Napier University

Edinburgh

Department of Mechanical, Manufacturing and Software Engineering

Robot Calibration Using Artificial Neural Networks

by

Xiaolin Zhong

A Thesis Submitted in Partial Fulfilment of the Requirements of Napier University,
Falculty of Engineering for the Degree of
Doctor of Philosophy

November 1995

Declaration

I hereby declare that the work presented in this thesis was carried out by myself at Napier University, Edinburgh except where due acnowledgement is made, and has not been submmited for any other degree.

Xiaolin Zhong

(Candidate)

Date

Contents

List of Figures and Tables

Glossary of Abbreviations

ANN (or ANNs, or NN)	Artificial Neural Netwoks
BP	Back-Propagation
CCD	Charge-Coupled Device
CIM	Computer-Integrated Manufacturing
CNC	Computerized Numerical Control
CMAC	Cerebellar Model Articulation Controller
CMM	Coordinate Measuring Machine
CPC	Complete and Parametrically Continuous
D-H	Denavit-Hartenburg
DoF	Degree of Freedom
FMS	Flexible Manufacturing System
GA	Genetic Algorithm
LMS	Least Mean Square
LVDT	Linear-Variable Differential Transformer
MLP	Multi-Layered Perceptron
N-R	Newton-Raphson
ODE	Ordinary Differential Equations
PUMA	Programmable Universal Machine for Assembly
RMS	Root Mean Square

RNN	Recurrent Neural Networks
SVD	Singular Value Decomposition
TCP	Tool Centre Point
VLSI	Very Large Scale Integrated-circuit
VSC	Variable Structure Control

Acknowledgements

It was a meeting with a Napier delegation in Shenzhen (a booming city in the corridor between Hong Kong and Canton, Southern China), especially a conversation with Prof. James Murray, Vice-Principal of Napier University, that made me seriously consider the idea to pursue a PhD abroad. However, without the financial support from Napier University Scholarship, and CVCP (Committee of Vice-Chancellor and Principals of Universities of the United Kingdom) ORS (Overseas Research Student) award, the sparkled idea would have never come to fruition.

I would have not survived my first day in Edinburgh without help from Simon Liang, a PhD student from Taiwan, who was kind enough to drive to the airport to meet me in the early morning through the heaviest snow I had ever seen. Many thanks, Simon, for your friendliness and hospitality, as well as for many delightful discussions about Chinese culture and politics --- the past and the future.

I am grateful to Dr John Hallam of Dept. of Artificial Intelligence, The University of Edinburgh for his instruction during the early stage of my research, which was invaluable in saving me from struggling in numerous murky ideas.

I would also like to acknowledge gratefully many other people for help and assistance during the course of my study. Among them are, Bill Campbell, for his assistance in robot programming; Heather Rea, for allowing me to use some of her data, and useful discussions; Ronnie Cohen, for his help with the use of Coordinate Measuring Machine; Dr Bob Stafford, for his tips on grammar and presentations; Bill Young, for being there when utilities were needed; and to all of my officemates, for tea-drinking time and entertainment.

I am indebted to my supervisors, Dr John Lewis, Dr Mike Mannion, Prof. Francis N-Nagy, Dr Barry Keepence, and Dr Duncan Marsh, for their inspiration, enlightenment and encouragement.

Last but not the least, I would like to thank my wife Du Ling, and my parents, for their love and consistent support.

Abstract

Robot calibration is an integrated procedure of measurement and data processing to improve and maintain robot positioning accuracy. Existing robot calibration techniques require extensive human intervention and off-line processing, which preclude the techniques from being used to perform on-site calibration in an industrial environment at regular intervals. This thesis investigates and develops intelligent calibration processing algorithms and a novel measurement method toward rapid autonomous robot calibration in a shop-floor environment.

Artificial Neural Network (ANN) techniques have been vigorously investigated for calibration data processing (modelling, identification and compensation). A new identification algorithm has been developed for estimating robot kinematic parameter errors using Hopfield continuous-valued type Recurrent Neural Network (RNN). The RNN-based algorithm is computationally more efficient and robust compared with conventional optimisation approaches.

A generic accuracy model which accounts for various error sources was introduced. A higher-order neural network was used for implementation of the generic accuracy model. Due to the ANN learning capability, computational power and adaptability, the ANN-based accuracy representation offers an appealing solution to the complex modelling problem.

Efficient and robust accuracy compensation algorithms have been developed under the framework of artificial neural networks. The ANN-based algorithms provide constant-time inverse compensation therefore are suitable for on-line implementation. Both path compensation and compensation near robot singularity were tackled using the new algorithm.

A novel autonomous calibration tool was developed using a trigger probe and a constraint plane. The new method eliminates any use of external measuring devices to determine robot end-effector location measurements, enabling the robot to perform self-calibration on a production line. Robot accuracy was improved to the level of its repeatability within the local calibration volume using the new calibration scheme, which is consistent with the results from using a precision external measuring device, in this case a Coordinate Measuring Machine (CMM).

