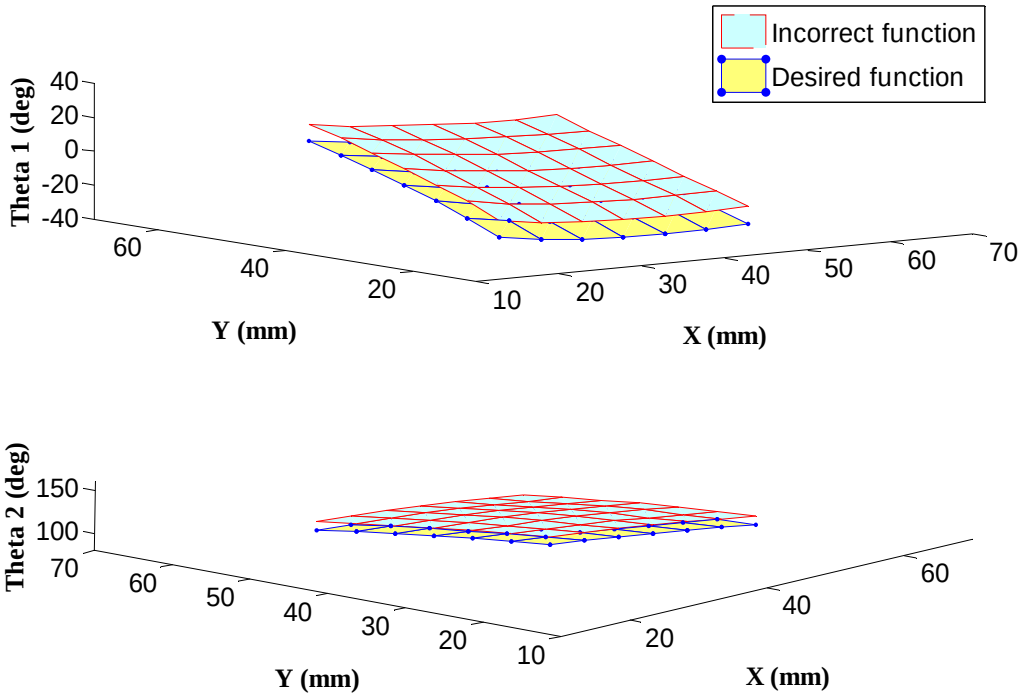


Figure 5.3 - Surfaces of desired (mathematical expression) and approximation functions (RBFN) in the test area.

To clarify the effect of online retraining in modifying the inverse kinematics approximation, a series of the network's performances through 5 training points, collected using the free interference rule, are shown in Figures 5.4 to 5.8. The retraining process was performed with a spread of 10 mm and a learning rate of 0.4. To illustrate how the inverse kinematics approximation is modified through retraining, three functions are shown on the same figure: the incorrect function (before retraining phase), modified function and the desired function.

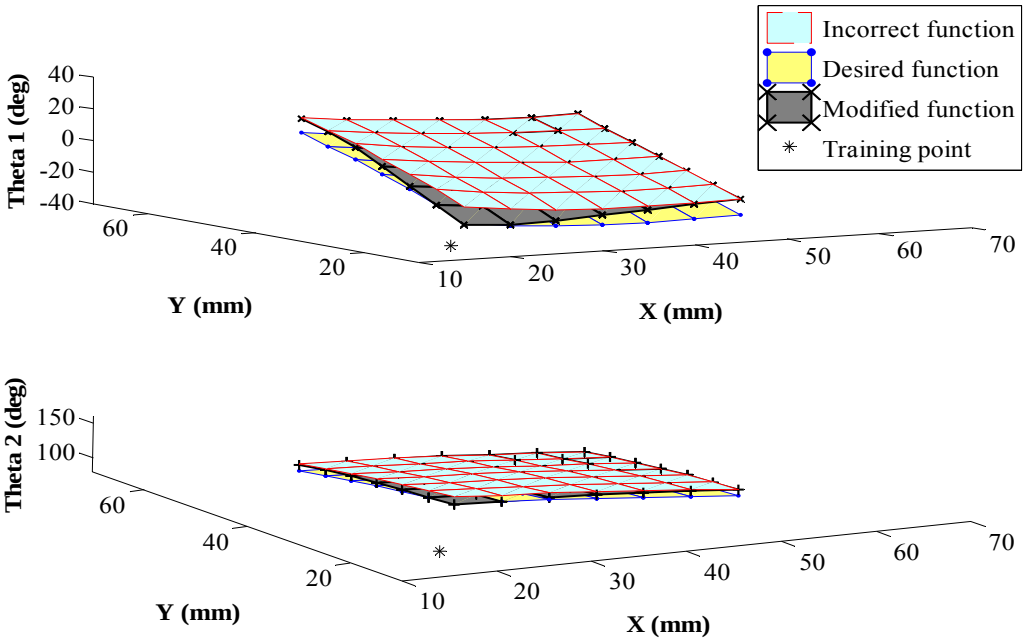


Figure 5.4 - The network's performance after retraining by the first point (20 mm; 21mm).

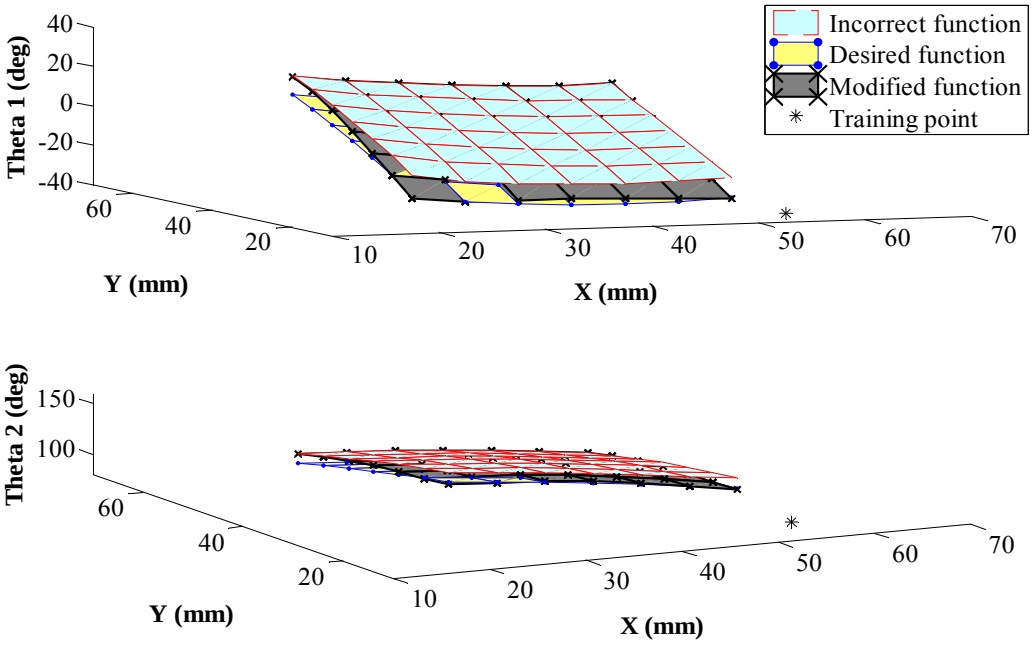


Figure 5.5 - The network's performance after retraining by the second point (56 mm; 19 mm).

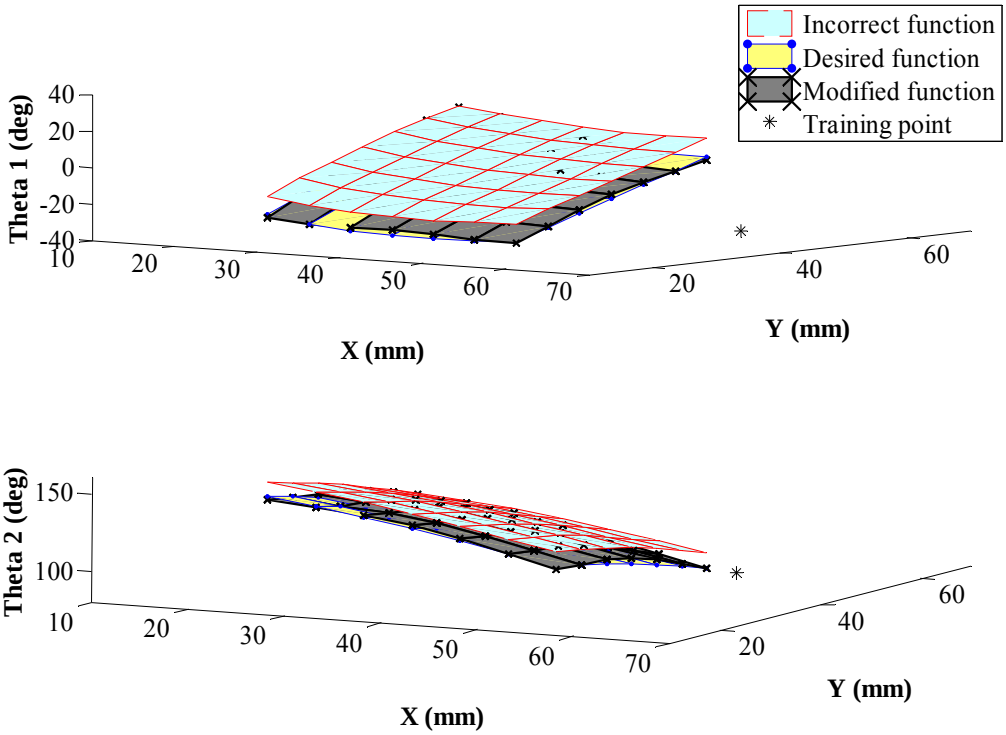


Figure 5.6 - The network's performance after retraining by the third point (63 mm; 56 mm).

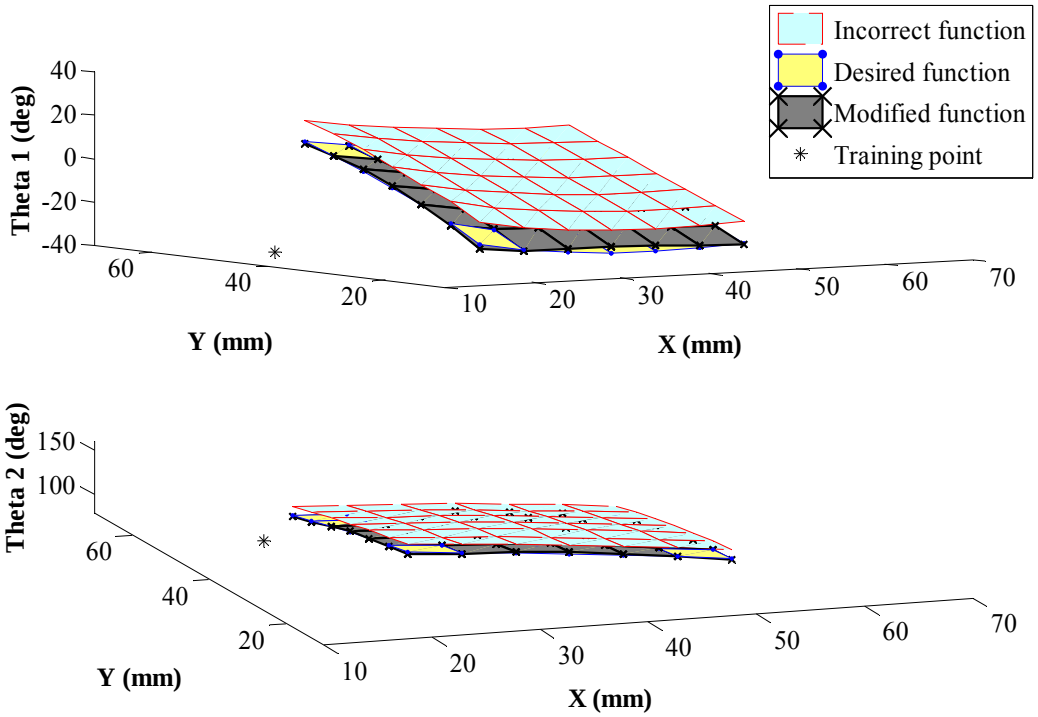


Figure 5.7 - The network's performance after retraining by the fourth point (20 mm; 54 mm).

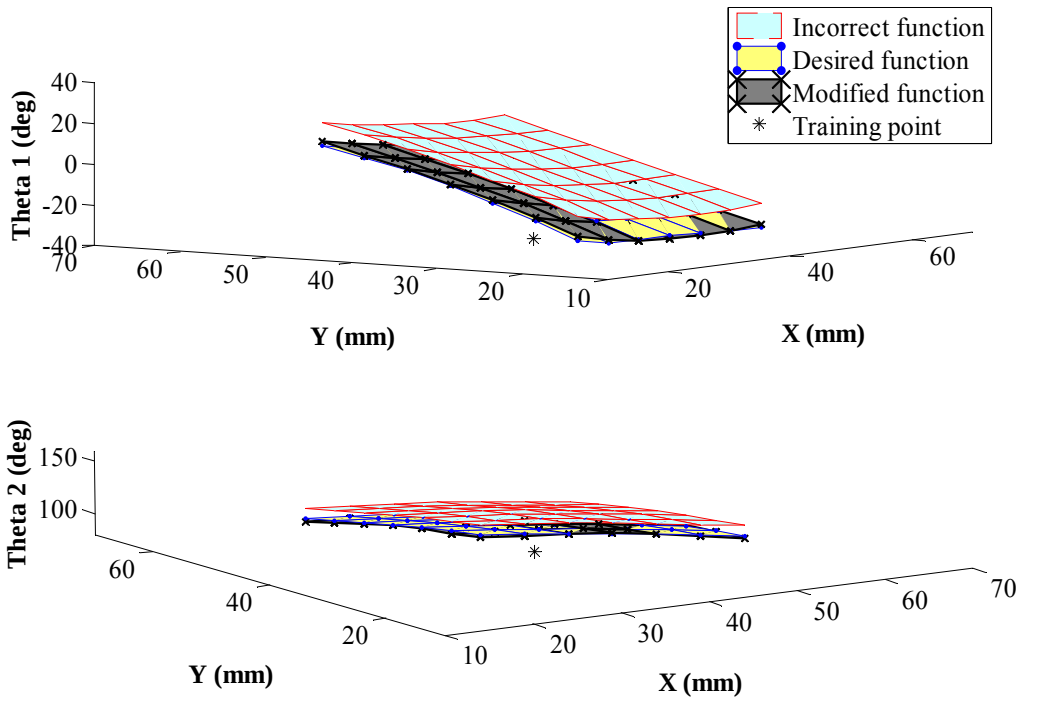


Figure 5.8 - The network's performance after retraining by the fifth point (40 mm; 40 mm).

The results show that the online retraining has modified the inverse kinematics approximation gradually. After 5 online retraining steps, the RBFN is close to the desired function. By selecting the training patterns following the free interference rule, the current modified function at each step approaches the desired function in the area surrounding the training point whilst not affecting other areas.

The effect of spread and learning rate on the retraining process is shown in Table 5.1 which presents the RMS errors (between the current modified function and desired function). A small spread (e.g., 8 mm) requires a high learning rate to produce a small RMS error. A large spread (e.g., 16 mm) requires a small learning rate to produce a small RMS error. For other spread values, the minimum RMS error is obtained for learning rate values between 0.1 and 0.4. For example, a learning rate of 0.4 produces the minimum RMS error with a spread of 10 mm. The selection of the spread and learning rate should be carefully considered to ensure a successful retraining process.

Spread / **L.r**	**8(mm)**	**10(mm)**	**12(mm)**	**14(mm)**	**16(mm)**
0.1	8.6	7.1	5.06	2.83	1
0.2	7.34	4.66	1.69	0.78	1.65
0.3	6.13	2.61	0.89	2.02	5.1
0.4	5	1.19	2.06	4.52	11.03
0.5	3.94	1.43	3.02	9.14	17

Table 5.1 - RMS errors (degrees) after online retraining with 5 training points.

The improvement of the RBFN is dependent on the learning rate, the spread and the position of training points in the workspace. A large learning rate and/or a large spread can improve the RBFN approximation in the area around the current training point whilst other areas become poorer due to learning inference. In this research, it is preferred to adopt a small learning rate and an average spread value for the retraining process.

The retraining process can be continued with new training points. Figure 5.9 presents the performance results of the RBFN after retraining with 5 more points (10 training points in total). These were: {(20 mm; 21 mm), (56 mm; 19 mm), (53 mm; 56 mm), (20

mm; 54 mm), (40 mm; 40 mm), (24 mm; 25 mm), (50 mm; 24 mm), (51 mm; 49 mm), (25 mm; 50 mm), (38 mm; 37 mm)}. A better performance is obtained:

- RMS error = 0.73 degrees,
- MAE_X = 0.74 mm,
- MEA_Y = 0.33 mm.

However, the improvement of the RBFN varies depending on the training points chosen and their presentation order. To illustrate this point, 15 training points were randomly collected in the test area: {(34 mm; 40 mm), (39 mm; 36 mm), (30 mm; 28 mm), (50 mm; 43 mm), (51 mm; 38 mm), (47 mm; 33 mm), (42 mm; 26 mm), (46 mm; 23 mm), (32 mm; 22 mm), (25 mm; 26 mm), (21 mm; 30 mm), (32 mm; 28 mm), (42 mm; 32 mm), (49 mm; 28 mm), (39 mm; 52 mm)}. The network's performance after online retraining (learning rate = 0.4, spread = 10 mm) with these training points is:

- RMS error = 2.52 degrees,
- MAE_X = 3.04 mm,
- MAE_Y = 0.96 mm.

This result is poorer than the network's performance after retraining with 10 training points (following the free interference rule) as presented previously.

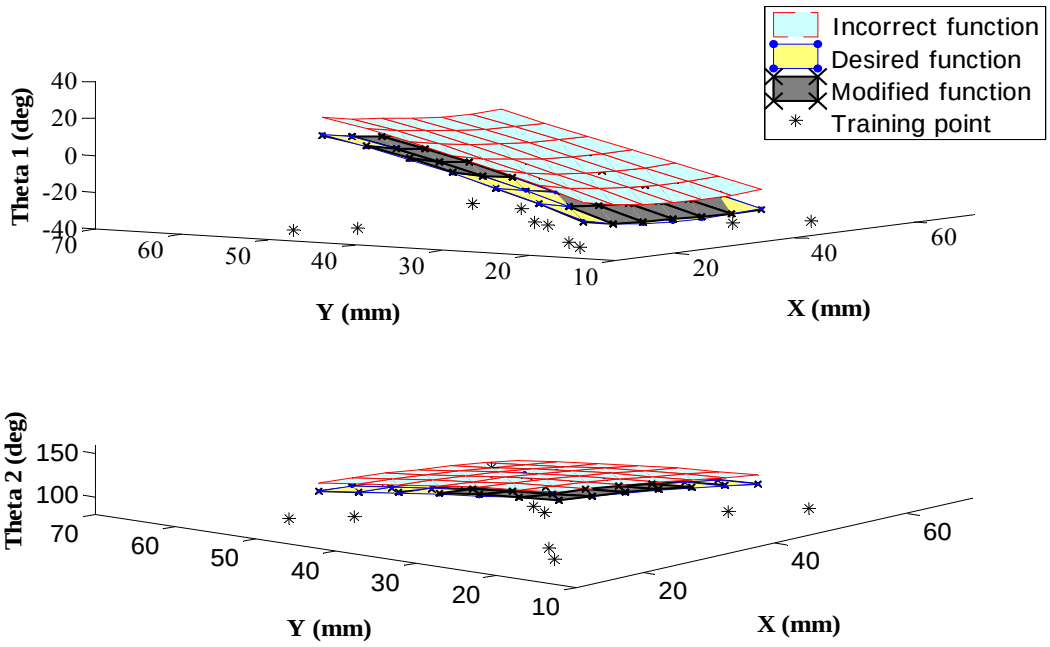


Figure 5.9 - The network's performance after retraining with a total of 10 training points following the free inference rule.

The retraining process tends to continuously improve the inverse kinematics approximation with further training points. If the number of training points is sufficient and other factors (learning rate and spread) are appropriately chosen, the RBFN can produce an approximation function similar to that obtained by the strict interpolation method with correct training data (Chapter 4).