NOMENCLATURE

$\Delta\mathbf{x} = [dx, dy, dz, \delta x, \delta y, \delta z]^T$	End-effector inaccuracy vector.
$\mathbf{p}_i = [x_i, y_i, z_i]^T$	The ith end-effector position vector.
Δ, δ	Position, and oriencation inaccuracy.
$\mathbf{T}_n$	End-effector homogenous transformation.
$\mathbf{A}_i$	The i-th link homogenous transformation.
$[\mathbf{n}, \mathbf{s}, \mathbf{o}]$	End-effector orientation vectors.
$[\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i]$	The i-th coordinate system
$\mathbf{a} = [a_1, a_2, ..., a_n]$	Link offset vector,n is the number of DoF.
$\mathbf{d} = [d_1, d_2, ..., d_n]$	Link length vector.
$\alpha = [\alpha_1, \alpha_2, ..., \alpha_n]$	Link twist angle vector.
$\theta = [\theta_1, \theta_2, ..., \theta_n]$	Joint angle vector.
$\rho = [\mathbf{a}, \mathbf{d}, \alpha, \theta]$	Kinematic (geometric) parameter vector.
$\Delta\rho$	Kinematic error vector.
$\mathbf{J} = [\frac{\partial \mathbf{f}}{\partial \mathbf{a}}, \frac{\partial \mathbf{f}}{\partial \mathbf{d}}, \frac{\partial \mathbf{f}}{\partial \alpha}, \frac{\partial \mathbf{f}}{\partial \theta}]$	Special Jacobian matrix.
$\mathbf{J}_\theta = \frac{\partial \mathbf{f}}{\partial \theta}$	Ordinary Jacobian matrix.
$d\mathbf{p}, d\mathbf{w}$	Position, and orientation error vector.
E	Network energy function.
T_{ij}, I_j	Network connection weight, and input.
$\mu = [\mu_i]$	learning rate.
$\mathbf{Q}$	Scaling weight matrix.
$\mathbf{e}$	Residual error vector.
α	Regulation coefficient constant.
$\mathbf{q} = [q_i]$	Generalised joint variables.
δq_i	The i-th joint correction.
$\mathbf{f}(\mathbf{k}, \mathbf{q}_i)$	Forward kinematic function.
$\Delta w_l, \Delta b_l$	The l-th weight, and bias correction.
η	Network learning parameter.
$\Lambda = \text{diag}(\lambda_i)$	Diagonal regulation weight matrix.
$\mathbf{H}$	Coefficient matrix of linear system.
$\Delta\mathbf{X}$	Aggregated inaccuracy vector.
L	Side length of the measuring cube.

CHAPTER 1

INTRODUCTION

1.1 Research Background and Motivation

The role played by industrial robots in modern factories is not as widespread as was predicted a decade ago. There are many reasons why robot systems have not met early expectations. One of the reasons is that most robot systems fail to deliver the promised flexible manufacturing environment. Industrial robots, as they currently exist, must be taught even the most basic tasks required of them for operation in a complex and highly diverse human world. Programming or teaching a robot to perform a desired task is exceedingly labour intensive. Any minor change in the task or uncertainties which exist in the robot or the environmental set-up will invalidate, partially or completely, the developed programs. Off-line programming which supports the development of a robot program in a simulated environment has resulted from research seeking better robot programming methods. Robot positioning accuracy, however, is critical for off-line generated programs to be implemented successfully on a shop floor.

One major goal of robotics research is to instil some human-like intelligence in robots, enabling them to function autonomously in an unstructured environment. An intelligent robot must have the ability to adapt its behaviour quickly and effectively to unpredictable environmental changes without human intervention. To position its end-effector as precisely as desired in a changing environment is one of the fundamental capabilities for a robot to achieve high level intelligence or autonomy. To address robot motion control accuracy in particular and robot programming flexibility in general, a

great deal of research endeavour has been made in various areas ranging from automated task planning, fine-motion planning, error detection and recovery, to learning control and adaptive control, focusing on robot controller's performance enhancement. While many encouraging results in laboratory environments have been reported in these areas, it is generally believed that it will still be years before there is a major impact on the robot systems currently used in most manufacturing applications. Robot calibration is an approach that can improve robot positioning accuracy significantly without increasing the robot controller's complexity, and can be easily integrated with existing systems. As robot positioning accuracy is a highly complex problem dealing with various error sources which exist in changing environments, an intelligent calibration scheme is needed to make error sources robust with minimum human intervention.

An artificial neural network (ANN) is a new technique in the field of artificial intelligence which imitates the functional processes of the human brain. By taking advantage of the structure of the human brain, which features massive parallelism and high interconnectivity, it is hoped that extremely complex problems can be solved by neural networks. Due to their computational power, learning methodology, adaptability and fault tolerance, neural networks offer appealing solutions to intelligent robotic problems. In this thesis, artificial neural network techniques are applied to the robot calibration processes, aiming at autonomous robot calibration in a shop floor environment.

1.2 Robot Positioning and Calibration Problem

One of the basic capabilities of industrial robots is to position their end-effectors precisely so that they can perform manipulation tasks successfully. Precise positioning control is difficult for multi-joint articulated manipulators since they are open-chain mechanisms with coupling between motions of each individual link. The desired locations (or pose: position and orientation) of a robot end-effector are normally specified in Cartesian space (workspace), while these locations are achieved by controlling joint values (angles for revolute-joint robots) in robot joint space (Figure 1.1). The transformation from Cartesian space to joint space is called the *inverse kinematics* in robotics. The inverse kinematics problem is typically a computationally intensive procedure for general-form multi-link manipulators, the accurate solution of

which depends upon the algorithms used and upon precise knowledge of robot parameters.

In practice, most industrial robots are designed to have simple-form kinematics (Hayati and Roston, 1986) so that the inverse kinematics problems have analytic close-form solutions. For instance, the widely-used PUMA robot is designed so that the second and third joint axes are parallel and the last three joint axes intersected orthogonal in a point. The inverse kinematics of the PUMA robot is analytically solvable due to its special kinematic structure (Fu, Gonzalez and Lee, 1987; Craig, 1986); However, for a number of reasons (including manufacturing tolerance, repair and set-up errors, wear and tear, transmission errors, and compliance), the internal design model used in the robot controller will not accurately describe the actual relationship between robot workspace and joint space. Therefore, the actual locations achieved by controlling joint values, obtained from the controller's internal model, will deviate from the desired locations. Robot calibration is defined as a process of improving robot positioning accuracy through modifying robot control software without changing the robot's hardware configuration (Mooring, Roth and Driels, 1991). The modification of control software can be performed either in robot workspace (*forward calibration*), or in joint space by finding the corrected joint values to drive the robot so that the end-effector deviations can be minimised (*inverse calibration*) (Shamma and Whitney, 1987).