5.5 Three-link manipulator simulation

In this simulation, a set of regularly-spaced position centres was selected as 10 mm x 10 mm x 10 mm cubes in the workspace. A training set was created where the inputs were coincident with pre-defined centre positions and an offset value of 5 degrees was added to the target outputs. As a result, the RBFN trained with this training set using the strict interpolation method produced an incorrect inverse kinematics approximation. In order to examine the network's performance, a range of test data distributed as 5 mm x 5 mm x 5 mm cubes was sent to the RBFN as shown Figure 5.10. It is not convenient to show the inverse kinematics function surfaces of the three-link manipulator in the same manner as the two-link manipulator. Thus, the network's performance can be presented as errors between the desired and the current modified function for the test points. These

errors are the differences between joint angles produced by the RBFN and the mathematical inverse kinematics expressions.

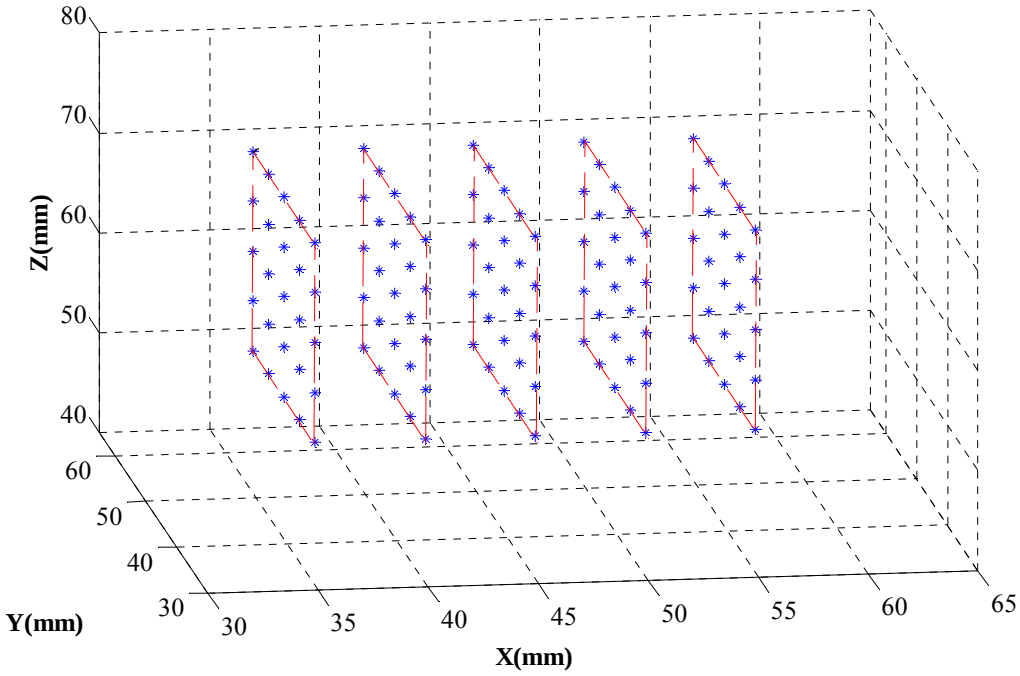


Figure 5.10- Test data distributed as 5 mm x 5 mm x 5 mm cubes in the workspace.

Figure 5.11 shows the network's performance after initial training with the incorrect data (offset value of 5 degrees added to all target outputs). The network's performance is:

- RMS error = 5.12 degrees,
- MAE_X = 5.87 mm,
- MAE_Y = 2.02 mm,
- MAE_Z = 10.52 mm.

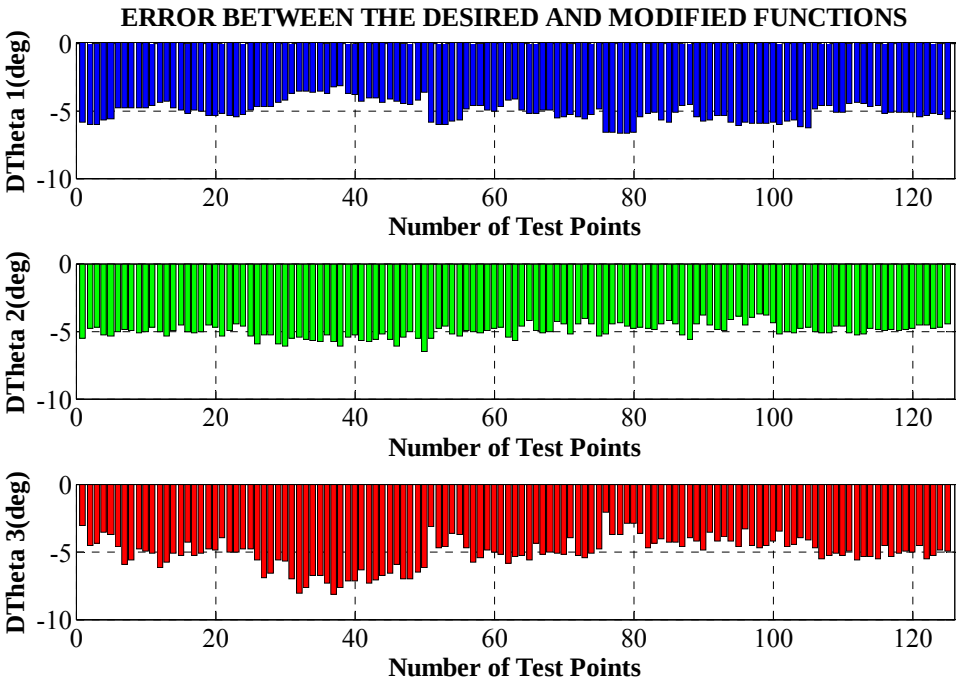


Figure 5.11- The network's performance after initial training with incorrect data.

To improve the network's performance a series of 15 training points (Figure 5.12), following the free interference rule, were applied to the online retraining process. These were: {(34 mm; 35 mm; 48 mm), (34 mm; 61 mm; 48 mm), (34 mm; 61 mm; 74 mm), (34 mm; 35 mm; 74 mm), (34 mm; 48 mm; 62 mm), (58 mm; 35 mm; 48 mm), (58 mm; 61 mm; 48 mm), (58 mm; 61 mm; 74 mm), (58 mm; 35 mm; 74 mm), (58 mm; 48 mm; 62mm), (46 mm; 35 mm; 48 mm), (46 mm; 61 mm; 48 mm), (46 mm; 61 mm; 74 mm), (46 mm; 35 mm; 74 mm), (46 mm; 48 mm; 62 mm)}. The order of these training points is important because it can result in performance variations.

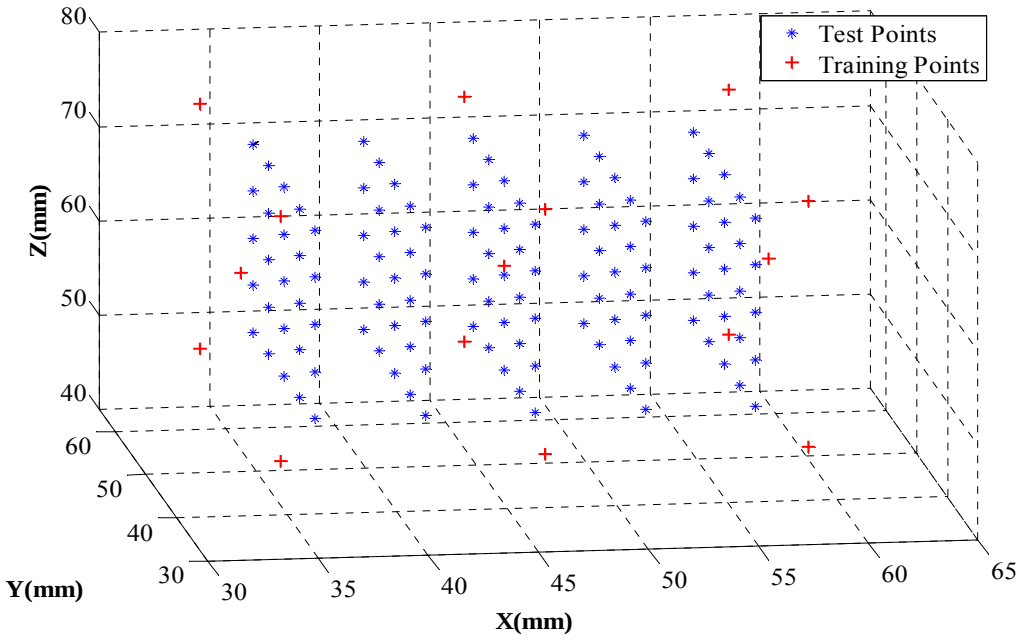


Figure 5.12 - Distribution of 15 training patterns following the free interference rule in the online retraining process.

Figure 5.13 shows the RBFN performance after retraining with 15 training points. This online retraining process corresponds to a learning rate of 0.2 and a spread of 10 mm. Compared to the performance of the incorrect approximation function (before the retraining phase), this result shows that after online retraining with 15 training points the network's performance was noticeably improved. This can be verified by:

- RMS error = 1.55 degrees,
- MAE_X = 1.1 mm,
- MAE_Y = 0.85 mm,
- MAE_Z = 2.23 mm.

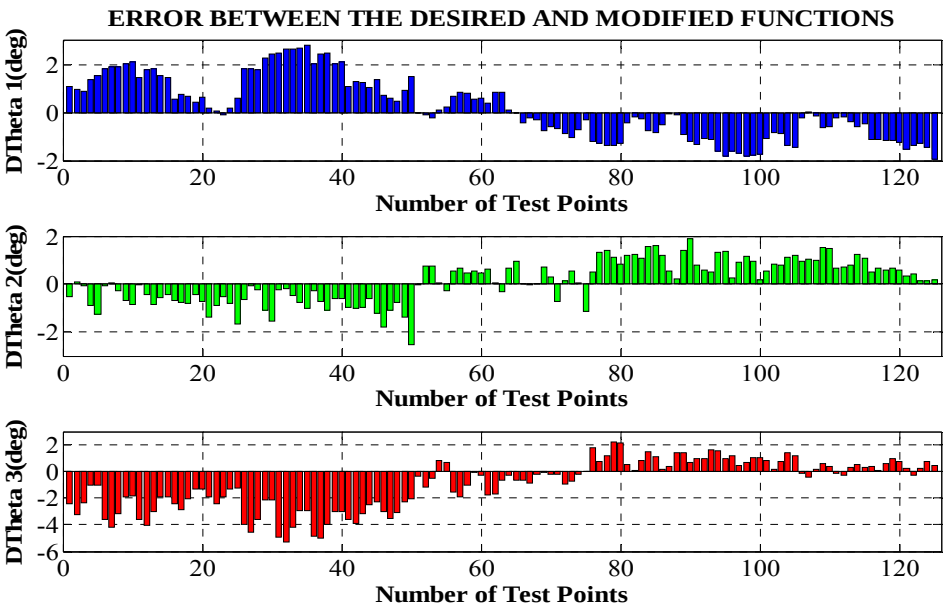


Figure 5.13 - The network's performance after retraining with 15 training points.

The effect of spread and learning rate on the retraining process is shown in Table 5.2. A large spread (e.g., 12mm - 14 mm) requires a small learning rate to produce a small RMS error. However, when using a spread of 14 mm, a small increase in the learning rate leads to a significant increase in the RMS error. Thus, an RBFN with a large spread value is not appropriate in the online retraining phase. A small spread value (e.g., 8 mm to 10 mm) requires a learning rate in the range from 0.1 to 0.4 to minimise the RMS error. For example, a learning rate of 0.2 produces the best network's performance (minimum RMS error) with a spread of 10 mm. The selection of the spread and the learning rate should be carefully considered to ensure a successful retraining process.

Spread / **L.r**	**8 mm**	**10 mm**	**12 mm**	**14 mm**
0.1	3.43	1.72	1.02	0.79
0.2	2.36	1.55	1.13	1.11
0.3	1.83	1.56	2.57	11.6
0.4	1.68	1.72	5.65	32.82
0.5	1.71	2.96	6.65	552.47

Table 5.2 - RMS errors (degrees) after online retraining with 15 points.

The retraining process can be continued to obtain further improvement. Figure 5.14 presents the performance results of the RBFN after retraining with 14 more points. These were: {(52 mm; 42 mm; 55mm), (40 mm; 42 mm; 55 mm), (40 mm; 55 mm; 55mm), (52 mm; 55 mm ; 55 mm), (52 mm; 55 mm; 68 mm), (52 mm; 42 mm; 68 mm), (40 mm; 42 mm; 68 mm), (40 mm; 55 mm; 68 mm), (46 mm; 35 mm; 62 mm), (34 mm; 48 mm; 62 mm), (46 mm; 61 mm; 62 mm), (58 mm; 48 mm; 62 mm), (46 mm; 48 mm; 74 mm), (46 mm; 48 mm; 48mm)}. The total number of online retraining steps was 29 and the improved performance is verified by:

- RMS error = 1.06 degrees,
- MAE_X = 0.71 mm,
- MAE_Y = 0.65 mm,
- MAE_Z = 1.44 mm.

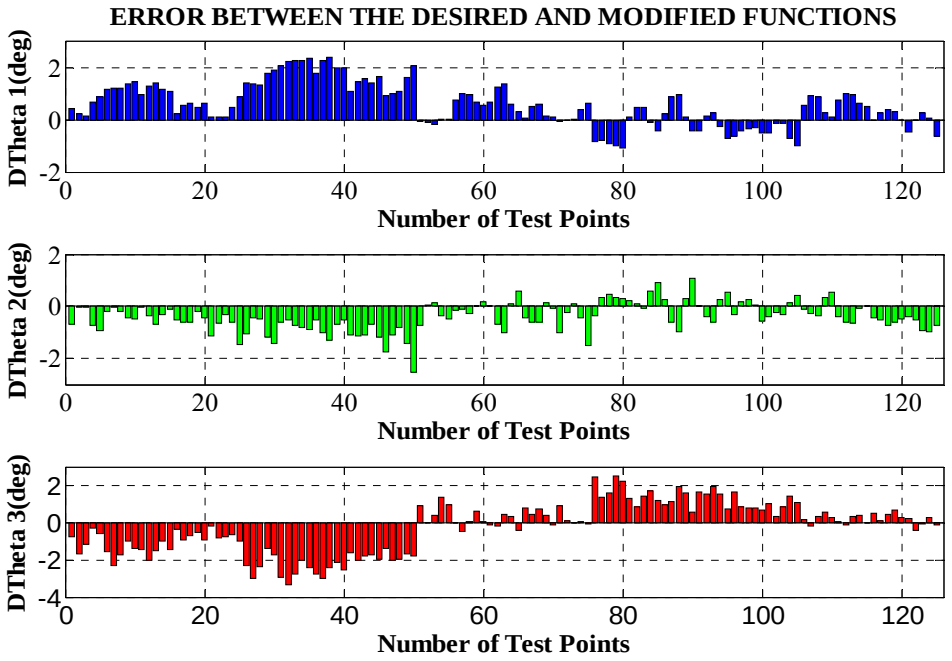


Figure 5.14 - The network's performance after retraining with 29 training points.

The robotic system in this simulation (three-link manipulator) has a more complex structure than the two-link manipulator. Therefore, the improvement of the RBFN in the online retraining phase is slower and requires more training patterns to achieve the same performance improvements. For the three-link manipulator case, the RBFN modified by the online retraining phase could obtain a similar level of performance as the RBFN trained by the strict interpolation method with correct training data (Chapter 4). This observation is valid as long as the number of training points is sufficient and the spread and learning rate are appropriately chosen.

5.6 Conclusion

This chapter has presented a novel approach where the inverse kinematics approximation is modified through an online retraining process. The simulations demonstrate that the RBFN performance after online retraining noticeably improves. There are three factors: the learning rate, the spread value and the position of the training points that can affect the online retraining phase. Thus, the choice of learning rate and the spread value must be carefully considered. The selection of training points following the free learning interference rule can produce better results. However, the effect of online retraining on the RBFN performance is dependent on how complex the desired function is. For a more complex function (e.g., the inverse kinematics of the three-link manipulator) the RBFN requires to be re-trained with more training patterns and the improvement is slower compared to a simpler function (the inverse kinematics of a two-link manipulator). This online retraining approach can be effectively applied when the structure of the practical robotic system alters due to environmental variations (Chapter 7).

5.7 References

[5.1] S. Haykin, *Neural Networks a Comprehensive Foundation – Second Edition*, Prentice Hall, 1999.

[5.2] G. W. Irwin, K. Warwick, and K. J. Hunt, *Neural Network Applications in Control*, IEE Control Engineering Series 53, 1995.

[5.3] M. J. L. Orr, *Introduction To Radial Basis Function Networks*. [Online]. Available: http://www.anc.ed.ac.uk/rbf/rbf.html. 1996.

[5.4] N. Murataa, M. Kawanabeb, A. Zieheb and S. Amari, "Online Learning in Changing Environments with Applications in Supervised and Unsupervised Learning", *Neural Networks*, vol. 15(4), June 2002, pp. 743–760.

[5.5] R. S. Sutton and S. D. Whitehead, "Online Learning with Random Representations", in *Proc. of the 10th Int. Conf. on Machine Learning*, 1993, pp. 314-321.

[5.6] D. H. Rao, M. M. Gupta and P.N. Nikiforuk, "Online Learning of Robot Inverse Kinematic Transformations", in *Proc. of 1993 Int. Joint Conf. on Neural Networks*, 1993, pp. 2827-2830.

CHAPTER 6
DEVELOPMENT OF A THREE-DIMENSIONAL POSITIONAL MEASUREMENT SYSTEM

6.1 Introduction

In robotic control, measuring the state variables (joint angle positions and velocities) in the joint space is simple and direct by using optical sensors attached to the shaft of the joints. However, measuring the position and velocity of the end-effector in the world space (Cartesian coordinates) is significantly more difficult due to the need for an indirect distance measurement system using sonic or vision sensors. In recent years, vision-based measurements have been developed and applied more frequently in robotic control because of benefits, such as efficiency, accuracy and low cost. In this chapter, a real-time visual measurement system based on a video camera is presented to estimate the position of the end-effector of a robotic manipulator in a three-dimensional workspace. It consists of a standard video camera (Webcam) mounted on a fixed pole to measure the position of a sample board attached to the end-effector. Image processing software has been programmed using functions from the Intel Open Source Computer Vision Library (OpenCV). A Graphic User Interface (GUI) has been developed to make this visual measurement tool more convenient for practical applications.

This chapter firstly describes some background information on computer vision and image processing. It includes the pinhole camera model which is the basis of calibration methods of physical cameras. Next, a camera calibration procedure using a calibration toolbox in MATLAB is presented to estimate the intrinsic parameters of the camera. A real-time visual measurement solution to estimate the position of the robotic manipulator in a 3-D workspace is then described. The set-up of the measurement system components and the specific features of the image processing software are presented.

6.2 Background of computer vision

Using a camera to transform a 3-D scene (world space) to a 2-D space (image plane) is important for vision-based measurement. This is performed by a digital camera where an object is digitally captured by recording images via an electronic image sensor. The relationship between the 3-D and 2-D data of the object is then used to determine the geometrical parameters of the camera. This section presents the general background of computer vision related to a real-time vision based measurement system presented in this chapter.

6.2.1 Image acquisition and processing

One of the most popular digital cameras at present is the CCD (Charge coupled device) [6.1] photo-sensor type. Each sensor (photocell) can be regarded as a small rectangular-black box that converts light energy into data as voltage levels. The quality and cost of a camera is likely to relate to how many CCD sensors are attached in an area unit called the camera resolution. In this research, a standard CCD-type webcam (i.e., a digital camera connected to a computer) is used to capture the scenes of an object (robot) in the world space. It records objects' images by scanning the photo-sensors and then producing video signals. These signals are continuously sent and stored in a memory buffer following a specific sequence in the CCD-cell array, normally line by line. Thus, the video stream is transferred to a computer as digital image frames, each consisting of (N x M) data, at a speed of up to 30 frames per second.