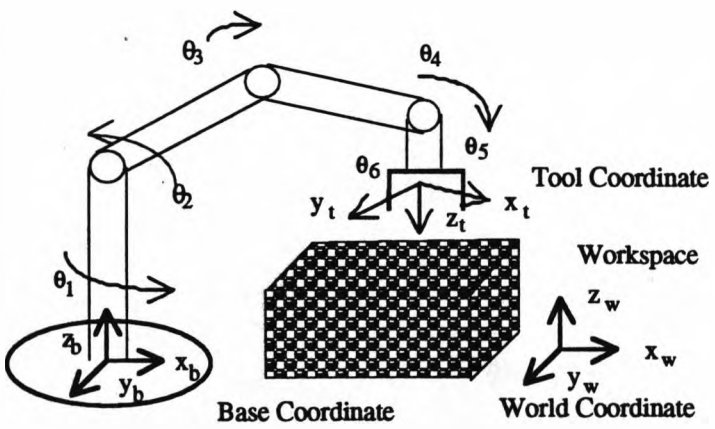


Figure 1.1 Positioning Control of a Robot Manipulator

Several performance specifications are used to describe positioning characteristics of robot manipulators, which include *accuracy*, *repeatability*, and *resolution* (Kozakiewicz, Ogiso and Miyake, 1990). The *resolution* is the smallest motion (linear and angular) the robot's end-effector can reliably execute. The *repeatability* is a measure of the robot's ability to return its end-effector to exactly the same point in workspace many times in succession. The (absolute) *accuracy* is a quantitative parameter describing the robot's ability to position its end-effector exactly in the World coordinate system. When a six degree of freedom (DoF) robot is issued a command to move its end-effector to a location **P** in workspace:

$$\mathbf{P}[x,\ y,\ z,\ \theta_x,\ \theta_y,\ \theta_z] \tag{1.1}$$

it will actually move to a location:

$$\mathbf{P}'[x+dx,\ y+dy,\ z+dz,\ \theta_x+\delta x,\ \theta_y+\delta y,\ \theta_z+\delta z] \tag{1.2}$$

By commanding the robot to move to the location **P** consecutively, each time the achieved location **P**' will be slightly different from **P** due to the joint servo and actuator repeatability errors. The average values of the deviations [dx, dy, dz, δx, δy, δz] from their mean are a measure of the robot's repeatability in a given direction in a given configuration.

If there were no other sources of errors in the robot except from the joint servo system, the average (mean) values of deviations [dx, dy, dz, δx, δy, δz] would be zero. However, because of other error sources, such as geometric errors in the robot components, assembly errors, transmission errors, etc., the average positioning errors are always non-zero. Therefore, when a robot is sent to many different locations in its workspace, the variation in [dx, dy, dz, δx, δy, δz] will be much larger than variations due to repeatability alone. The average values of the deviation in different configurations are a measure of the robot's accuracy in a given direction. For a six DoF robot, the position and orientation accuracy are defined as:

$$\Delta_{av} = \frac{1}{m}\sum_{i=1}^{m}\sqrt{dx_i^2 + dy_i^2 + dz_i^2} \tag{1.3}$$

$$\delta_{av} = \frac{1}{m}\sum_{i=1}^{m}\sqrt{\delta x_i^2 + \delta y_i^2 + \delta z_i^2} \tag{1.4}$$

where dx_i, dy_i, dz_i, δx_i, δy_i, δz_i are linear and angular errors measured at randomly chosen locations (i = 1, 2, ..., m) in the specific region of the robot workspace. Root Mean Square Error (RMS) can also be used as another measure of accuracy. Robot position and orientation accuracy according to the RMS are defined as :

$$\Delta_{rms} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(dx_i^2 + dy_i^2 + dz_i^2)} \tag{1.5}$$

$$\delta_{rms} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\delta x_i^2 + \delta y_i^2 + \delta z_i^2)} \tag{1.6}$$

Robot accuracy will always be positive according to the definitions above.

Another useful accuracy specification is standard deviation. Robot position and orientation standard deviations are specified as:

$$\sigma_\Delta = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(\sqrt{dx_i^2 + dy_i^2 + dz_i^2} - \Delta_{av})^2} \tag{1.7}$$

$$\sigma_\delta = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(\sqrt{\delta x_i^2 + \delta y_i^2 + \delta z_i^2} - \delta_{av})^2} \tag{1.8}$$

With the definitions of average and standard deviation errors, there is 99.7% probability that the positioning error of the end-effector will be within the limits of the linear and angular error ranges as specified by $\Delta_{av} \pm 3\sigma_\Delta$; and $\delta_{av} \pm 3\sigma_\delta$ (Kozakiewicz, Ogiso and Miyake, 1990). The average and standard deviation errors as defined above, together with the absolute error measured by the maximum deviation, are used in this work to describe robot positioning (both linear and angular) accuracy.

Robot resolution and repeatability are determined by the joint servo and actuator system of the robot, therefore they can only be changed by modifying hardware design. Robot accuracy, however, is determined by the software of the robot (robot geometric model, forward and inverse kinematic control software), and can be improved by the robot calibration process. Robot repeatability is the limit of any robot calibration efforts. Hardware based robot teaching methods, such as manual teaching robot joints by using a teaching pendant, require only good repeatability so that the robot can return to the memorised location exactly time after time. Software based robot teaching (programming), such as off-line programs, require good absolute accuracy in addition to a good repeatability because the robot is controlled indirectly through a computer

model of the manipulator, so that an exact robot model is needed to ensure that the robot will move exactly where it is commanded. Most industrial robots have reasonably good repeatability but rather poor absolute accuracy. For example, the repeatability of the commonly used PUMA robot is 0.1-1 (mm), but its accuracy is normally up to 10-20 (mm). The aim of robot calibration is to improve robot position accuracy to the order of its repeatability.

1.3 Application of Robot Calibration

Robot calibration plays an increasingly important role in all areas of robot production, integration and operation within Flexible Manufacturing Systems (FMS). The utilities of robot calibration can be explained as follows (Bernhardt and Albright, 1993):

1) Implementing off-line planned and simulated robot tasks: Whilst off-line programming can reduce significantly robot programming time and avoid costly mistakes compared with on-line teaching methods, the discrepancies between the simulated environment and the actual physical workcell must be minimised through calibration before the off-line generated programs can be implemented on a shop floor.

2) Evaluating robot production: Robot accuracy can be achieved by manufacturing the robot closely to its design specifications at a minimum tolerance. However, high precision is a costly manufacturing demand. In addition, many specifications can not be explicitly evaluated after a robot is completely manufactured and assembled. Robot calibration, on the other hand, provides a practical and effective means of accuracy improvement by implicitly determining its physical parameters.

3) Improving control of robot motion: Robot control accuracy can be improved by incorporating the identified parameters into the robot controller. Advanced control strategies can also take advantage of the precise knowledge of model parameters for accurate motion control, e.g. in adaptive control, robot kinematic parameters are assumed to be known (Bennett, Geiger and Hollerbach, 1991).

4) Monitoring robot component wear: Once a robot is operating in a flexible manufacturing system, component wear-and-tear or repairs can detrimentally affect positioning accuracy. A periodic re-calibration can be performed to determine if repairs

are necessary and/or if programs need adjusting (re-programming). If a robot is replaced, robot re-calibration also enables replacement robots to share programs of the old robot with necessary adjustments.

A robot calibration system consists of two major subsystems: measurement and data processing. Requirements for the measurement and data-processing of calibration systems are different depending on the purpose of robot calibration and the circumstances under which it is performed. When a robot is to be calibrated on the shop floor, its environment can place severe limitations on measurement and identification capabilities. While an accurate and sophisticated global measurement device is desirable for accurate identification of all model parameters in a laboratory before robot installation, it may not be practical for robot re-calibration in a production line. There are a number of robot calibration systems commercially available which are mainly for robot calibration in a controlled laboratory environment (Silma Inc., 1992). A calibration system, which is suitable for rapid re-calibration at regular intervals over the lifetime of the robot in a shop floor environment, is still not available in practice. The aim of this research is to develop measurement and data processing techniques suitable for rapid and automatic calibration in the shop-floor environment at periodic time intervals.

1.4 Overview of Thesis

This thesis focuses on the application of artificial neural network (ANN) techniques in robot positioning accuracy modelling, identification and compensation processes. The aim of this research is to develop measurement and data processing techniques suitable for robot autonomous calibration in an industrial application environment. For the purpose of this thesis, autonomous calibration is defined as a fully automated process for the robot to improve its positioning accuracy using its internal sensor measurements on-site whenever and wherever necessary (after a certain period of robot operation and in the volume where high accuracy is required). The remainder of this thesis is organised as follows:

Chapter 2 reviews the previous work in the area of robot calibration and related techniques.

Chapter 3 introduces robot kinematics based on the Denavit-Hartenberg (D-H) parameter description. The linear error model and the special Jacobian matrix are derived using a geometric approach, which are the basis for kinematic identification. A modified D-H parameter notation using an extra rotation parameter for consecutive parallel joints is discussed.

Chapter 4 develops a new kinematic identification algorithm using Hopfield continuous-valued recurrent neural networks (RNN). The network energy function is constructed such that its minimum corresponds to the minimum least square error between the actual and desired end-effector locations. The network connection weights are determined directly from the nominal kinematcs and the network neuron states represent the kinematic parameter errors to be identified. A full-pose (position and orientation) measurement scheme using a coordinate measuring machine (CMM) is described. Kinematic identification results for a six DoF Puma 560 robot are obtained using the RNN-based algorithm and conventional optimisation approaches. The identification network finds optimal solutions within a few characteristic time constants of the neural circuit, even for the singular model and the measurements are constrained to a local volume. Issues about the optimal number of measurement points and the modelling of the robot base and tool are also discussed.

Chapter 5 presents a novel robot autonomous calibration method using a trigger probe. The new method eliminates the use of any other external measuring devices to determine the robot end-effector location measurements, thus it is suitable for a periodic robot re-calibration on a production line. The kinematic constraint conditions are obtained from the known shape of the constraint surfaces, rather than from known reference locations as used by other researchers. The fully automated data collection scheme is described in detail. Kinematic identification is performed using the developed RNN-based algorithm. Both simulation and experimental results for a PUMA robot are presented, which show that robot positioning accuracy can be improved to the level of repeatability using the proposed method.

Chapter 6 discusses the development of a robot generic accuracy model which accounts for various error sources using feedforward neural networks. The generic accuracy function is introduced based on an expansion of the Fourier series, which serves as the basis for the design of a neural network architecture. The Pi-sigma network architecture is used as a generic model for robot accuracy problem because of its capability to generate higher-order trigonometric polynomial approximations

efficiently and dynamically, which is suited to the structure and the order of the generic accuracy function. Results for a six DoF Puma robot within a local volume of workspace are presented, and compared with the results of accuracy modelling using a Back-propagation network.