A digital image is composed of a number of discrete image units called pixels and is organised as a two-dimensional array (NxM) to build an image plane [6.2]. Figure 6.1 presents the general structure and geometry of a digital image. An element I[i, j] of the digital image represents a value (image brightness) at the i^{th} row, j^{th} column pixel corresponding to coordinates (x_p, y_p) in the image plane. Note that image coordinates and indices of image data at one pixel are not the same, although both can be measured in pixel units. The image coordinates are defined with respect to the origin of the image plane - the principal point (centre). The indices of image data are determined with respect to the left upper corner of the image array.

If the image in Figure 6.1 is a monochromatic (grey) image, then I[i, j] occupies a one-byte memory to store an integer value in the range [0, 255]. The element (i, j) of a colour image consists of three separated memory boxes to store three image-colour components (red, green, blue) [6.2], [6.3] as shown in Figure 6.2. Hence, a three-dimensional array I[i, j, k] with size (NxMx3) is used to represent the colour image in which the third index specifies a particular colour.

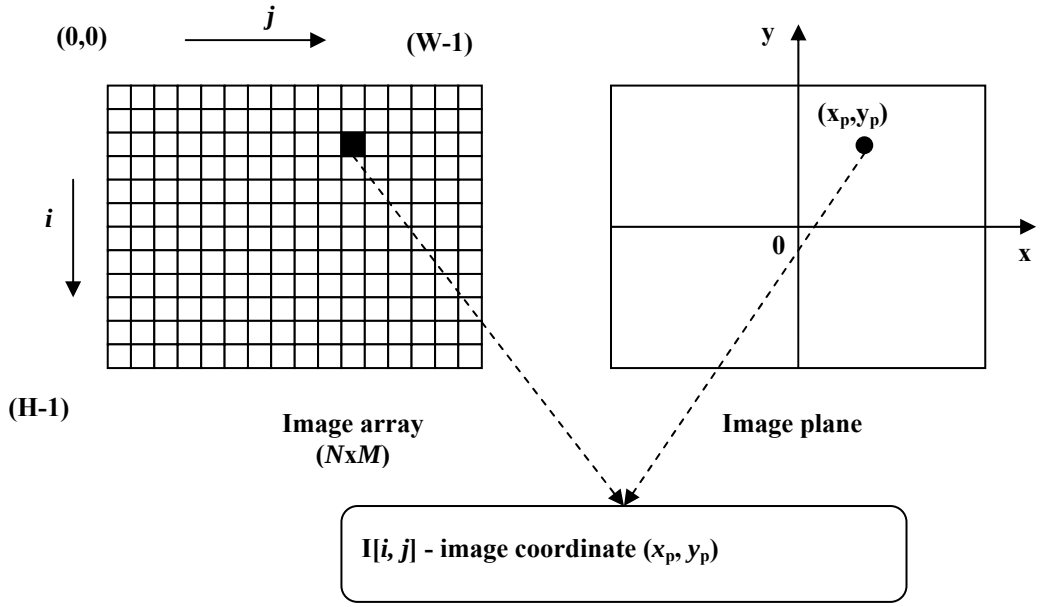


Figure 6.1 - Relationship between the image plane and image array of a digital image.

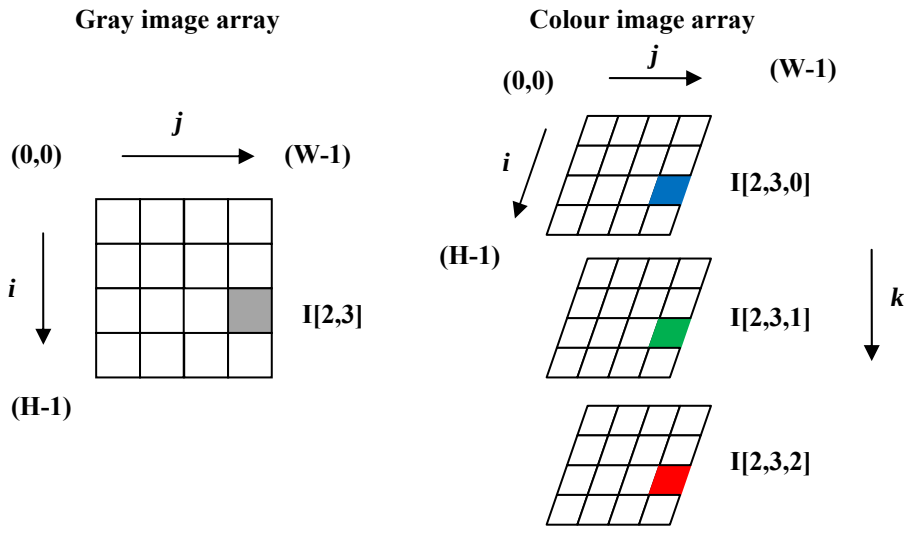


Figure 6.2 - The structure of image data for a gray and a colour image.

Colour image information is organised as a multi-dimensional array according to a particular colour space where RGB (Red-Green-Blue) is the most popular. It consists of three colour channels, red, green and blue, each of which is represented by a specific value (colour part) within the interval [0, 255]. By combining the three colour parts in this format, the vision features (e.g., sharpness, colour, shadow, etc.) of the object can be displayed. However, the RGB colour space is not stable with regards to alterations in the illumination because the representation of a colour in the RGB colour space contains no separation between the illumination and the colour parts [6.3]. It means that with an illumination point (e.g., a light emitting diode) in either red, green, or blue, all colour parts have the same brightness of 255. This problem has affected the performance of the visual measurement system developed in this research where the data acquisition programme is required to distinguish between two different-colour points in the calibration sample. In order to eliminate this phenomenon, some adjustments of the camera specifications (hardware) such as using lower brightness, reducing contrast level and applying a colour filter function (software) have been implemented.

Most of the image processing functions to analyse image features (e.g., edge detector, contour fragments, corner extraction) [6.1], [6.2] have been developed with gray-scale or binary image data. Thus, a colour image should be first converted to the suitable image space before processing. The threshold value of the colour conversion functions should be selected experimentally depending on the particular situation.

6.2.2 Perspective transformation from 3-D to 2-D space

The perspective transformation is based on the pinhole camera model where each point in the world space is projected by a straight line through the projection centre into the image plane [6.2]. Figure 6.3 shows the projection of a point P(X_P, Y_P, Z_P) into the image plane. It produces an image point p(x, y) correspondingly.

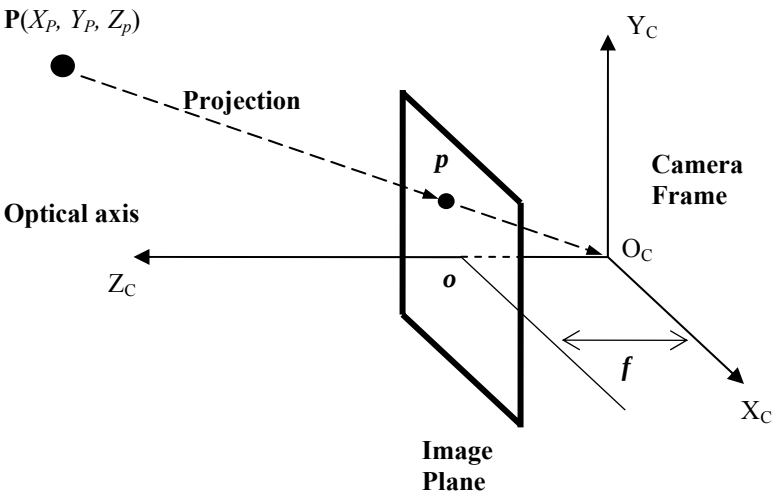


Figure 6.3 - The pinhole model – projection of a 3-D point into the image plane.

A 3-D coordinate frame attached in the projection centre of the camera is called the camera coordinate system where the Z_c axis is perpendicular to the image plane. This camera coordinate frame can be determined with respect to the object coordinate system which is attached to an object by a translation vector $\boldsymbol{T}$(1x3) and a rotation matrix $\boldsymbol{R}$(3x3). Figure 6.4 expresses the relationship between the world (object) coordinate system and the camera coordinate system.

In order to determine image coordinates of a point $P(X_P, Y_P, Z_P)^W$ given in the world coordinate system, it is first transformed to the camera coordinate system as

$$\begin{bmatrix} X_P^C \\ Y_P^C \\ Z_P^C \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} X_P^W \\ Y_P^W \\ Z_P^W \end{bmatrix} + \boldsymbol{T} \tag{6.1}$$

or

$$\begin{bmatrix} X_P^C \\ Y_P^C \\ Z_P^C \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_P^W \\ Y_P^W \\ Z_P^W \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}. \tag{6.2}$$

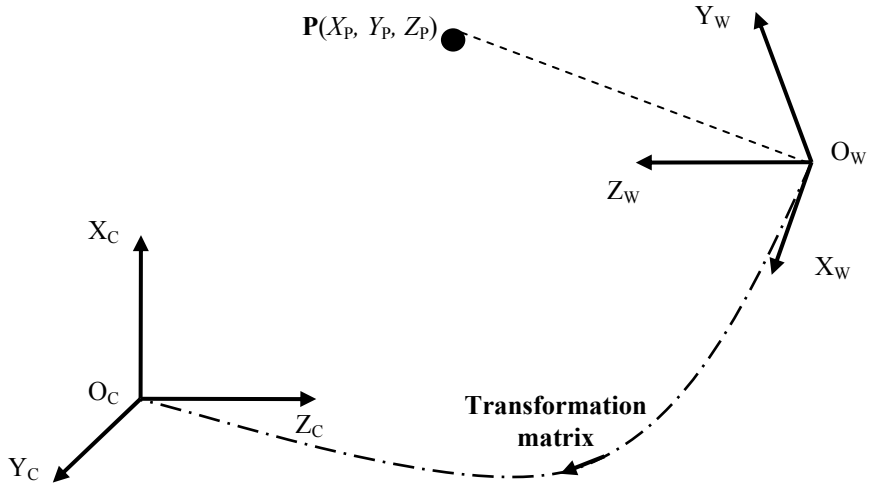


Figure 6.4 - Transformation between the world coordinate frame and the camera coordinate frame.

The translation vector $\boldsymbol{T}$ and orthogonal matrix $\boldsymbol{R}$ ($\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}$) specify a homogenous transformation from one reference frame to another and are commonly called the camera extrinsic parameters. A projection from the camera coordinate frame through the optical lens into the image plane is then implemented and the corresponding image coordinates are determined by

$$x = f\frac{X^c}{Z^c} \tag{6.3}$$

$$y = f\frac{Y^c}{Z^c}. \tag{6.4}$$

Note that these coordinates (x, y) are defined with respect to the origin of the image plane (centre of the image plane). However, the position of an image point is always determined by counting the indices of the image array in pixel units, (u, v), as shown in Figure 6.1. Thus, if the distortions of the optical lens are neglected, the relationship between the 3-D and 2-D image coordinates (in pixels) can be expressed as

$$u = -\frac{f}{s_u}\frac{X^c}{Z^c} + o_u = -f_x\frac{X^c}{Z^c} + o_u \tag{6.5}$$

$$v = -\frac{f}{s_v}\frac{Y^c}{Z^c} + o_v = -f_y\frac{Y^c}{Z^c} + o_v \tag{6.6}$$

where (o_u, o_v) is the coordinate of the image centre in pixels and (s_u, s_v) are the scale coefficients of the pixels (in millimetres) in the horizontal and vertical directions respectively.

However, this pinhole model is only an approximation for a real projection. In fact, all optical lenses have some distortions affecting the projection. It is essential that some coefficients can be attached to compensate for this phenomenon. In many cases, the radial lens distortion [6.4] is likely to be a main cause of displacement in the image coordinates. It can be expressed as

$$dx = x - x_d = x_d(k_1 r^2 + k_2 r^4 + ...) \tag{6.7}$$

$$dy = y - y_d = y_d(k_1 r^2 + k_2 r^4 + ...) \tag{6.8}$$

where (x_d, y_d) is the coordinate of the distorted point and $r^2 = x_d^2 + y_d^2$.

Typically, radial distortion coefficients are small and often $k_2 << k_1$, so in most practical cases only coefficient k_1 is adopted as an intrinsic parameter [6.2]. As a result, the intrinsic parameters of a camera includes:

- f_x , f_y – focal lengths in effective horizontal and vertical pixel size units,
- (o_u, o_v) - image centre coordinate in pixels, and
- k_1 – radial distortion coefficient.

The projection from the object coordinate system to the image plane can also be expressed in a more general form by a linear matrix such as

$$\begin{bmatrix} u.w \\ v.w \\ w \end{bmatrix} = \mathrm{M} \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} \tag{6.9}$$

where

$$\boldsymbol{M} = \begin{bmatrix} -f_x r_{11} + o_u r_{31} & -f_x r_{12} + o_u r_{32} & -f_x r_{13} + o_u r_{33} & -f_x T_x + o_u T_z \\ -f_y r_{21} + o_v r_{31} & -f_y r_{22} + o_v r_{32} & -f_y r_{23} + o_v r_{33} & -f_y T_y + o_v T_z \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix}. \tag{6.10}$$

$\boldsymbol{M}$ is called the perspective transformation matrix where w is an arbitrary scale value [6.2]. This can be rewritten as a product of two simpler matrices, $\boldsymbol{M}_{\text{int}}$ and $\boldsymbol{M}_{\text{ext}}$, as shown by

$$\boldsymbol{M} = \boldsymbol{M}_{\text{int}} . \boldsymbol{M}_{ext} \tag{6.11}$$

where

$$\boldsymbol{M}_{\text{int}} = \begin{bmatrix} -\dfrac{f}{s_u} & 0 & o_u \\ 0 & -\dfrac{f}{s_v} & o_v \\ 0 & 0 & 1 \end{bmatrix} \tag{6.12}$$

$$\boldsymbol{M}_{ext} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix}. \tag{6.13}$$

$\boldsymbol{M}_{ext}$ performs the transformation between the object coordinate frame and the camera coordinate frame and $\boldsymbol{M}_{int}$ performs the projection from the camera coordinate frame to the image plane. Therefore, this perspective matrix can be defined, dependent on an arbitrary scale factor w, by 11 independent parameters, 6 extrinsic and 5 intrinsic.

6.3 Camera calibration methods

6.3.1 Overview

Camera calibration is the estimation of the intrinsic and extrinsic parameters of the camera model from a set of known coordinates of 3-D objects and their corresponding images [6.6]-[6.10]. As presented in Section 6.2.2, projecting an object from the world space to the image plane can be expressed by the perspective transformation matrix on the basis of the pinhole camera model. However, this transformation cannot be described perfectly due to optical lens distortions. Thus, the pinhole model is only an approximation to the real physical camera. For this reason, the transformation procedure should be modified by using additional intrinsic parameters to model the lens distortions. In [6.4], [6.5] a literature review of calibration methods and various distortion models was presented. It listed a range of existing camera calibration approaches and mentioned various lens distortion models (e.g., radial and tangential), as well as how these distortions affect image displacements in the image plane. A calibration method called the direct linear transformation which uses the pinhole model and ignores the nonlinear radial and tangential distortion components was described in [6.2], [6.6]. A nonlinear estimation approach [6.6] applied an optimisation method which minimises the distances between perspective model images, calculated by equation (6.9), and actual measured images to estimate the camera parameters. The radial and tangential distortion had also been added in the constraint equations. This is a nonlinear optimisation problem due to the vision-geometry relationships and an iterative approach using the Levenberg-Marquardt method was applied to simultaneously estimate the camera parameters. A similar nonlinear estimation solution called the maximum-likelihood estimate was proposed in [6.7]. It built the constrained equations in the least squares manner with a simplification in the homograph matrix between the model plane and its images. Most of the camera calibration tools combine the direct linear transformation and nonlinear estimation together. Firstly, an initial phase is implemented based on the direct linear transformation without any distortion

components, then the nonlinear estimation is applied to refine all camera parameters including distortion components. Given a sufficient number of calibration patterns, a camera-parameter estimation approach can be implemented as follows.

If a set of N planar points (3-D space) and their corresponding image coordinates (2-D space) are known, by applying (6.10) a matrix equation which represents the relationships between the N pairs of 3-D and 2-D points can be written as

$$\boldsymbol{A}\boldsymbol{m} = 0 \tag{6.14}$$

where the coefficient matrix $\boldsymbol{A}$ is determined by

$$\boldsymbol{A} = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -u_2X_2 & -u_2Y_2 & -u_2Z_2 & -u_2 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -v_2X_2 & -v_2Y_2 & -v_2Z_2 & -v_2 \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -u_NX_N & -u_NY_N & -u_NZ_N & -u_N \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -v_NX_N & -v_NY_N & -v_NZ_N & -v_N \end{bmatrix} \tag{6.15}$$

and the variable vector is

$$\boldsymbol{m} = \left[m_{11}, m_{12}, m_{13}, m_{14}, m_{21}, m_{22}, m_{23}, m_{24}, m_{31}, m_{32}, m_{33}, m_{34}\right]. \tag{6.16}$$

By replacing the known object coordinates (X_i, Y_i, Z_i) and image coordinates (u_i, v_i) in (6.15), the parameter vector $\boldsymbol{m}$ can be estimated based on the least squares technique. It is performed by applying the SVD (Singular Value Decomposition) transformation for matrix $\boldsymbol{A}$

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T. \tag{6.17}$$

Since matrix $\boldsymbol{A}$ has rank 11, the solution of vector $\boldsymbol{m}$ can be obtained using the SVD technique as a column of matrix $\boldsymbol{V}$ corresponds to the smallest singular value of $\boldsymbol{A}$ [6.2]. This solution is formed by the entries of the projection matrix dependent on an unknown scale factor. Once $\boldsymbol{m}$ is estimated, the camera intrinsic parameters can be computed from the perspective matrix and the extrinsic parameters are then determined correspondingly [6.2].

In order to refine the camera parameters for real conditions where noise is present, a nonlinear estimation technique can be performed which minimises the error between the model plane coordinates $(\tilde{u}_i, \tilde{v}_i)$ and measured coordinates (u_i, v_i). If it is assumed that the image points are corrupted by noise with a normal distribution (Gaussian noise), the cost function is expressed as a sum of squared errors [6.7]:

$$F = \sum_{i=1}^{N} (u_i - \tilde{u}_i)^2 + \sum_{i=1}^{N} (v_i - \tilde{v}_i)^2 \tag{6.18}$$

where $(\tilde{u}_i, \tilde{v}_i)$ are dependent functions of all camera parameters including distortions. Equation (6.18) is a nonlinear function and an iterative algorithm can be used to minimise it. The camera parameters estimated by the direct linear transformation are used as the initial values in the optimisation algorithm.

The precision of the calibration algorithm depends on how accurately the coordinates of 3-D objects and their corresponding images can be located. This is highly dependent on which sample type and image-extraction technique are applied to obtain adequate data.

In order to provide a range of 3-D objects for a calibration algorithm, some samples called the calibration patterns are used with known geometry. These are located in a known position in the world space so that their image coordinates are easily determined from the image plane. Figure 6.5 shows two popular types of calibration patterns. Both provide a set of points as planar grids and the corresponding image positions of these points in the image plane can be easily determined by any common image processing toolbox, e.g., MATLAB Image processing toolbox or OpenCV.

Chessboard **Planar light points**

Figure 6.5 – Two popular types of calibration patterns used in computer vision.

Recently, many of the calibration samples in 3-D reconstruction/measurement fields have used structured light systems [6.11]-[6.13]. The term structured light is defined as the projection of simple or encoded light patterns (i.e., points, lines, grids, complex shapes) [6.11] onto the illuminated object. The main benefit of using structured light is that features in the images are better defined. As a result, both the detection and extraction of image features are simplified and are more robust. The quality of camera calibration algorithms can be improved significantly.

6.3.2 Camera calibration toolbox in MATLAB

An interesting toolbox with full instructions for camera calibration in MATLAB has been developed [6.14]. Most functions in this toolbox are also included in the OpenCV library which was used to programme the real-time visual measurement software in this research.

This camera calibration toolbox was used to estimate the intrinsic parameters of the Trust 380 USB 2.0 SPACEC@M webcam used in this research. The calibration procedure was implemented as follows:

- Prepare a chessboard pattern as in Figure 6.5 and attach it to a planar surface.