Chapter 7 focuses on robot accuracy compensation using ANNs, this being a subset of the inverse kinematics problem of the calibrated robot. A Pi-sigma feedforward network is used to approximate the relationship between robot nominal joint configurations and joint compensation. The trained network is used to perform a constant-time inverse compensation. While the feedforward network is effective for robot inverse compensation in a small portion of robot workspace, its training efficiency and accuracy is compromised if a large calibration volume is considered. For robot accuracy compensation which involves a large number of work points, the inverse compensation problem is reformulated and the Hopfield type recurrent neural network (RNN) is applied to the design of a robust and efficient accuracy compensator. The derivation of the RNN-based compensation algorithm is similar to that of kinematic identification, with the difference being the structure of the Jacobian matrix and the interpretation of neuron states. The RNN-based inverse compensation algorithm takes advantage of the *a priori* knowledge of kinematic structure therefore requires no training, and finds an accurate solution efficiently. Both path compensation and compensation near robot singular configurations are solved successfully using the RNN-based algorithm, and are compared with the widely-used Newton-Raphson approach.

Chapter 8 concludes the thesis and gives the directions for future research.

CHAPTER 2

REVIEW OF PREVIOUS WORK

2.1 Methods of Robot Calibration

There has been extensive robot calibration research over the past decades and good reviews of the subject can be found in (Roth, Mooring and Ravani 1987), (Hollerbach 1989) and (Mooring, Roth and Driels 1991). Calibration methods can be classified as model-based parametric calibration and model-free non-parametric calibration. Most work on model-based parametric calibration has concentrated on kinematic model-based calibration or simply kinematic calibration (Hayati 1983; Wu 1983 1984; Stone 1986; Hollerbach 1989; Zhuang 1989; Mooring, Roth and Driels 1991), while a few papers have also taken non-geometric factors such as backlash, gear eccentricity, and joint compliance into account (Whitney, Lozinski and Rourke 1986; Judd and Knasinski 1991). In the category of model-based calibration methods, geometric and/or non-geometric factors are modelled and identification techniques are applied to identify the model parameters. The identified parameters are then used in algorithms for on-line compensation. In the category of non-parametric calibration, instead of modelling and identifying specific error sources, numerical fitting methods are adopted to approximate robot inaccuracy data which has been collected from local workspace (Shamma and Whitney 1987; Kozakiewicz, Ogiso and Miyake, 1990; Rea 1992). It is difficult to judge which method is better since the relative contributions of geometric and non-geometric errors to robot inaccuracy vary from one particular robot to another. While Whitney et al (1986) reported that non-geometric errors are as significant as geometric errors in affecting robot accuracy for a geared robot (PUMA 560), Judd and Knasinski

(1991) showed that as much as 95% of robot inaccuracy arises from geometric errors. Veitschegger and Wu (1987, 1988) found that non-geometric errors only accounted for less than 0.3 (mm) of PUMA robot inaccuracy, which is in agreement with the result by Stone, Sanderson and Neuman (1986).

Generally kinematic model-based calibration is considered as a global calibration method which improves robot accuracy across the whole volume of robot space, while non-parametric calibration is a local calibration method which only works within a portion of the robot workspace. However, no clear boundary line can be drawn between these two categories of calibration. Kinematic calibration can be regarded as non-linear regression which uses kinematic functions as its basis functions. Kinematic parameters identified with data collected from the local workspace may perform better in the specific workspace than in the total work volume. This implies that these parameters do not necessarily represent the real parameters of the robot over the entire workspace but are the best fitting for the collected data in a least square sense. Therefore, some non-geometric factors can also be compensated in kinematic calibration by collecting enough data and choosing an adequate model. For most robot tasks, only accuracy over a subset of robot workspace is critical, in which most fine motions such as assembly operations are executed. As we concentrate on robot on-site calibration over a local area of robot workspace, both kinematic model-based calibration and non-parametric calibration are studied and evaluated in this work.

2.2 Kinematic Model-based Calibration Methods

Generally kinematic calibration consists of four sequential procedures: 1) modelling, 2) measurement, 3) identification, and 4) compensation, enabling precise kinematic parameters to be identified thus leading to improved accuracy. These procedures are described below. Related work has been reviewed and categorised on the basis of their primary emphasis.

1) Modelling:

A kinematic model is a mathematical description of the geometry and motion of a robot. Choosing a kinematic model to describe the relationship between robot joint space and its workspace co-ordinates is the basis for the kinematic model-based robot

calibration. Denavit-Hartenburg (D-H) homogenous transformation is a mathematical tool adopting four parameter pairs to describe the spatial relationship between manipulator workspace and joint space co-ordinates (Denavit and Hartenberg, 1955). Early work on robot calibration concentrated on robot accuracy model development based on D-H representation (Wu, 1983, 1984; Ibarra and Perrier, 1986; Zhen, 1985). Mooring (1983) and Hayati (1983) pointed out the model singularity problem inherent in the D-H formalism. Parameter jumps occur in the identification process when the D-H modelling convention is used to describe two consecutive nominally parallel axes. A modification to D-H modelling was proposed by Hayati (1983) by incorporating an extra rotation parameter for parallel revolute axes. Similar modifications were reported subsequently by Veitschegger and Wu (1986), and Judd and Knasinski (1987).

Many other alternative kinematic models have also been proposed for robot calibration. Examples of these include: the 'zero-reference model' by Mooring and Tang (1984) which avoids model singularity by not using a common normal as a link parameter; the S-model by Stone, Sanderson and Neumann (1986) which uses six parameters for each link to allow an arbitrary placement of link coordinate frames; the shape matrix model by Broderic and Cipra (1988) and Ziegert and Datseris (1990) which separates the joint variables from other link parameters based on screw theory as described by Suh and Radcliffe (1978); the CPC model by Zhuang (1989) and Zhuang, Wang and Roth (1993a,b) which is complete and parametrically continuous as it is defined for manipulator calibration. However, the kinematic models used in most existing robot controllers are still based on D-H notations. The alternative models designed for calibration need to be converted back to D-H equivalent parameters after calibration for model consistency consideration.

2) Measurement

Experimental measurements of robot end-effector locations are collected using external co-ordinate measuring devices in this phase. The actual measured locations of the robot end-effector are then compared with the locations predicted by the theoretic model to obtain the workspace inaccuracy data.

Measurement is the most difficult and time-consuming phase of robot calibration. A variety of measurement methods have been used and a survey of major techniques

designed for robot test and calibration can be found in (Lau, Dagalakis and Myers, 1988). Such techniques include the use of co-ordinate measuring machines (Driels, Swayze and Potter, 1993; Zhuang, Wang and Roth, 1993b), visual and automatic theodolites (Chen and Chao, 1986; Whitney, Lozinski and Rourke 1986; Judd and Knasinski, 1991), servo-controlled laser interferometers (Lau, Hocken and Haynes, 1985; Prenninger, Vincze and Gander, 1993; Mayer and Parker, 1994), acoustic sensors (Stone, Sanderson and Neuman 1986, and Stone 1992) and visual sensors (Tsai and Lenz, 1989; Zhuang, Wang and Roth, 1993a). The measurement devices vary considerably in their cost, ease of use and accuracy, but they all have certain drawbacks which include:

- The measuring techniques are mainly designed for robot calibration in a well-controlled laboratory environment. The robot has to be removed from its normal operating environment in order to perform the calibration.

- Trained personnel are required to operate the measuring devices properly.

- Data collection is time-consuming and difficult to automate.

- Set-up and measurement processes require a lot of human intervention. Therefore, these techniques are not suitable for robot on-site calibration in an industrial application environment.

It is known that partial pose information of robot end-effector is sufficient for complete kinematic parameter identification. Tang and Mooring (1992) utilised a mechanical fixture to obtain partial information of a robot end-effector location. The fixture consists of a flat plate with some accurately located points on it. An end-effector is designed with a flat surface at a known angle to the last axis of the robot. In the 'free' mode of the robot, the robot end-effector was manually moved to the known points of the plate and against the flat plate such that components of the end-effector position and orientation were 'measured'. Veitschegger and Wu (1988) calibrated a PUMA robot based on the use of the similar plate fixture with a set of precisely positioned holes. The end-effector with a pointing device was moved passively to the holes to make point measurements. The partial pose measurement scheme eases the requirements for measuring devices. The low cost and elimination of large-sized external measuring device make it appealing for on-site applications. However, the measuring process is not automatic and requires intensive human intervention. The success of such a scheme

also relies on the predetermined locations of reference points on the fixture. In addition, as pointed out by Driels and Swayze (1994), not every robot provides a 'free' mode in which the manipulator can be moved manually while the joint encoders are powered up and the joint servos are disabled.

The approach to use physical constraints in the workspace was further developed by Bennett and Hollerbach (1990, 1991), who proposed that a passive mechanism be used to transform the open-loop manipulator into a closed kinematic chain. The concept of autonomous robot calibration was introduced which was defined as the automated process of determining a robot's model by using only its *internal sensors* (Bennett, Geiger and Hollerbach, 1991). It has been observed that autonomous calibrations are possible for robot manipulators with either some a priori knowledge of the task constraint or redundancy of the sensing systems (e.g., adding additional links or joints to connect robot end-effector and ground, or two robots gripping together to form kinematic chain closure). Based-on these observations, the automated data collection schemes were proposed for robot calibration using LVDT (linear-variable differential transformer) ball bar system (Goswami, Quaid and Peshkin, 1993) or wired potentiometer (which can be considered as a flexible ball bar system) (Driels and Swayze, 1994) connecting the robot end-effector to the known reference point in the ground. Closed-loop constraints were formed for kinematic identification by obtaining accurate radial measurements of the ball bar or the wired potentiometer. But special fixtures are needed for such a system, which may require painstaking efforts to set up; and the added fixtures are rather difficult to model.

Autonomous calibration of hand-eye systems has also been performed by using robot joint readings and camera co-ordinate measurements to form the closed-loop constraints (Tsai and Lenz, 1989; Bennett, Geiger and Hollerbach, 1991; Zhuang, Wang and Roth, 1993a). The drawbacks for autonomous calibration of hand-eye systems are that not all robotic applications incorporate a visual camera as part of the system; and the camera measurements are known to be insufficiently accurate for manipulator calibration covering a large workspace volume. Another kind of task constraint has been proposed for robot kinematic parameter identification which utilised laser line tracking in the robot workspace (Newman and Osborn, 1993). While the motion of the robot tip-point was constrained to a line motion in the workspace, the robot joint values were recorded for kinematic identification. But only simulation results for a planar two-link manipulator were presented. An active and fully

autonomous calibration scheme was proposed by Zhong and Lewis (1995) which uses a trigger probe to touch a constraint surface in a workspace. The constraint conditions are obtained from the known shape of the constraint surface rather than the known locations of reference points. This autonomous calibration scheme will be discussed in depth in Chapter 5.