- Capture several images of the pattern with the webcam. The pattern is changed by varying the viewing angles and distances with respect to the camera to collect a range of different images. The calibration toolbox is then run to load images

(*Read images function*). As suggested in the toolbox instructions, this calibration procedure should be implemented with about 20 images (or more) to obtain an acceptable result.

- Detect the feature points in the image planes by using the *Extract grid corners function*. The four extreme corners on the rectangular chessboard pattern are then manually selected to bound the effective area and to assign the origin of the object coordinate system. The corner extraction engine includes an automatic mechanism for counting the number of squares in the grid. This tool is especially convenient when working with a large number of images since the user does not have to manually enter the number of squares in both the *x* and *y* directions of the pattern.

- After corner extraction, the camera calibration function is then performed. The calibration procedure is done in two phases: initialisation and then nonlinear optimisation. The initialisation phase computes the calibration parameters by the direct linear transformation without any lens distortion. The nonlinear optimisation phase iteratively minimises the total re-projection error between measured and image coordinates (which are calculated from the camera model estimated in the pervious iteration step) for all the calibration parameters. This toolbox uses an optimisation approach based on the gradient descent method.

Calibration result

Focal Length (pixels): f_C = [838.65; 833.57] ± [4.31; 4.7].

Principal point (pixels): O_C = [327.16; 268.84] ± [7.49; 6.48].

Skew: alpha_c = [0.00] ± [0.00] => angle of pixel axes = 90 ± 0.00 degrees.

Distortion: k_C = [- 0.22; 0.09; 0.0017; 0.0017 ; 0] ± [0.029; 0.197; 0.0016; 0.0017; 0].

Pixel error: err = [0.293; 0.422].

This calibration result has been used for the visual measurement tool developed in this research. The MATLAB calibration toolbox has been used to verify the performance of this real-time visual measurement system.

6.4 A real-time 3-D measurement based video camera

This section presents a real-time visual measurement system which consists of a camera and image processing software to measure the position of a sample board attached in the end-effector. The software is programmed in C++ using functions of the OpenCV library. A GUI makes this visual measurement tool more convenient for practical applications.

6.4.1 Set-up of the vision-based measurement system

The first component of the visual measurement system is the sample board. Most 3-D vision-based measurement systems have used structured light systems as the calibration sample [6.15], [6.16]. In [6.11] a range of popular types of structured light samples in computer vision were reviewed. Such systems are often composed of one or two cameras and a projector which emits structured light patterns onto the object surface. The projector can produce structured light patterns which are easily distinguished or detected by their different features. The cameras then capture and process images of the structured light sample. This system has been very popular in 3D reconstruction/measurement fields where accuracy depends on the technology of the light emitting devices (liquid-crystal or laser-based) and the camera resolution [6.12]. However, this is costly and inconvenient to attach to the manipulator. A low-cost sample was built in which four light points (LEDs), in two different colours, were used to form a calibration pattern. This sample board consists of four points fitted in a square of 61 mm x 61 mm as shown in Figure 6.6. It is small and light enough to fix to the end-effector without changing its dynamic characteristics and is simple for real-time processing. However, using the LEDs as active light sources for the visual measurement system can lead to some disadvantages. Specifically, high illumination, lack of distinction between coloured parts and interference due to the fluctuation of the power supply can affect the accuracy of the visual measurement system.

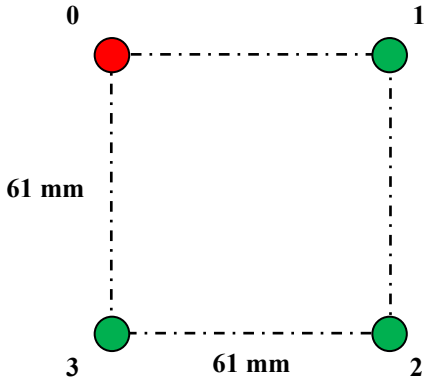


Figure 6.6 – Sample board attached to the end-effector as the calibration pattern.

The second component of the visual measurement system is a Trust 380 USB 2.0 SPACEC@M webcam (USB port, 640 x 480 resolution, 30 frames/second video stream) which was used to capture images of the sample board attached at the end-effector.

Figure 6.7 shows the structure of the robotic system consisting of the manipulator and the visual measurement system.

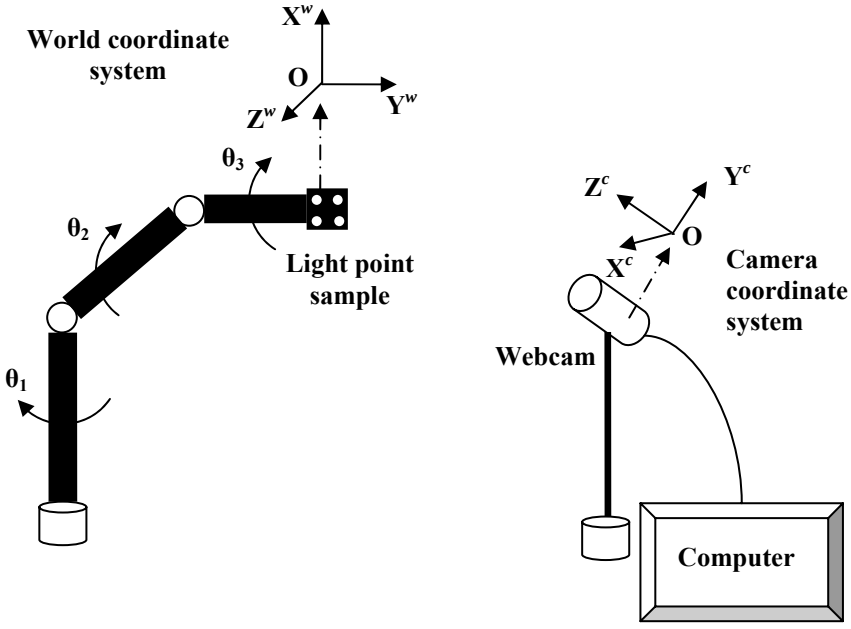


Figure 6.7 – Three-dimensioned visual measurement system.

The position of the origin of the sample board (i.e., point 0 in Figure 6.6) is seen as the position of the manipulator end-effector with respect to the camera reference system. Thus, the reference point has to be distinguished from other points on the image plane. To achieve this, LEDs with two different colours, red for the origin point and green for the others were used. This solution together with a new approach to extract image points based on the angle comparison is a convenient approach for 3-D vision based measurement. This is distinct from other solutions mentioned in [6.3] where the discrimination was based on the difference of sizes or edge distances between the sample points. Other solutions mentioned in [6.11]-[6.13] using coded strips for the structured light system, which is costly due to the expensive projector and active camera. Therefore, the proposed approach is simple, low cost and practically convenient.

6.4.2 OpenCV library

OpenCV, a popular image processing library, was used to develop the image processing software. It is a collection of C/C++ functions and classes of popular Image Processing and Computer Vision [6.17] algorithms. OpenCV is free for both non-commercial and commercial use. It consists of the modules:

- *cv* - Main OpenCV functions.
- *cvaux* - Auxiliary (experimental) OpenCV functions.
- *cxcore* - Data structures and linear algebra support.
- *highgui* - GUI functions.

As presented in [6.18], the OpenCV library includes image processing functions and computer vision algorithms for:

- Image data manipulation (allocation, release, copying, setting, conversion).
- Image and video I/O (file and camera based input, image/video file output).

- Matrix and vector manipulation and linear algebra routines (products, solvers, eigenvalues, SVD).

- Various dynamic data structures (lists, queues, sets, trees, graphs).

- Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, colour conversion, morphological operations, histograms, image pyramids).

- Structural analysis (connected components, contour processing, distance transform, various moments, template matching, Hough transform, polygonal approximation, line fitting, ellipse fitting, Delaunay triangulation).

- Camera calibration (finding and tracking calibration patterns, calibration, fundamental matrix estimation, homography estimation, stereo correspondence).

- Motion analysis (optical flow, motion segmentation, tracking).

- Object recognition.

- Basic GUI (display image/video, keyboard and mouse handling, scroll-bars).

- Image labelling (line, conic, polygon, text drawing).

6.4.3 Image processing software for 3-D visual measurement

Software was developed using the OpenCV library, Visual C++ and displayed as a GUI to produce a convenient interface for users. Figure 6.8 presents the flowchart of this software.

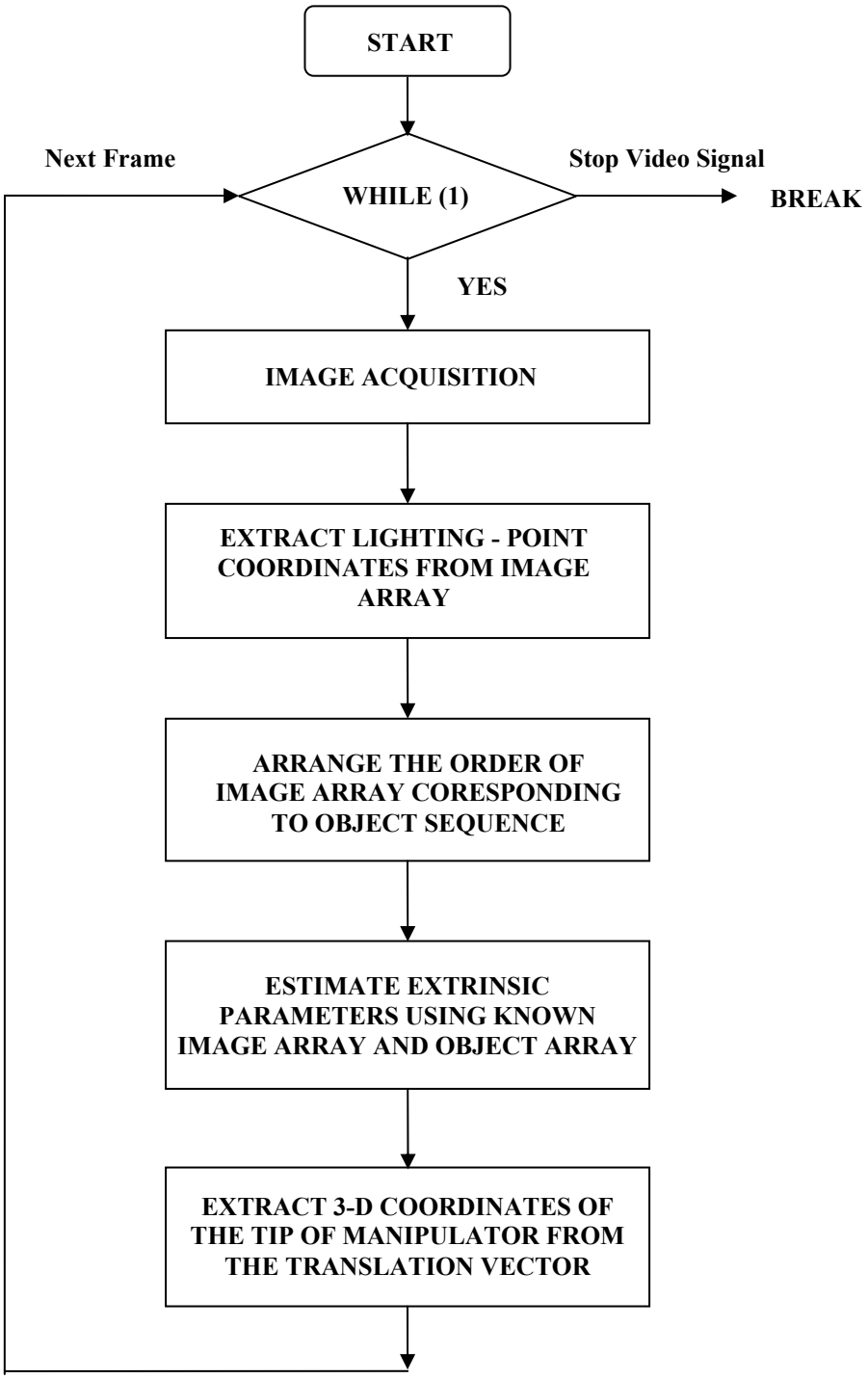


Figure 6.8 - Flowchart of the image processing software.

Image acquisition: This sub-programme starts capturing images of the sample board in the workspace. A video stream at a speed of 30 frames/second is established. Each frame creates a 3-D image data array in the RGB colour space using the order [Blue, Green, Red] as defined in [6.18].

Extract image point coordinates: This sub-programme extracts the image coordinates of 4 light points in the image plane (Figure 6.9). Image data are first filtered to recognise the different colour points and then converted to binary space (black and white pixels) to determine the areas of these points. Contours of the four light points are extracted from the background. Each point centre is determined based on its own contours. In this step, the order of these image points follows the order of contour sequence (the order in which each point contour was loaded [6.18]). This order is obviously different from the order of object data in the object-position vector.

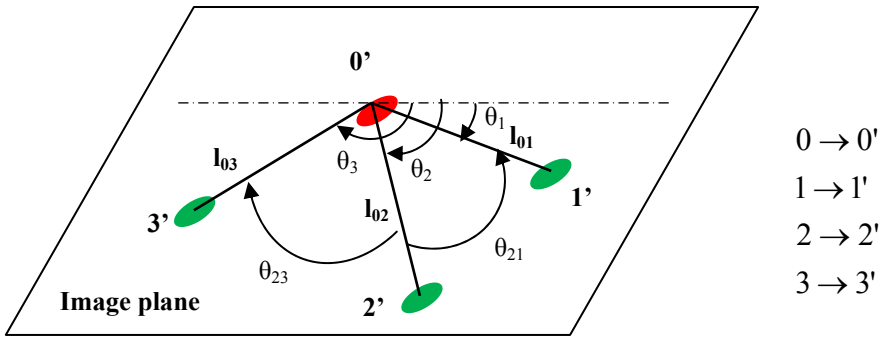


Figure 6.9 – Images of the sample points in the image plane.

Arrange the order of lighting point images corresponding to the sequence of light point in the sample board: This is a novel solution to distinguish the different image points based on an angle comparison algorithm. The aim of this sub-programme is to match each image point (Figure 6.9) with its corresponding object point (Figure 6.6) as required by the estimation algorithm. It is performed by the following procedure.

1. Recognise image point 0'of the point 0 based on its particular colour.

2. Calculate the distances in pixel units between image point 0' and all other image points. The longest distance corresponds to image point 2' and the two remaining are either point 1' or point 3'.

3. Define two adjacent angles θ_{21} and θ_{23} from three lines 1_{01}, 1_{02}, and 1_{03} as shown in Figure 6.9. According to the real arrangement of the four object points in the sample board, points 1' and 3' (in image plane) are distinguishable by comparing these angles from the following equations:

$$\theta_{21} > 0 \text{ , and } \theta_{23} < 0 \tag{6.19}$$

where

$$\theta_{21} = \theta_2 - \theta_1 \tag{6.20}$$

$$\theta_{23} = \theta_2 - \theta_3 \ . \tag{6.21}$$

θ_1, θ_2 and θ_3 are the angles of lines 1_{01}, 1_{02} and l_{03} respectively.

This algorithm can automatically recognise image points corresponding to the sample object points exactly. It arranges the order of image points in the data array corresponding with the order of sample points in the sample board.

Calculate the extrinsic parameters of camera :

The function **cvFindExtrinsicCameraParams2**(Object_point, Image_point, &Intrinsic_matrix, &Distortion_coeffs, Rotation_vector, Translation_vector) [6.17] is used. The rotation vector and translation vectors are the position and orientation of object points with respect to the camera coordinate system. The position of the manipulator end-effector is the position of the reference point (point 0 in Figure 6.6) in the sample board.

6.4.4 Summary of results

Figure 6.10 presents the image of the sample board attached to the manipulator end-effector on the computer screen. The image processing software has recognised the coordinates of the image points in the correct order to match with the LED points in the sample board.

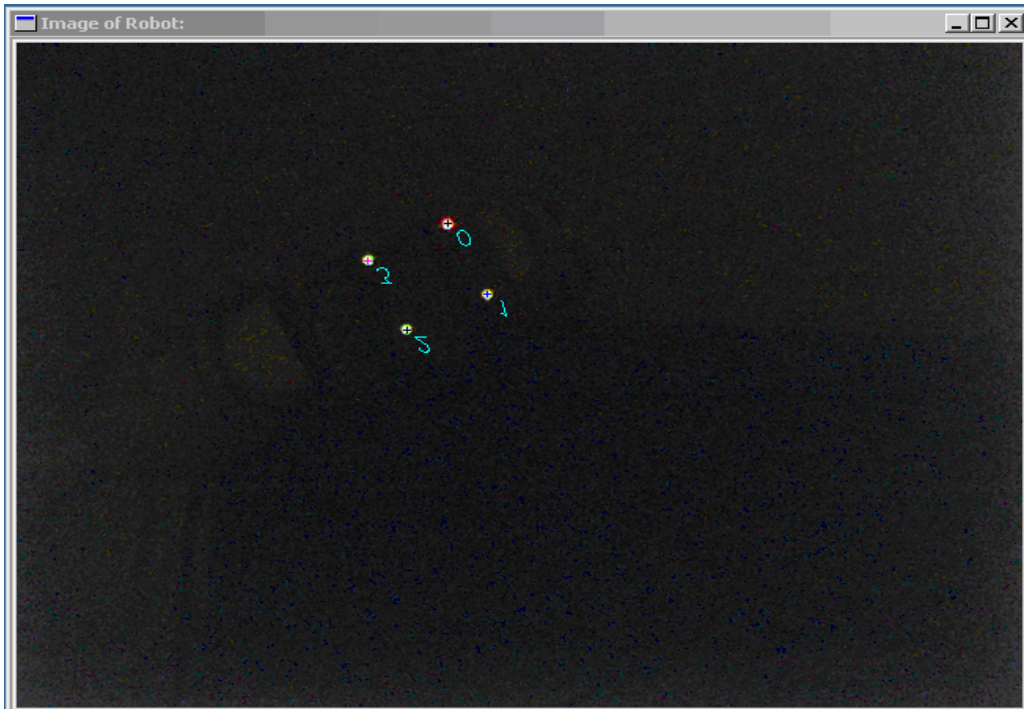


Figure 6.10 - The view of the manipulator end-effector in the camera window.

Figure 6.11 presents the GUI of the visual measurement software where 3-D coordinates of the manipulator are displayed in the window ACTUAL POSITION OF PA10 ROBOT. When the manipulator operates in the workspace, its position is measured almost in real-time with respect to the camera location.

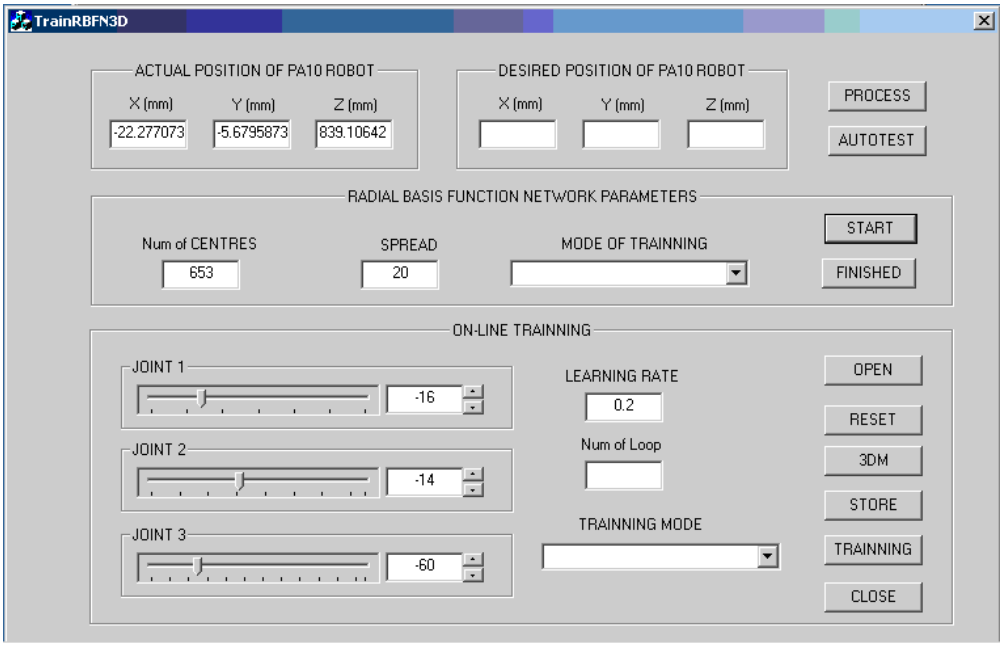


Figure 6.11 - GUI of the application software using the 3-D visual measurement system.