3) Identification

Kinematic parameter errors are identified in this phase by minimising the collected workspace inaccuracy in the least mean square sense. Kinematic identification is basically a standard non-linear or linear least square optimisation procedure. Non-linear algorithms do not require the identification Jacobian and are computationally more robust but more computation time is required for convergence. Linear least square algorithms require less computation time to converge but suffer from numerical problems of ill-conditioning of the identification Jacobian. Robust minimisation techniques such as the Levenberg-Marquardt algorithm have been applied to cope with the problem at the expense of computation time (Bennett and Hollerbach, 1991; Mooring and Padavala, 1989). More advanced parameter estimation techniques are also applied in kinematic identification. A maximum likelihood estimator was used by Renders et al (1991). Mooring, Roth and Driels (1991) applied Kalman Filtering techniques to investigate the relationship between calibration accuracy and measurement noise.

To improve kinematic identification robustness and efficiency, some theoretic issues have been addressed by a number of researchers. Kinematic identifiability was defined by Bennett and Hollerbach (1991). Meng and Borm (1988) introduced an observability index to find the optimal measurement configurations for robot calibration, while Khalil, Gautier and Enguehard (1991) used the condition number of the identification Jacobian to determine optimum calibration configurations. Experimental and simulation studies were performed by Borm and Menq (1989, 1991) and Pathre and Driels (1990) to demonstrate the importance of observability to kinematic identification. Determining the optimal configurations for robot calibration according to the observability criteria is a high dimensional non-linear optimisation problem. An advanced optimisation technique, simulated annealing, was used by Zhuang, Wang and Roth (1994) for off-line selection of measurement configurations. Generally the optimal measurement

configurations determined by using the observability index are that the measurement points should spread across the whole workspace as widely as possible. This observation is useful for robot calibration performed in the laboratory environment where the robot can be controlled to move to arbitrary configurations. For robot on-site calibration in a crowded industrial environment, the calibration movement of robots is normally constrained. Given the limitation of constrained movement for data collection, numerically more robust and efficient algorithms are needed for robot on-site calibration processing. A Hopfield-type recurrent neural network(RNN)-based algorithm was proposed by Zhong and Lewis (1994) for efficient and robust kinematic identification, which is the focus of Chapter 4.

4) Compensation

Implementation of the identified kinematic model is the final and crucial stage of kinematic calibration. Due to the difficulty in modifying kinematic parameters in the robot controller directly, joint compensations are made to the encoder readings of the robot obtained by solving the inverse kinematics of the calibrated robot. The assumption of simplified kinematic structure which applies to the nominal robot is no longer valid for the calibrated robot due to kinematic parameter changes. Therefore the inverse kinematics of the calibrated robot is generally not analytically solvable. Numerical algorithms such as the Newton-Raphson approach are normally adopted to find the joint corrections needed to compensate for Cartesian errors (Kirchner, Gurumoorthy and Printz, 1987; Mirman and Gupta, 1992). However, the Newton-Raphson method is based on iterative inversion of the compensation Jacobian, therefore on-line compensation is problematic due to the computation expense, and the algorithm breaks down in the vicinity of robot singular configurations. The differential transformation compensation algorithm was presented by Veitschegger and Wu (1988), in which two nominal inverse problems are solved for one task point compensation. A comparison of various compensation algorithms was made by Vuscovic (1989). Zhuang, Hamano and Roth (1989) who formulated the accuracy compensation as a linear optimal control problem such that the linear quadratic regular method was applied to the design of a robust accuracy compensator. Existence and uniqueness are ensured in robot configurations near singularities by adding a regulation term to the performance index. The computation of the linear quadratic regulator algorithm is rather expensive, though simplification can be made in special cases. Zhong and Lewis

(1994), and Zhong, Lewis and N-Nagy (1995) presented neural network-based algorithms for inverse compensation. The neural network-based inverse compensation algorithms will be discussed in detail in Chapter 7.

2.3 Non-parametric Calibration Methods

Model-based parametric calibration is limited by the inability to model and identify all error sources which contribute to robot inaccuracy. Non-parametric calibration, on the other hand, employs non-parametric methods to establish an approximation function based on a sufficient number of measurement data collected from the local volume. Shamma and Whitney (1987) distinguished between forward calibration, which determines the end-effector location from joint angles, and inverse calibration, which determines the joint angles from the end-effector location. The inverse calibration was considered by Shamma and Whitney, in which the third-order trivariate polynomials were applied as approximation functions to relate the end-effector location to joint angles. The single calibration of a six DoF PUMA robot was separated into two calibrations, which comprised the first three major DoF calibration and then the remaining three minor DoF calibration. The training data points were generated by Tchebychev spacing in one quadrant of the robot workspace. Simulation showed that accuracy was reduced to below 0.3 (mm). Direct extension of the three DoF robot to a general robot would be difficult due to the limitation of polynomial approximations. In the case that a higher DoF are considered, the polynomial functions required will be too complex to be determined by the least square solutions, and would require a large number of data points which are practically difficult to obtain.

Mooring , Roth and Driels (1991) discussed a table lookup scheme for a simple two-link planar robot based on CMAC (Cerebellar Model Articulation Controller), which was originally developed by Albus (1975a,b) to model the function of the cerebellar cortex of the brain, but it can also be used as a general purpose function approximator. Even for a simple two-link planar robot, CMAC implementation of inverse kinematics exhibited unacceptable low accuracy and poor interpolation ability, and required a large number of training points. It concluded that CMAC scheme is still not ready to use for multiple joint robots. Another table lookup scheme was proposed by Everestt and McCarroll (1986) which was based on a finite element method.