The real-time visual measurement system has been tested by using the Camera calibration toolbox in MATLAB [6.14] with the same data. Both approaches produced similar results. However, as the key factor in determining measurement accuracy is how well images of the light points could be extracted in pixel units, this visual measurement tool is limited due to the low quality of hardware (camera and sample board). The simple sample board was a self-manufactured device with common LEDs whose images vary significantly according to the camera view and fluctuate due to variation in the power supply. For example, two different camera views affect the accuracy of the visual measurement as shown in Figure 6.12. In case (b) the shapes of the LEDs' images are deformed compared to case (a) and the position (coordinates) of image points which are determined as the centre of each point may deviate from the real position. As a result, using this incorrect image data in the estimation algorithm leads to a measurement error. A deviation value of one pixel in the image plane leads to an approximate error of 1.8 mm in the 3-D coordinates corresponding to the camera resolution of 640x480 and the distance between the manipulator and the webcam in the experiment.

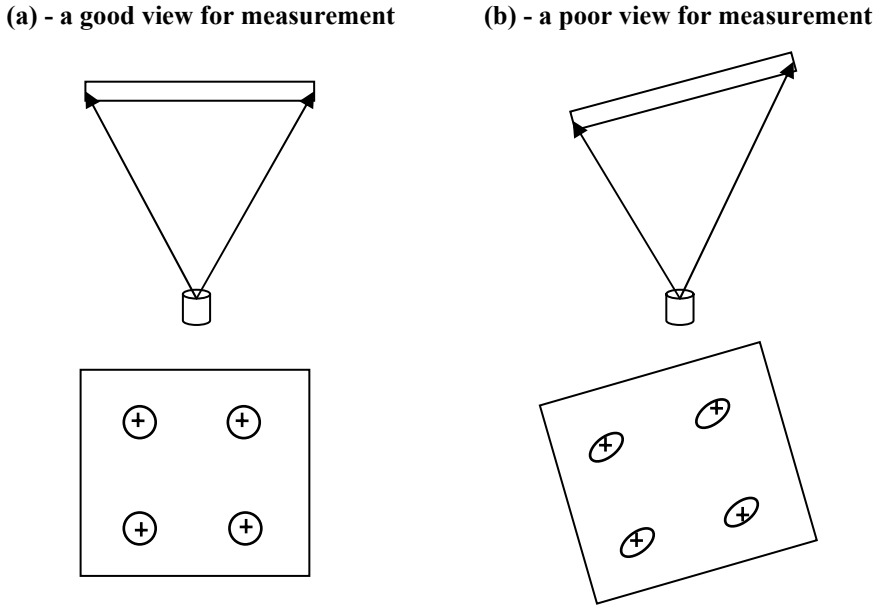


Figure 6.12 - Two views of the sample board with respect to the camera that affect the accuracy of the visual measurement system.

In the practical work, the measured position is not consistent, even with the same set of joint angles sent to the servo-controllers. This situation is more serious in the *Z* direction where the variation was about 6-12 mm, instead of 2-6 mm in the other two directions when measuring the same point multiple times.

This error level is acceptable for practical work and could be improved if a high resolution camera and a better light sample board were obtained.

6.5 Conclusion

A real-time 3-D visual measurement system, consisting of a light sample board, one video webcam and image processing software, has been presented. It includes a novel solution to automatically detect the image data for real-time applications. It is convenient to apply this measurement system to a robot manipulator where the sample board is easily attached to the manipulator end-effector. The proposed system is portable, reasonably accurate and low cost.

6.6 References

[6.1] R. Jain, R. Kasturi and B. G. Schunk, *Machine Vision.* McGraw-Hill, Inc., 1995.

[6.2] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision.* Prentice-Hall, Inc., 1998.

[6.3] S. Florczyk, *Robot Vision Video-based Indoor Exploration with Autonomous and Mobile Robots.* Wiley-VCH, 2005.

[6.4] D. C. Brown, "Close-Range Camera Calibration", *Photogrammetric Engineering*, vol. 37(8), 1971, pp. 855-866.

[6.5] T. A. Clarke, J. F. Fryer and X.Wang, "The principal point and CCD cameras", *Photogrammetric Record*, vol. 16(92), 1998, pp. 293-312.

[6.6] J. Heikkilä and O. Silvén, "A Four-step Camera Calibration Procedure with Implicit Image Correction", in *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, San Juan, Jun. 1997, pp. 1106-1112.

[6.7] Z. Zhang, "Flexible Camera Calibration By Viewing a Plane From Unknown Orientations", in *Proc. of the 7th IEEE Int. Conf. on Computer Vision,* Kerkyra, Sep. 1999, vol. 1, pp. 666 - 673.

[6.8] T. A. Clarke and J. G. Fryer, "The Development of Camera Calibration Methods and Models", *Photogrammetric Record*, vol. 16(91), 1998, pp. 51-56.

[6.9] R. Y. Tsai, "A Versatile Camera Calibration Techniaue for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lense", *IEEE Journal of Robotics and Automation*, vol. RA-3(4), August 1987, pp. 323-344.

[6.10] P. F. Sturm and S. J. Maybank, "On Plane-Based Camera Calibration: A General Algorithm, Singularities, Applications", in *Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, Fort Collins, 1999, pp. 23–25.

[6.11] M. Ribo and M. Brandner, “State of the Art on Vision-Based Structured Light Systems for 3D Measurements”, presented at the *IEEE Int. Workshop on Robotic Sensors: Robotic and Sensors Environments*, Ottawa, 2005, pp. 2-6.

[6.12] C. Rocchini, P. Cignoni, C. Montani, P. Pingi and R. Scopigno, “A low cost 3D scanner based on structured light”, *Eurographics 2001*, vol. 20(3), 2001, pp. 1-6.

[6.13] H. Cui, N. Dai, W. Liao and X. Cheng, “An Accurate Reconstruction Model Using Structured Light of 3-D Computer Vision”, in *Proc. of the 7th World Congress on Intelligent Control and Automation*, Chongqing, June 2008, pp. 5100 – 5104.

[6.14] J. Y. Bouguet, *Camera Calibration Toolbox for MATLAB*. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/.(Last updated June 2nd, 2008).

[6.15] P. Renaud, N. Andreff, J. M. Lavest and M. Dhome, “Simplifying the Kinematic Calibration of Parallel Mechanisms Using Vision-Based Metrology”, *IEEE Trans. on Robotics*, vol. 22(1), Feb. 2006, pp. 12-22.

[6.16] L. Li, Z. Feng, Y. Feng and Q. Peng, “A High Accuracy Camera Calibration for Vision-Based Measurement Systems”, in *Proc. of the IEEE 5th World Congress on Intelligent Control and Automation*, Hangzhou, 2004, pp. 3730-3733.

[6.17] *Open Source Computer Vision Library - Reference Manual*, *Version 04.* Intel Corporation, Dec. 2001.

[6.18] G. Agam, *Introduction to programming with OpenCV*, 2006.

[6.19] X. Wang and T. A. Clarke, “An algorithm for real-time 3-D measurement”, *ISPRS*, vol. 31(part B5), 1996, pp. 587-592.

CHAPTER 7
PRACTICAL INVESTIGATION OF RADIAL BASIS FUNCTION NETWORK PERFORMANCE

7.1 Introduction

A solution using a radial basis function network with regularly spaced position centres to approximate the inverse kinematics transformation of a robotic manipulator was demonstrated through various simulations in Chapters 4 and 5. This chapter demonstrates the proposed approach by applying it to a practical robotic system consisting of a Mitsubishi PA10-6CE manipulator. The experiments have been conducted for two different situations. In the first case, the RBFN has been trained to approximate the inverse kinematics transformation using an offline training phase. In the second case, an additional online retraining phase has been used to cope with a variation in the visual measurement structure of the robotic system. This chapter starts with a brief description of the components of the practical robotic system. The implementation procedures for the practical experiments in two- and three-dimensional space are presented. The results are shown to demonstrate the effectiveness of the proposed approach and conclusions are stated.

7.2 Components of the robotic system

Practical experiments have been developed and performed using the existing facilities in the Intelligent Robotics Laboratory [7.1]. The robotic system is controlled via an Internet interface as shown in Figure 7.1. The following elements of the system can be identified:

- The Mitsubishi PA10-6CE manipulator with servo controller. This is a six-link multipurpose arm connected to an industrial PC (IPC) via an ARC-Net interface.

- The IPC running under the QNX Neutrino real-time operating system is used to execute control programmes and communicate with an application PC (APC) via an Internet interface.

- A standard webcam mounted on a vertical shaft that permits rotation captures the manipulator images in the workspace.

- The main application programmes were written in C++ and run on the APC.

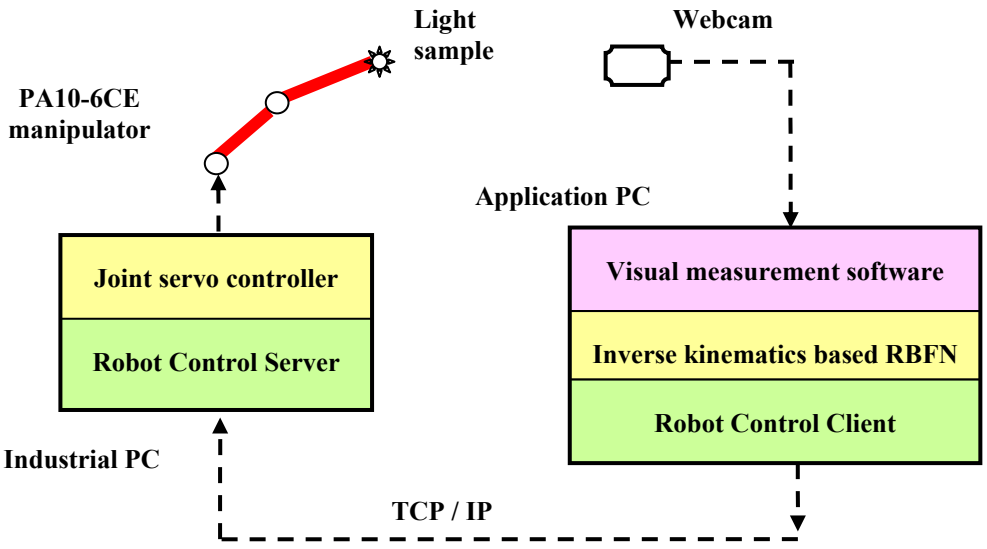
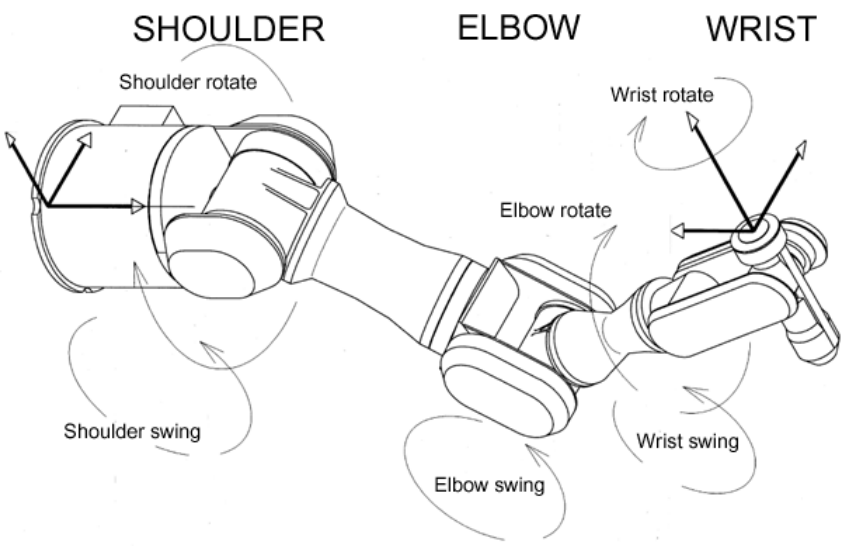


Figure 7.1 - General structure of the robotic control system.

7.2.1 Mitsubishi PA10-6CE manipulator

This PA10-6CE manipulator is a general-purpose robot manufactured by Mitsubishi Heavy Industries. Figure 7.2 shows a diagram of its range of movements and a photograph of the actual manipulator. The technical specifications and servo-control driver of the PA10-6CE [7.2] are presented in Appendix A.



(a) – Schematic of manipulator joints

(b) – Mitsubishi PA10 manipulator

Figure 7.2 – General purpose robot manipulator Mitsubishi PA10 – 6CE.

7.2.2 Robot control server

An open-architecture computer system for remote control of a robotic system has previously been developed [7.1], [7.4]. It includes a robot control server and a network communication module running on an industrial personal computer (IPC) under the QNX operating system environment. The application programming interface (API) and necessary drivers on the IPC have been already developed to interface with the PA10 manipulator. This provides a flexible tool for programming the control application algorithms.

The QNX operating system is ideal for real-time applications [7.5]. It provides all the essential ingredients of a real-time system, such as multitasking, priority-driven pre-emptive scheduling and fast context switching. It is also flexible in that the developers can easily customise the operating system to meet the needs of their applications. QNX achieves its high efficiency, modularity and simplicity through two fundamental principles: a micro-kernel architecture and message-based inter-process communication [7.5]. The robot control server programme (robot server) performs two programming tasks, control and communication, synchronously under two threads of the QNX. From a user's point of view, this robot server can be seen as a hidden layer between the user's application algorithm and the joint servo-controllers. Thus, a control task (e.g., joint position or velocity) is sent from the user's programme to the joint servo-controllers without any attention as to how it is transformed to a reference signal (e.g., voltage, current) for the controller. The robot server is also seen as a remote communication partner where the application programme communicates to the server through some standardised commands. For example, the application programme of this practical work communicates to the robot server via the internet interface to set new joint positions for the robot and to receive the status reply. It is implemented by a command using the format: "j *nn nn nn nn nn nn nn*" [7.1], where *nn* = 10 x joint angle [degrees].

This is an open-architecture control system, which means that the control layer (application programmes) can be replaced and easily manipulated. Figure 7.3 presents the block diagram of the open-architecture control system. This allows the users to develop and test any desired control algorithm in a high-level programming language without directly accessing the base control level.

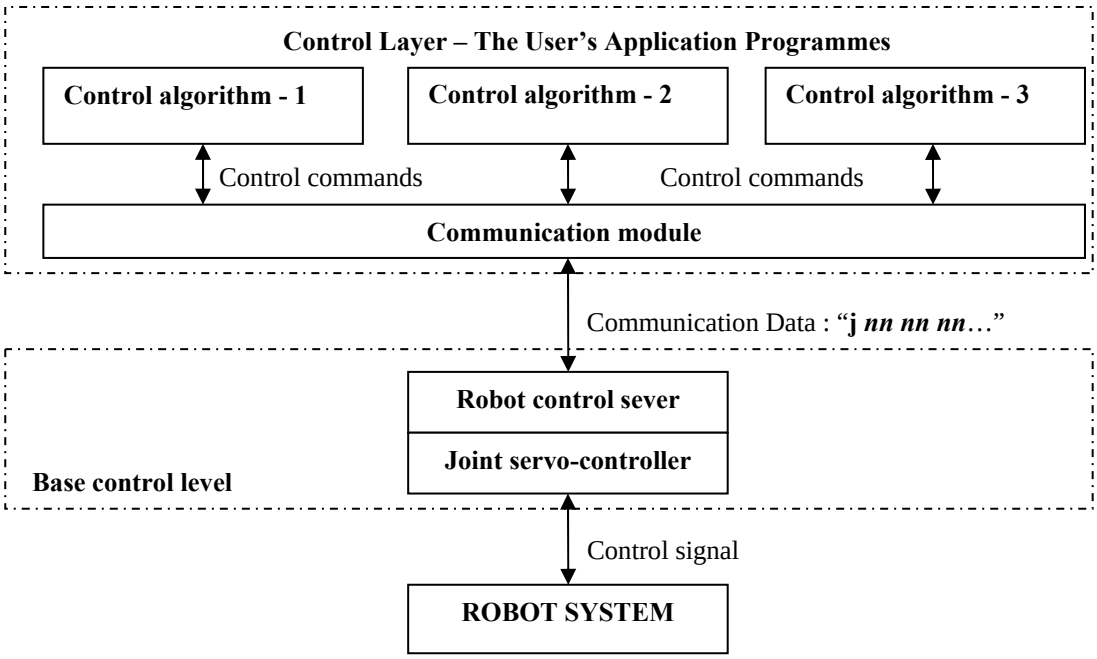


Figure 7.3 - Block diagram of the open-architecture control system.

7.2.3 Application programmes

Application programmes were developed in Visual C++ and run on the application computer to perform the inverse kinematics transformation of the PA10 manipulator. Two programmes were developed, one for the two-dimensional workspace and the other for the three-dimensional workspace. Figures 7.4 and 7.5 show the GUIs' developed for these experiments. There are three main software modules: communication, visual measurement and RBFN.

Communication module: This establishes a connection between the application programme and the robot server. If there are any control commands issued, it provides a communication pathway to pass the control data packet to the robot server via the internet interface.

Visual measurement module: This image processing software analyses image data from the webcam and then estimates the position of the PA10 manipulator in the workspace. There are two different visual measurement packages, one for the two-dimensional application and the other for the three-dimensional application. The first

one is based on an image-based scheme in which the position of the LED attached to the end-effector is directly determined by image coordinates (pixel units) in the image plane. The second uses a 3-D visual measurement tool (described in Chapter 6) to determine the Cartesian coordinate position of the end-effector with respect to the camera base. Both were developed using the OpenCV library [7.7].

RBFN module: This module implements both the training and operational phases. Two different software packages were developed to perform the inverse kinematics approximation of the PA10 in the two-dimensional and three-dimensional workspace.

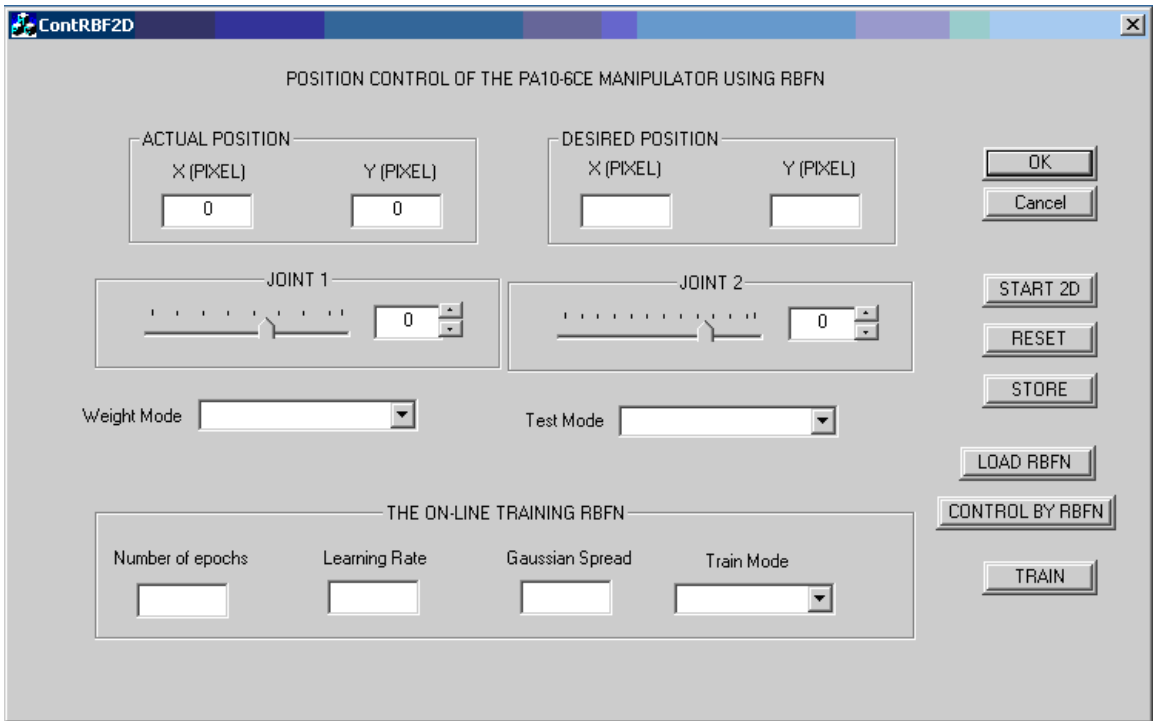


Figure 7.4 - GUI of the application software for 2-D workspace experiments.

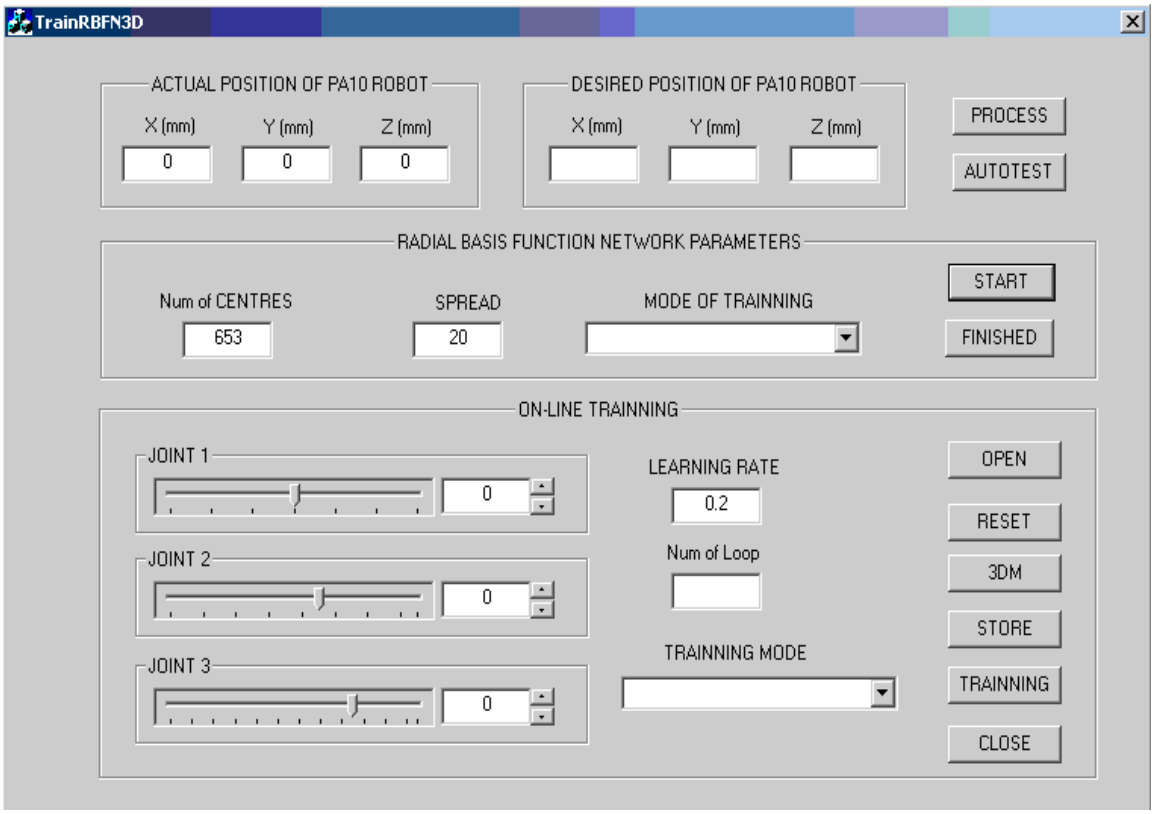


Figure 7.5 - GUI of the application software for 3-D workspace experiments.

7.3 Description of the robotic system for practical experiments

This section presents the set-up of the practical experiments using two different schemes for the two-dimensional (2-D) and three-dimensional (3-D) workspaces. For each case the structure of the manipulator (controlled by two or three joints), the RBFN and the visual measurement system are different. Thus, the application software for each experiment has been separately developed.

7.3.1 Structure of the robotic system for the 2-D experiments

This is implemented using an image-based control scheme in which the position of the LED attached to the end-effector is directly determined in the webcam plane as shown in Figure 7.6. The position of the LED is observed as the position of the end-effector.

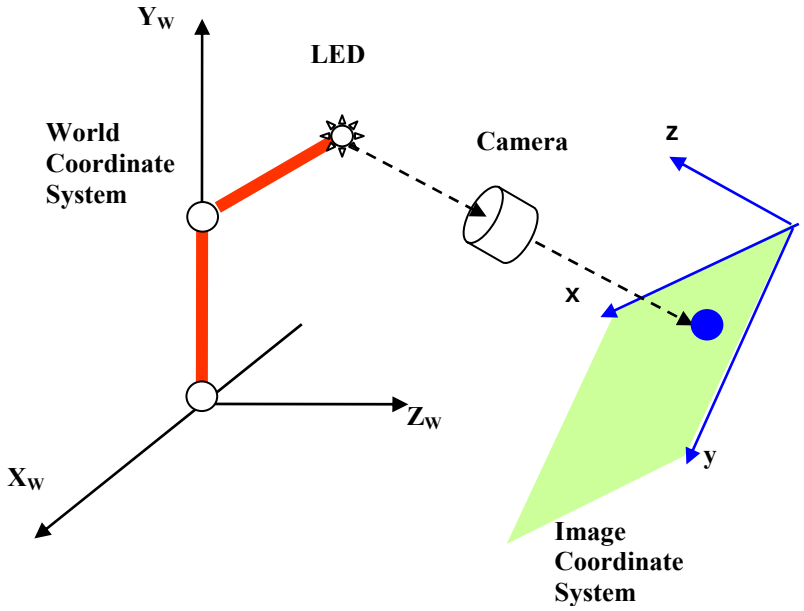


Figure 7.6 – Simple visual measurement system for the 2-D workspace.

Figure 7.7 shows the general structure of the robotic system for the 2-D experiment. The RBFN is used to approximate the inverse kinematics transformation of the PA10 manipulator from the image coordinate space to the joint angle space [7.8], [7.9]. This RBFN transforms a desired position (in the image plane) to the corresponding desired set of joint angles. The joint servo-controllers then use this set of joint angles as reference commands to move the PA10 in the workspace. Using this simple vision system, the position of the PA10 manipulator is represented by image coordinates in pixel units, instead of the world coordinates with respect to the base frame, so the geometry of the manipulator is not required.

The PA10 manipulator is controlled to move in two dimensions by only allowing movement of the shoulder-swing (S_2) and elbow-swing (E_1) joints (Figure 7.2 (a)). The technique of using an RBFN, consisting of two inputs (x, y) and two outputs (θ_1, θ_2), to approximate the inverse kinematics transformation of the two-link manipulator is illustrated in Figure 7.7.

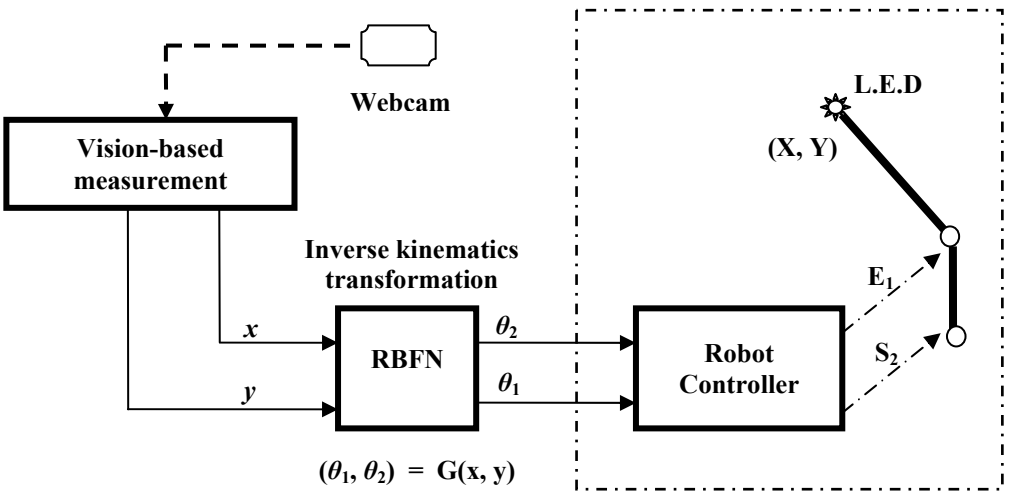


Figure 7.7 – General structure of the robotic system for 2-D experiments.

A simple visual measurement system based on a webcam and image processing software is used. The webcam captures the LED scene and sends video frames with a resolution of 640 x 480 [pixels] to the image processing software with a transmission speed of up to 30 frames per second. This system can perform measurements without any knowledge of camera calibration parameters and the vision geometry (i.e., distance from the camera to the manipulator). Therefore, this visual measurement system can directly determine the relative position of an unknown geometric robotic system in the image plane.

7.3.2 Structure of the robotic system for the 3-D experiments

The position of the PA10 manipulator is determined by the 3-D coordinates of the light sample board attached to the end-effector with respect to the camera frame as shown in Figure 7.8. This position, as described in Chapter 6, is the 3-D coordinates of the first point (reference) of the light sample board with respect to the webcam base. By using this 3-D visual measurement system, the workspace of the PA10 manipulator is represented in the camera coordinate system (X_c, Y_c, Z_c), instead of the world coordinate system with respect to the base frame. The coordinate system attached to the base can be determined from the camera coordinate frame by a homogenous transformation matrix. Although this is feasible, it makes the measurement system more complex and increases the system errors as well.

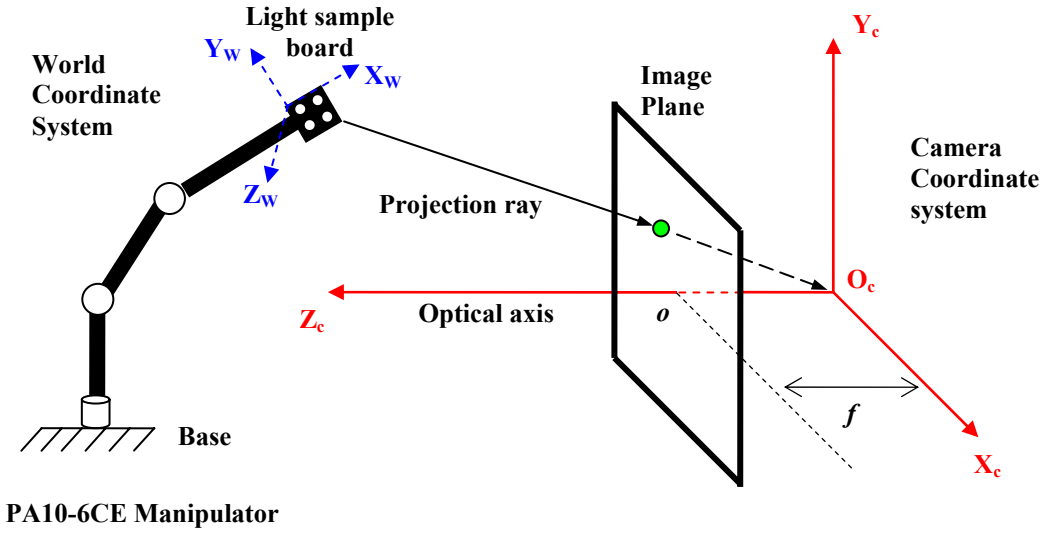


Figure 7.8 - Determination of the PA10 manipulator position in the 3-D workspace.

The visual measurement system consists of a webcam, a light sample board and image processing software (described in Chapter 6). This light sample board is a self-manufactured device in which light points are produced from four standard 5mm L.E.D's and powered by a 220VAC/6VDC adaptor. The image processing software estimates the position (3-D coordinates) of these light points with respect to the camera location according to the images of these points in the webcam plane.

Figure 7.9 shows the block diagram of the robotic system for the 3-D experiments. The RBFN, consisting of three inputs (X_c, Y_c, Z_c) and three outputs (θ_1, θ_2, θ_3), is used to transform a desired position in the 3-D workspace to the corresponding set of joint angles. The PA10 manipulator is controlled to move in the 3-D workspace by only allowing movement of the shoulder-rotate (S_1), shoulder-swing (S_2) and elbow-swing (E_1) joints (Figure 7.2 (a)).

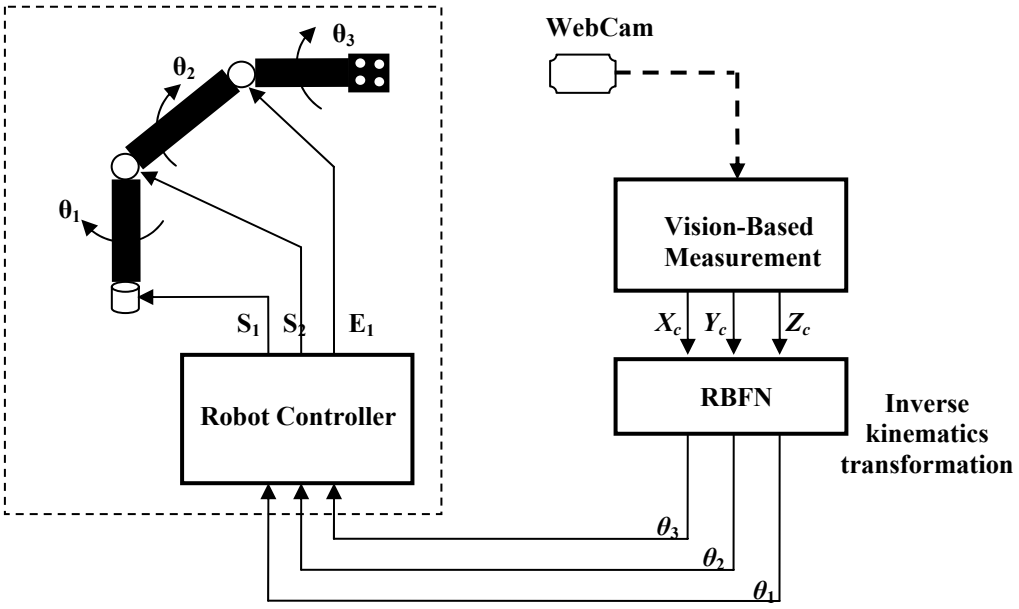


Figure 7.9 - General structure of the robotic system for 3-D experiments.

7.4 Practical determination of the inverse kinematics of the robotic system

7.4.1 Experimental description

The experiments using an RBFN to approximate the inverse kinematics were performed using two different structures of the robotic system for the 2-D and 3-D workspaces. These can be described as follows:

- Firstly, the hidden layer of the RBFN was built with a set of pre-defined centres regularly distributed in the workspace. Figures 7.10 and 7.11 present the distribution of 10 pixel x 10 pixel grids for the 2-D workspace and 20 mm x 20 mm x 20 mm cubes for the 3-D workspace. These centres had to be chosen in the operational region which is constrained due to joint angle limits. The number of hidden centres was 167 points (2-D) and 653 points (3-D) for the two experiments. The spread of Gaussian functions was experimentally selected as a proportion of the centre distance.

- Secondly, training data was manually collected as constrained patterns, which are as close to the centres as possible, by using the joint servo-controllers and the visual measurement system. For the 2-D experiment, the patterns were formed as $\{(x, y); (\theta_1, \theta_2)\}$ where the inputs are image coordinates in pixel units. For the 3-D experiment, the patterns were formed as $\{(X_c, Y_c, Z_c); (\theta_1, \theta_2, \theta_3)\}$ where the inputs are 3-D coordinates with respect to the webcam base. The quality (or accuracy) of collected data depends on careful observation and a poor pattern means that its input deviates from the pre-defined position (centres). This deviation was no higher than 30% of the centre distance. Collected data for the 3-D experiment involved unpredictable interference due to the low quality of the visual measurement system (the light sample board and webcam), the fluctuation of the power supply and the positional errors of the joint servo-controllers. Consequently, sometimes the measured position is not consistent, even with the same joint angles sent to the servo-controllers. This situation is more serious in the Z direction in which the variation was about 6-12 mm, instead of 2-6 mm in the other two directions when measuring the same point several times. This effect is not due to the performance of the inverse kinematics function because at this stage only the joint servo-controllers have been used. In order to reduce this variation, each training pattern should be sampled at least three times with the same set of joint angles and the input value of each training pattern is determined by the mean value.

- Thirdly, the linear weights were adjusted by either the strict interpolation or the least mean square (LMS) algorithms [7.6] with previously collected training data. At this stage, various spread values can be investigated to select a suitable RBFN structure that produces a good approximation of the inverse kinematics transformation.

- Finally, to verify the network performance a test data set was presented as a number of desired positions in the workspace. The robotic system, which uses the RBFN to perform the inverse kinematics transformation, moved to actual positions dependent on the response of the RBFN. The error between the desired and actual position was calculated to verify how well the RBFN approximates the actual inverse kinematics function. This practical error is affected not only by the quality of the RBFN but also measurement error and joint servo-controller error.

The test data set consists of 20 and 50 test points for the 2-D and 3-D experiments respectively.

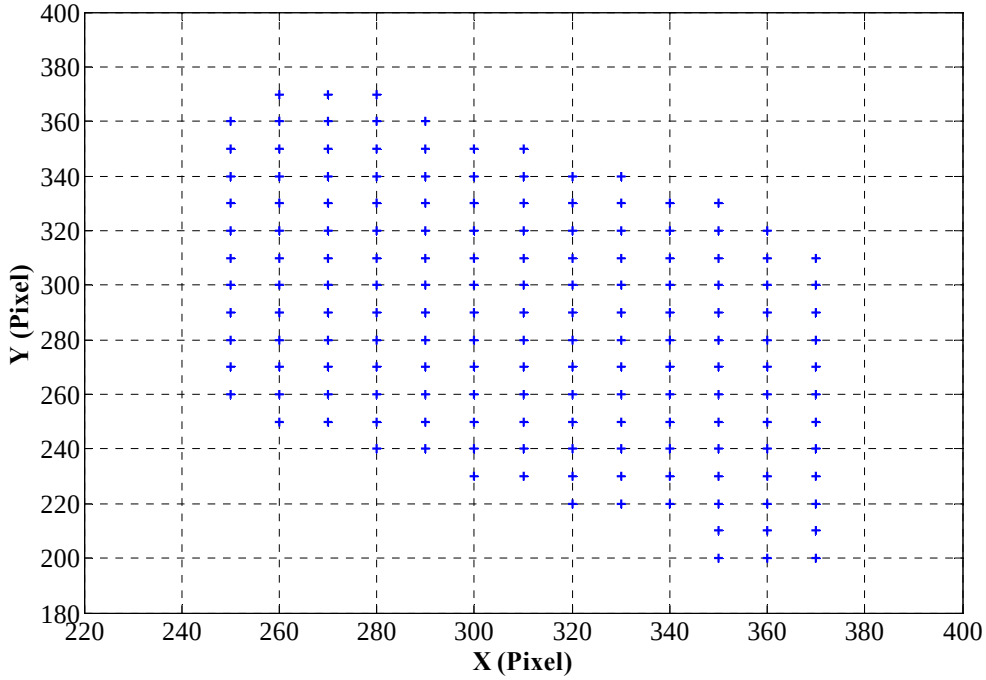


Figure 7.10 - Distribution of the hidden-layer centres in 2-D workspace.

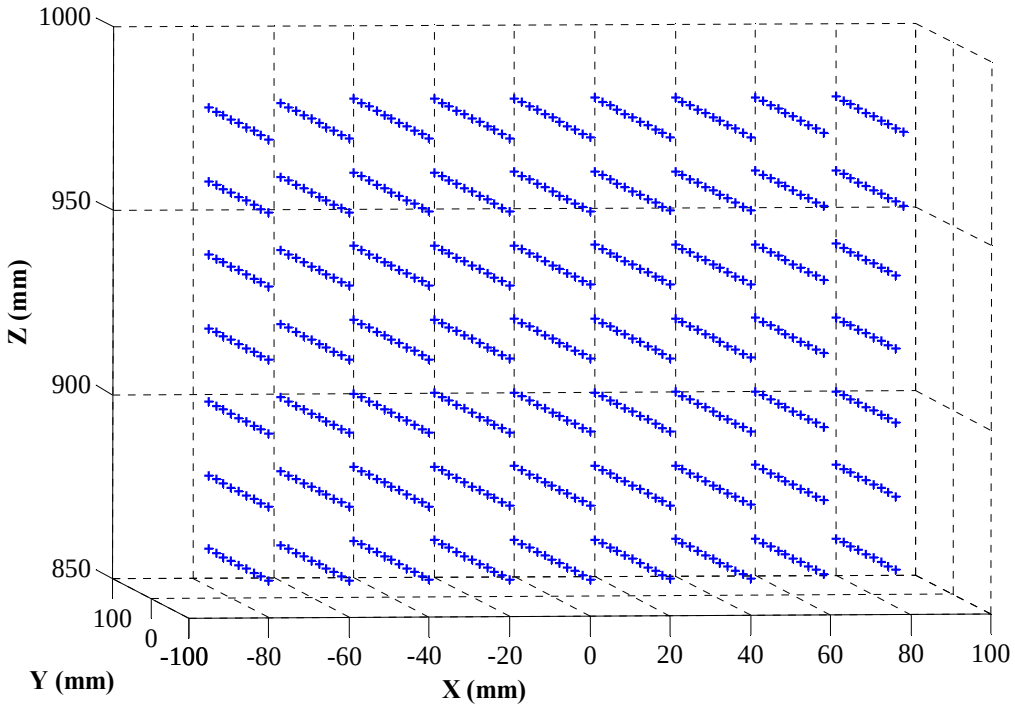


Figure 7.11 - Distribution of the hidden-layer centres in 3-D workspace.

7.4.2 Experimental results

Table 7.1 describes the RBFN performance in the 2-D workspace after training by the strict interpolation and LMS algorithms. The columns show the performance criteria (e.g., mean absolute error between desired and actual positions in *x* and *y* coordinates, MAE_X, MAE_Y) and the rows contain various spread values used for each training method.

TRAINING METHOD APPLIED		MAE_X (Pixel)	MAE_Y (Pixel)
STRICT INTERPOLATION	**Spread = 6 (Pixel)**	2.35	2.95
	Spread = 7 (Pixel)	0.9	1.55
	Spread = 8 (Pixel)	0.95	2.1
	Spread = 9 (Pixel)	0.95	2.2
	Spread = 10 (Pixel)	1	2
LEAST MEAN SQUARE	**Spread = 6 (Pixel)**	2.45	3.2
	Spread = 7 (Pixel)	1.25	1.6
	Spread = 8 (Pixel)	1.3	1.95
	Spread = 9 (Pixel)	1.5	2
	Spread = 10 (Pixel)	1.4	2.05

Table 7.1 - Performance results of the experiment in 2-D workspace.

This shows that the RBFN with a spread of 7 pixels produced the best performance after training by both methods.

Figures 7.12 and 7.13 present the RBFN performance with a spread of 7 pixels. Figure 7.12 presents the distribution of desired and actual positions in the workspace and Figure 7.13 shows the errors between the desired and actual positions using the strict interpolation and LMS methods to compare their effectiveness.

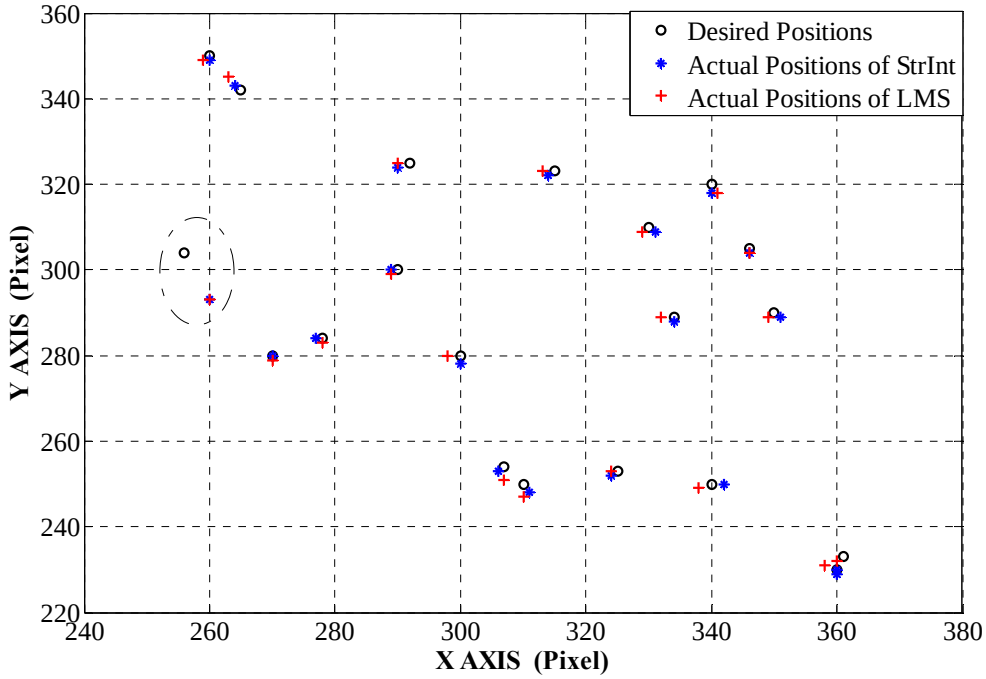


Figure 7.12 - RBFN performance (centre distance of 10 pixels, spread of 7 pixels).

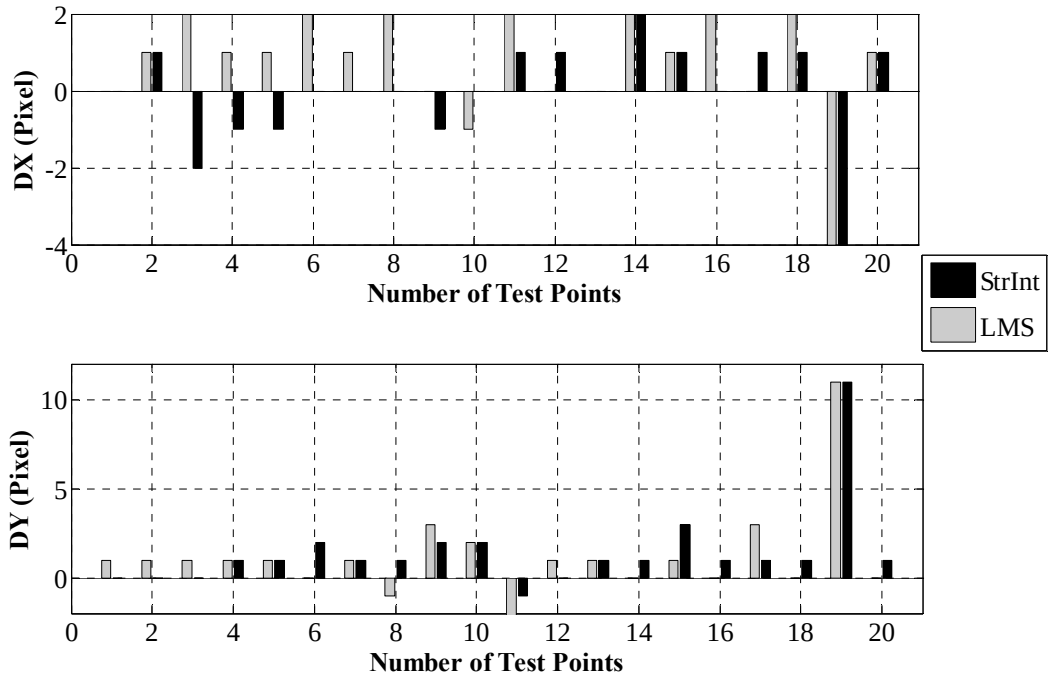


Figure 7.13 – Error between desired and actual positions.

Figure 7.13 shows that most of the actual positions were close to the desired positions with an error of 1 or 2 pixels (one pixel is approximately equivalent to 1.8 mm). The

RBFN trained by the strict interpolation method had a slightly better performance than when trained by the LMS algorithm. These results demonstrate that the RBFN can produce an appropriate approximation of the inverse kinematics transformation of the robotic system in the 2-D workspace. However, there was one position which deviated significantly from the actual desired position. This happened in all experiments with different spreads because this test point was located near the edge of the workspace where the RBFN may have insufficient basis functions to produce an adequate generalisation in that region. This situation is similar to the simulation results in Chapter 4 with the test points located near the edge of the workspace.

Compared to the simulation results presented in Section 4.3.2, the experimental results are poorer. For the simulation case, an error smaller than 3% of the centre distance was achieved (where the maximum deviation of training data was no higher 30% of the centre distance). If the centre distance is selected at 10 mm, then the error is smaller than 0.3 mm. In the practical work, the additional errors are because of the visual measurement system and the joint servo-controllers. The visual measurement error always exists as there is at least an error of 0.5 pixels due to the discrete form of image data.

Table 7.2 presents the 3-D experimental results. The columns show the mean absolute errors between desired and actual positions in X, Y and Z coordinates (MAE-X, MAE-Y, MAE-Z) and the rows contain the spread values. The 3-D visual measurement system measured the position of the PA10 manipulator with respect to the webcam reference. Thus, the coordinate values are in metric units (mm), instead of pixel units as used in the 2-D experiments.

As shown in Table 7.2, the RBFN with a spread of 24 mm produced the best performance after training by both methods. Figures 7.14 and 7.15 show the experimental results with a spread of 24 mm. Figure 7.14 presents the distribution of the desired and actual positions in the workspace and Figure 7.15 shows the errors between the desired and actual positions using the strict interpolation and LMS methods to compare their effectiveness.

METHOD APPLIED		MAE_X (mm)	MAE_Y (mm)	MAE_Z (mm)
STRICT INTERPOLATION METHOD	**Spread = 16 (mm)**	5.75	10.69	7.66
	Spread = 18 (mm)	3.54	6.19	7.32
	Spread = 20 (mm)	2.52	4.06	8.09
	Spread = 22 (mm)	2.33	3.00	7.63
	Spread = 24 (mm)	2.07	2.43	7.52
LMS	**Spread = 16 (mm)**	6.04	10.68	8
	Spread = 18 (mm)	3.69	6.55	8
	Spread = 20 (mm)	3.08	5.05	7.89
	Spread = 22 (mm)	2.12	3.52	7.51
	Spread = 24 (mm)	2.1	3.52	6.49

Table 7.2 - Performance results of the experiment in 3-D workspace.

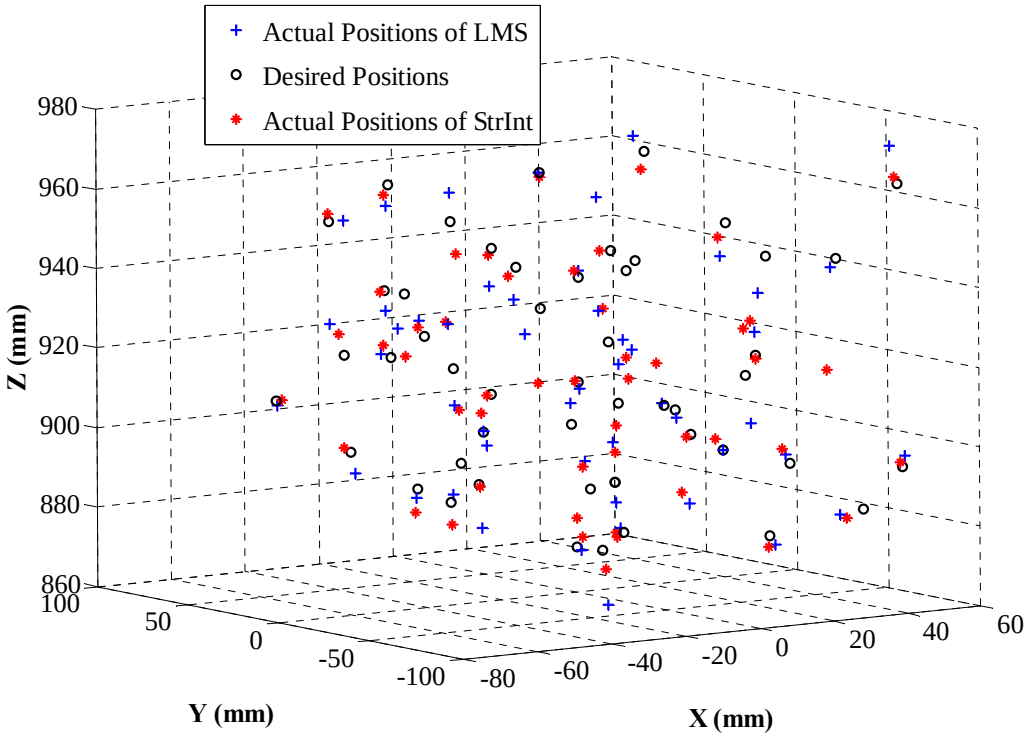


Figure 7.14 – RBFN performance (centre distance of 20 mm, spread of 24 mm).

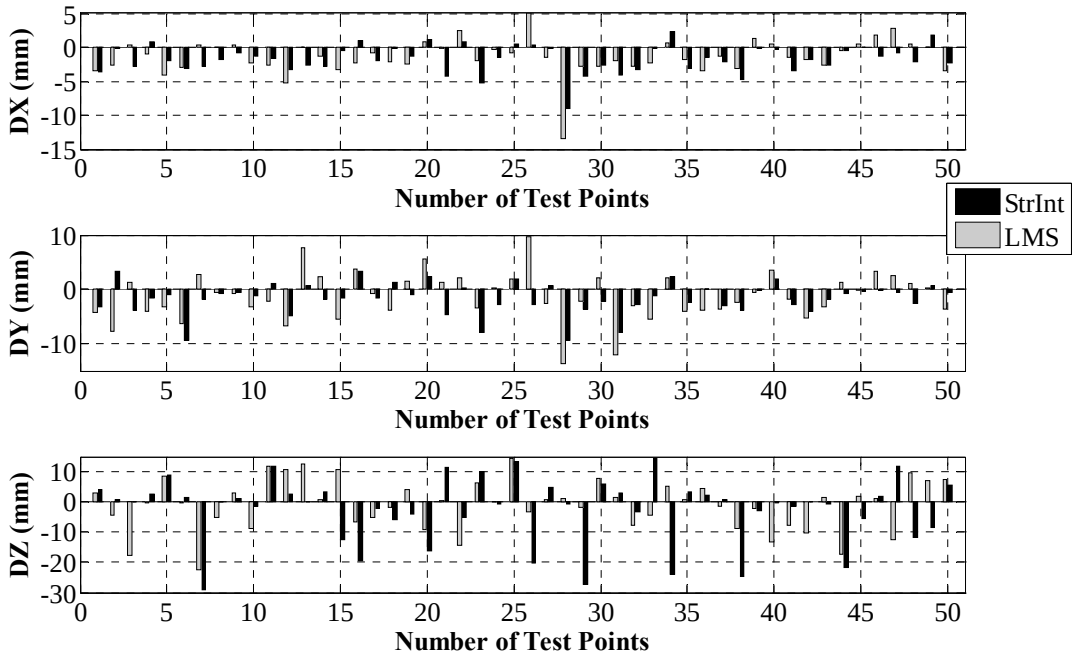


Figure 7.15 - Errors between desired and actual positions.

The experimental results demonstrate that the RBFN can produce an adequate inverse kinematics transformation of the robotic system in the 3-D workspace. Similar to the 2-D workspace case, the experimental results are poorer than the simulation results (Section 4.4.2). The practical performance is affected by not only the RBFN but also other components of the robotic system, such as the visual measurement system and the joint servo-controllers. When compared to the 2-D workspace experimental results with an image-based control scheme, the 3-D workspace experiments are affected more significantly by the operation of the visual measurement system. This is because the 3-D measurement involves a conversion error from the 3-D view to 2-D data adding further estimation errors. Additional errors are due to the low quality of the light sample board and the fluctuation of power supply. Thus, the 3-D visual measurement system errors are higher than the 2-D visual measurement system errors.

7.4.3 Summary of results

In general, using the pre-defined centres of the hidden layer as regularly-spaced positions means that the hidden layer structure remains the same and therefore is able to generalise throughout the whole workspace. As the inverse kinematics transformation of

this robotic system does not contain high frequency components, the strict interpolation method seems to be a suitable solution if the training patterns can be collected around the centres. The experimental results demonstrate that the proposed method can produce a well-generalised RBFN with a small number of hidden units. This is similar to the simulation case with random training data whose the maximum deviation from the centres is no higher than 30% of the centre distance.

It is observed that when the distance between the centres becomes smaller, better generalisation can be achieved. Therefore, the RBFN needs more hidden units to improve the accuracy of the inverse kinematics function. Obviously, there is a practical limit to the number of centres of the hidden layer due to lack of computer memory and the complex architecture of the network. In addition, most of the practical errors are due to additional errors (visual measurement system). The improvement of the RBFN alone does not lead to an improvement in the overall performance of the robotic system.

7.5 Practical work using online retraining to modify the RBFN

The approach described in Chapter 5 is used to modify online the RBFN to cope with a change in the robotic system structure. This was only implemented for the 2-D workspace with the image-based control scheme (Figure 7.7) as the 3-D workspace experiments were too problematic to investigate the online training progress due to the inaccuracy of the 3-D visual measurement system.

The experimental procedure is similar to the previous experiment with an additional online retraining step. The practical experiment is described as follows. The RBFN trained in the previous experiment was used to provide the inverse kinematics approximation of the robotic system. The best RBFN, which had been trained by the strict interpolation method with a spread of 7 pixels (Table 7.1), was used. A test data set, consisting of 12 test points, was used as a rectangular trajectory. The performance is presented in Figure 7.16.

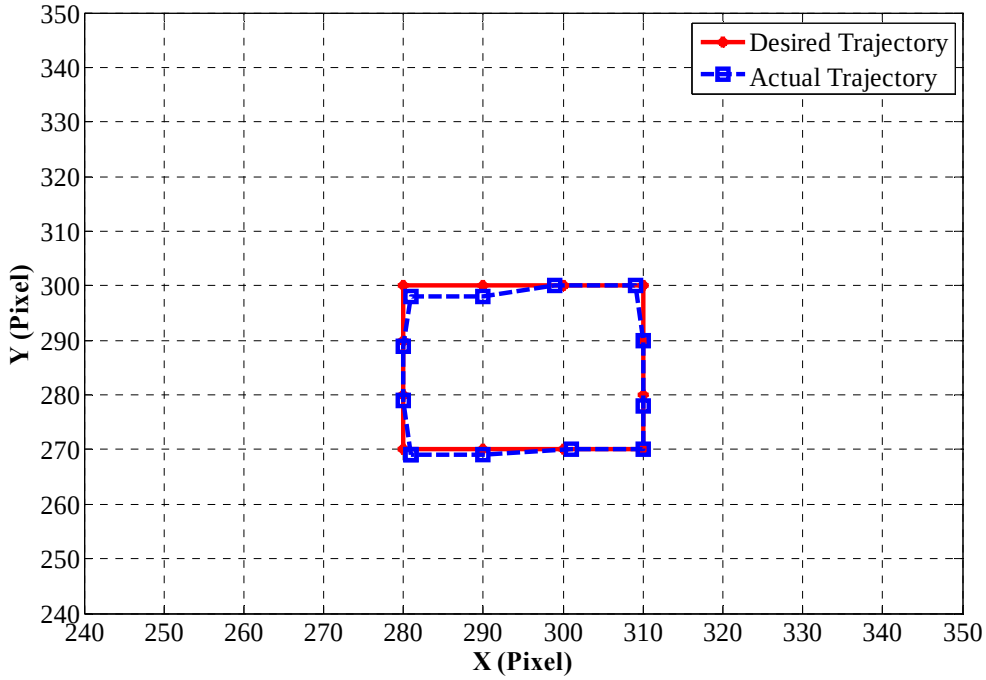


Figure 7.16 - Performance of the RBFN trained by the strict interpolation method.

To alter the set-up of the robotic system, the webcam was rotated through an arbitrary angle so that it changed the image transformation of the visual measurement system correspondingly. Consequently, the inverse kinematics approximation stored in the existing RBFN no longer matches with the new structure of the robot-vision system. Figure 7.17 presents the performance of the RBFN with the new condition of the visual measurement system and the errors introduced are obvious from inspection.

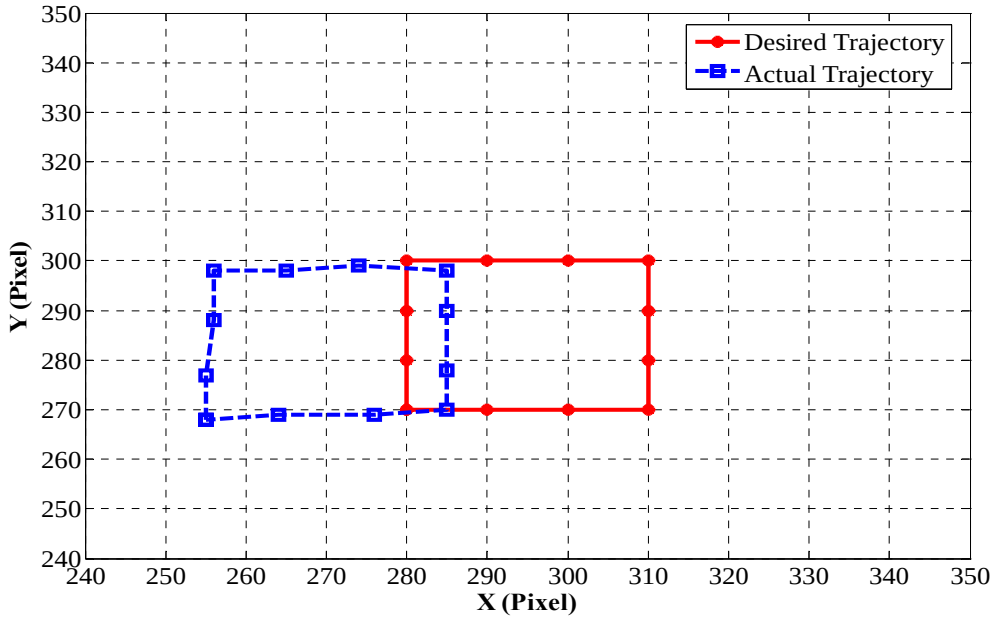


Figure 7.17 - Performance of the existing RBFN with a new condition of the visual measurement system.

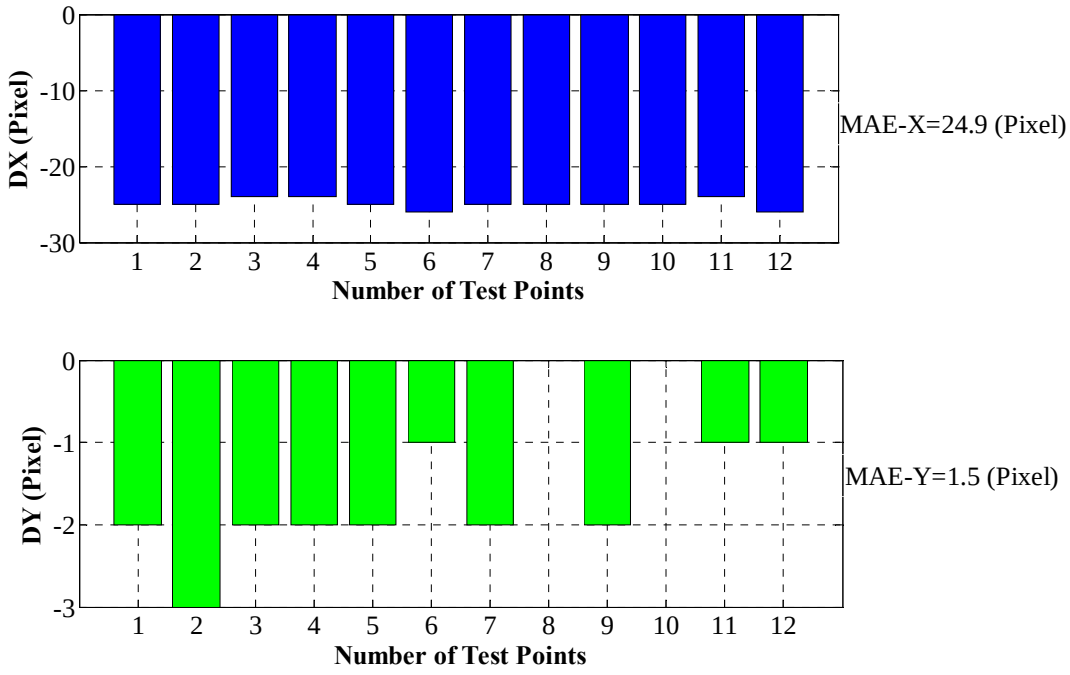


Figure 7.18 - Error between desired and actual positions with a new condition of the visual measurement system.

To obtain the new correct function, an online training process was applied using the delta rule (Chapter 5) instead of being retrained by the strict interpolation method from the start again. In this procedure, the linear weights are adjusted with each recent training pattern obtained from moving the PA10 manipulator in the workspace. However, as mentioned in Chapter 5, the improvement of the RBFN by the online learning process is dependent on the spread, the learning rate and training patterns. As suggested from the simulations in Chapter 5, this experiment was implemented by updating the linear weights with 10 training points that were collected according to the free interference rule. The learning rate was chosen as a small value (e.g., 0.1-0.2) to maintain the smoothness of the inverse kinematics approximation. A new retraining loop, which is then repeated, used the same existing 10 training points to continuously update the linear weights. Thus, after incremental updating with a sufficient number of training points, the RBFN can adapt to the new condition of the visual measurement system and the performance of the robotic system is clearly improved. Figures 7.19 and 7.20 shows the performance of the RBFN after training with 100 training patterns (10 training points applied to 10 retraining loops).

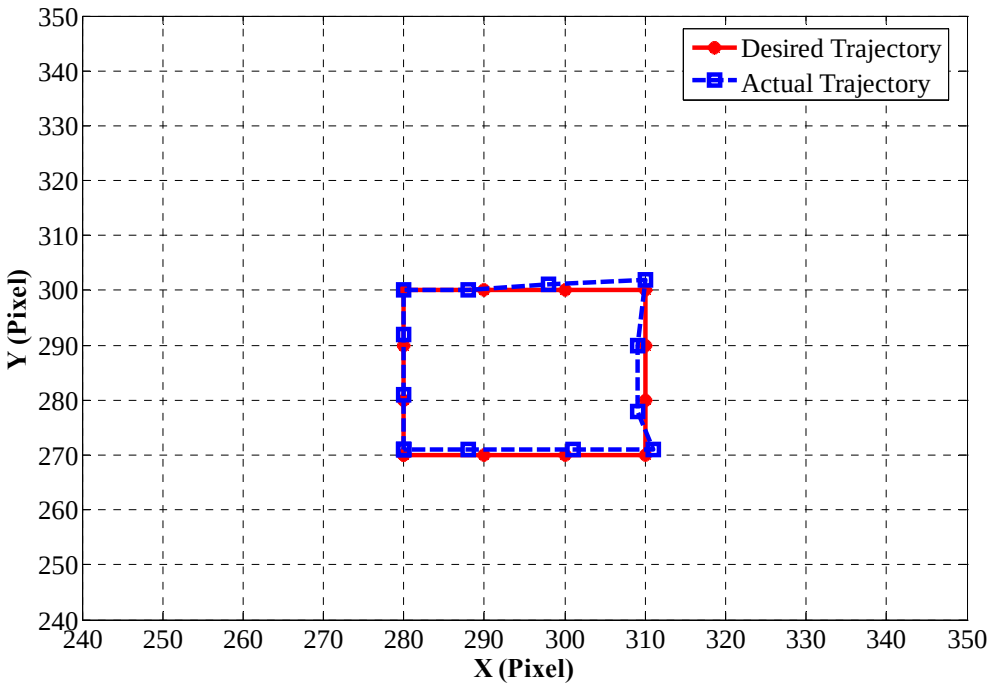


Figure 7.19 - Performance of the RBFN after online retraining with 100 training patterns.

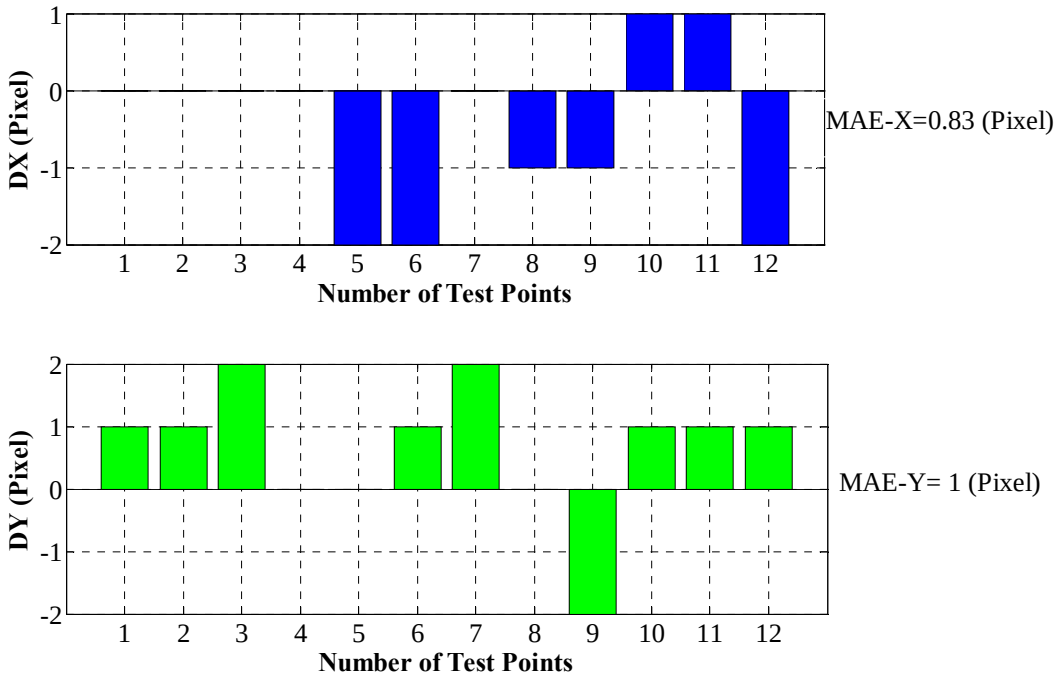


Figure 7.20 - Error between desired and actual positions after online retraining with 100 training patterns.

As mentioned before, the online learning process is affected by the position of the training patterns, the learning rate and the spread. Therefore, if different training parameters are applied, the improvement of the RBFN in the online retraining phase will vary and the overall performance of the robotic system will change as well.

Since the RBFN acts as a locally tuned function, only hidden units close enough to the training pattern positions contribute noticeably to the network output. As a result, only the linear weights connected to these hidden units are adjusted via online training. It means that the positions of the training data have an important impact in the online training process. The closer the training pattern to the test points, the stronger the effect in modifying the linear weights of the RBFN in that area. Different patterns presented to the RBFN can produce different improvement effects in the approximation function. Thus, the distribution of training patterns should cover the entire workspace to modify the whole of the inverse kinematics function.

The choice of learning rate is an important variable in the delta rule algorithm. The variation in learning rate can lead to completely different training results. Therefore, in

order to maintain the smoothness of the inverse kinematics approximation over the whole workspace, a small learning rate (η = 0.1- 0.2) should be adopted even though it made the training process slower. This is the reason that the retraining process required a hundred training points for the small test area. The actual number of training patterns for these experiments was collected at a maximum of 10 points and they were used repeatedly in the incremental mode with a small learning rate to ensure the stability of the learning process. In other experiments, when using the same training patterns with a larger learning rate (e.g., 0.3 to 0.5), the RBFN performs well in the test area after training with only two retraining loops (e.g., 20 training points). However, its response in other neighbouring areas appeared to be worse due to the learning interference.

7.6 Conclusion

This chapter has presented the practical results produced from an investigation of the effectiveness of using an RBFN to approximate the inverse kinematics. The experiments were performed for two different situations. In the first case, an RBFN was trained to approximate the inverse kinematics transformation of the robotic system, including the PA10 manipulator and the visual measurement system, through an offline training phase using the strict interpolation and the LMS methods. The second experiment investigated a variation in the structure of the robotic system and proposed a solution using an additional online retraining phase based on the delta rule. Experimental results verified the effectiveness of the proposed approach and practical limitations were discussed.

7.7 References

[7.1] C. M. Wronka and M. W. Dunnigan, “Internet Remote Control Interface for a Multipurpose Robotic Arm”, the *Int. Journal of Advanced Robotic Systems*, vol. 3, June 2006, pp. 179-182.

[7.2] *General Purpose Robot PA10-6CE – Instruction Manual for Installation, Maintenance and Safety*, Mitsubishi Heavy Industries, Ltd.

[7.3] *General Purpose Robot PA10-6CE – Operation Manual for Operation Support Program*, Mitsubishi Heavy Industries, Ltd.

[7.4] A. Uhlmann, *Development of Driver Software to Support The Mitsubishi PA10 Manipulator under The QNX Neutrino Real-Time Operating System*. Master thesis, Heriot-Watt University, 2003.

[7.5] *The Philosophy of QNX*. QNX Software Systems Ltd. [Online]. Available: http://www.qnx.com/developers/docs/qnx_4.25_docs/qnx4/sysarch/ intro.html. 2006.

[7.6] S. Haykin, *Neural networks - A Comprehensive Foundation.* Prentice Hall, Inc., 1999.

[7.7] *Open Source Computer Vision Library - Reference Manual, Version 04,* Intel Corporation, Dec. 2001.

[7.8] B. H. Dinh, M. W. Dunnigan and D. S. Reay, "Position Control of a Robotic Manipulator Using a Radial Basis Function Network and a Simple Vision System", in *Proc. of the IEEE Int. Symposium on Industrial Electronics (ISIE'08),* Cambridge, United Kingdom, July 2008, pp.1371-1376.

[7.9] B. H. Dinh, M. W. Dunnigan, and D. S. Reay, "A Practical Approach for Position Control Of a Robotic Manipulator Using a RBFN and a Simple Vision System", *WSEAS Transactions on System and Control*, vol. 3 (4), April 2008, pp. 289-298.

CHAPTER 8
CONCLUSIONS

8.1 General conclusions

In this thesis, several solutions to determine the inverse kinematics of robotic manipulators have been presented. Two analytical approaches, the geometric and algebraic, are traditional algorithms used to solve the inverse kinematics problem. Several alternative approaches using two of the most popular neural networks, MLPNs and RBFNs, were reviewed with regards to their abilities to approximate the inverse kinematics of unknown geometry manipulators. However, it is appropriate to propose some modifications to improve the performance of these existing approaches, especially in terms of practical applications.

A new approach using an RBFN with regularly-spaced position centres has been proposed to solve the inverse kinematics problem. Constrained data whose inputs are collected approximately around the centre positions in the workspace was also suggested. Two training methods, strict interpolation and the least mean square algorithms were introduced to update the network weights in an offline training phase. The effect of centre distances, spreads and training methods on the network's performance was investigated through various simulations. The effect of training data randomly collected around the centre points was also examined. A suggestion for the maximum deviation has been mentioned to improve the network's performance. The simulation results showed that using these proposed ideas improved the RBFN performance significantly. Moreover, an RBFN has a local mapping characteristic in which the RBFN will only respond to any inputs that fall in the trained area of the workspace. If a new input which is beyond the trained area is presented to the RBFN, it will respond by a "do nothing" action or by resetting all joint angles (zeros). Therefore, according to this property, the inverse kinematics approximation could avoid the violation of the mechanical limitations of joint angles, but its operation is also limited to a specific area dependent on the training data. This is useful for practical work when all training data are actually collected in the range of the joint angles. The operational phase is thus restricted to the trained region.

An online retraining approach was proposed to deal with the incorrect operation of the network because the initial network training occurred in an environment that is not exactly the same as the environment where the system is actually deployed. This online retraining approach can be effectively applied for systems whose characteristics change due to environmental variations. Various simulations to investigate the effects of the spread, learning rate and presented training patterns on the network's performance were conducted for a two-link and a three-link manipulator. The results demonstrated the effectiveness of the approach.

A real-time visual measurement system based on a video camera was developed to estimate the position of a robotic manipulator in a 3-D workspace. It consists of a camera, a light sample board and image processing software. This system is portable, low cost and has reasonable accuracy.

Practical experiments were performed with a Mitsubishi PA10-6CE manipulator and the visual measurement system. The performance of the inverse kinematics transformation using an RBFN was examined for operation of the manipulator in two- and three-dimensional spaces. The practical results were compared to the simulation results. Advantages and disadvantages of the proposed approach were discussed. The results demonstrated the effectiveness of the proposed approach for practical applications.

8.2 Author's contributions

Chapter 4 proposed a new approach using an RBFN with regularly-spaced position centres to solve the inverse kinematics problem. This requires a sufficiently small number of centres and can achieve a satisfactory accuracy for the inverse kinematics approximation through the whole workspace. The concept of using constrained data that are collected close to the centre positions enhances the generalisation of the RBFN. The maximum deviation for training data that was randomly collected around the centre positions was suggested based on experimental evidence to ensure a good performance is achieved in the operational phase. Simulation results verified this approach.

Chapter 5 proposed a new approach using the delta rule to update the linear weights through an online retraining phase. It was effectively applied to modify an incorrect

network due to operational environment variations, instead of retraining the network from the start again. A simple rule was suggested to select appropriate training points so that the effect of learning interference was minimised. The three factors (learning rate, spread value and the position of training points) that can affect the online retraining phase were investigated though various simulations. The key recommendations were presented to ensure a successful online retraining phase.

Chapter 6 presented a novel real-time 3-D visual measurement system. A light sample board consisting of four LED points with two different colours was a simple and economic solution. Image processing software was developed to be convenient for the users. It included an efficient solution to automatically extract the appropriate image data based on a real-time angle comparison algorithm. This visual measurement system is portable, low cost and has reasonable accuracy for use in a practical robotic system.

The most important contribution of this thesis is that it demonstrates that a neural network solution can be effectively applied to approximate the inverse kinematics of a practical robotic system. In Chapter 7, two different experimental schemes were presented. The first used an image-based control scheme where the image coordinates in pixel units represented the manipulator position in the 2-D workspace. The second scheme used the visual measurement system developed in Chapter 6 to determine the position of the manipulator in the 3-D workspace. Experimental results verified the effectiveness of the proposed solution to deal with common practical situations. This approach is a promising solution for high performance control applications using remotely controlled robots (e.g., underwater intervention or space exploration).

8.3 Suggestions for future work

Although the proposed approach has been verified for a two- and three-link manipulator, it should be investigated further for some more complex configurations such as greater than three D.O.F manipulators. This can be feasibly implemented for simulation cases in which the mathematical expression of the inverse kinematics is used to create training data. However, it is more difficult for practical work because a new measurement system will be required to estimate the orientation of the end-effector with respect to a fixed coordinate frame.

The solution using regularly-spaced position centres for a radial basis function still has performance limitations in operational areas near the edge of the workspace due to the lack of necessary radial basis function nodes in these areas. Thus, the network's performance will be improved if additional nodes can be added in these areas. This could be implemented in an incremental mode, which does not require the inverse of the interpolation matrix each time, by using a technique known as the inverse of a partitioned matrix or an orthogonal least squares learning approach.

In the online retraining phase (Chapter 5), the distribution of training patterns and the learning rate both influence the incremental modification of the linear weights. To keep the retraining process smooth and fast, the learning rate should not be a constant and should be related to the distribution of training patterns. For example, if a new pattern presented to the network is far away from previously used patterns, the learning rate can be large. In contrast, it should be small to keep the training function surface smooth when the current pattern presented is close, or the same, as previous patterns. This aspect requires further investigation.

As presented in Chapter 4, a set of random training data used to train an RBFN where the maximum deviation is set to a specific value (30% and 20% of the centre distance for a 2-link and 3-link manipulator respectively) produces a good approximation of the inverse kinematics transformation. Therefore, a new way of collecting data should be investigated. The training data could be sampled following a distribution in the joint angle space instead of a regularly-spaced distribution in the workspace. The set of collected data can then be sorted according to the maximum deviation from the predefined centre points. This approach is a more convenient technique than the current way of collecting data in the practical experiments.