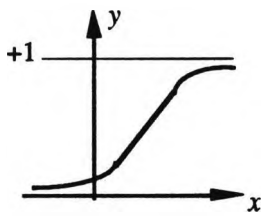
Kozakiewicz, Ogiso and Miyake (1990) applied a multi-layered neural network approximation of the joint corrections for a four DoF Scara robot. Simulations were performed which included non-geometric model such as joint compliance. Comparisons with the polynomial approximations showed that the neural network gave poor accuracy. More recently, Miyazaki, Maekawa and Bamba (1992) proposed a hybrid compensation method to improve positioning accuracy of industrial robots by introducing a feedforward layered neural network in addition to the conventional kinematic model. The maximum position error for test points was improved from 17.67 (mm) before compensation to 1.73 (mm) after kinematic calibration, to 4.30 (mm) after neural network compensation, and to 1.01 (mm) after both kinematic calibration and neural network compensation were used. Only forward calibration was discussed and inverse compensation was not addressed. Zhong, Lewis and Rea (1994) proposed a generic accuracy compensator for industrial robots based on the Pi-sigma neural network. The ANN-based accuracy compensation eliminates the need for model-based calibration, with the various error sources being represented in the distributed network weight connections. The ANN-based forward compensation will be discussed in Chapter 6 and the inverse compensation discussed in detail in Chapter 7.

2.4 Artificial Neural Network Techniques

Artificial Neural Networks (ANNs) have emerged from studies of how human and animal brains perform operations. The human brain is made up of many millions of individual processing elements, called neurons, that are highly interconnected. Artificial neural networks are made up of individual models of the biological neuron (artificial neurons or nodes) that are connected together to form a network. The neuron models that are used are typically much simplified versions of the actions of a real neuron. Information is stored in the network often in the form of different connection strengths, or weights, associated with the synapses in the artificial neuron models. A neuron model processes information by summing the weighted inputs to the neuron and passing the result through some non-linear activation functions such as a sigmoid function to an output.

There are many types of neural network available, depending on the specific arrangements of artificial neurons and their interconnections. The most widely used neural network architecture is the Multi-Layered Perceptron (MLP) because of its

simplicity. The network consists of an input layer, a number of hidden layers (typically only one or two hidden layers are used) and an output layer as shown in Figure 2.1. Data flows through the network in one direction only, from input to output; hence, this type of network is called a feedforward network. The most common training algorithm for the network is back-propagation algorithm originally proposed by Werbos (1974) and Rumelhart, Hinton and Williams (1986). An important feature of the MLP is that this network can accurately represent any continuous non-linear function relating inputs and outputs (Hornik, 1991; Hornik, Stinchcombe and White, 1990). Hence, the MLP network exhibits potential for many applications which can be formulated as a non-linear mapping problem. Other famous neural network architecture include Hopfield networks (Hopfield and Tank, 1986); Counter-Propagation networks (Hecht-Nielsen, 1990); and self-organising Kohonen networks (Kohonen, 1984), etc.



Sigmoid Function

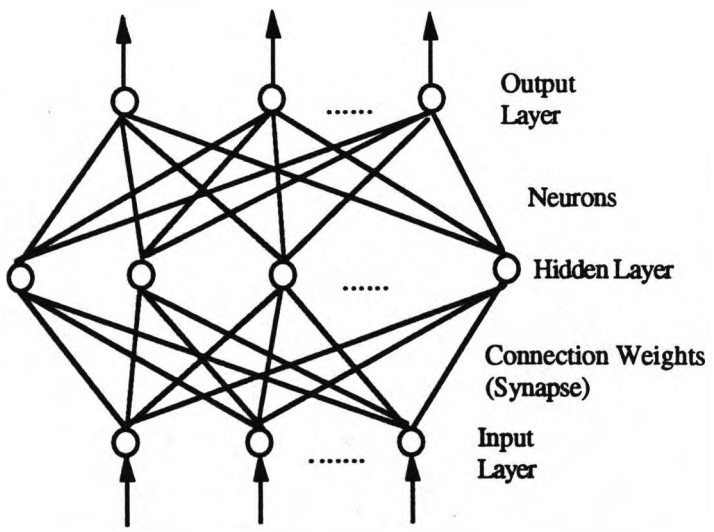


Figure 2.1 Biologically-Inspired Neurocomputing Model --- Multiple-Layered Perceptron with One Hidden Layer

2.5 Artificial Neural Network Applications in Robotics

A variety of uses for ANNs related to robotics and control have recently been reported. The use of ANN control is particularly suited to problems related to the control capabilities of animal nervous systems, and robot inverse kinematics transformation application naturally fall into this domain of applications. One of the earliest ANN approaches to robot control is due to Albus (1975b). The basic idea of his CMAC scheme for robot control is to compute control commands by look-up tables rather than by solving control equations analytically. While the CMAC has a computational advantage over conventional approaches due to its distributed fashion, it requires a large size of computer memories for multiple DoF robot, and is not able to perform interpolations. Kuperstein (1988) concerned himself with models of visual motor coordination in robots. While he did not explicitly address the inverse kinematics problem, his work did in fact use neural networks to obtain the transformation needed to convert desired hand coordinates in Cartesian space into appropriate joint coordinates. A neural controller called INFANT, which learns sensory-motor coordination from its own experience, has been reported which achieved an average positioning accuracy of 3% of the arm's length in position and 6 degrees in orientation (Kuperstin and Rubinstein, 1989). Other attractive features of the INFANT include real-time operation, learning and maintaining its own calibration, and fault tolerance.

Inverse kinematic control has also been studied by a number of researchers, e.g., Guez and Ahmad (1988), Josin (1988) and Josin, Charney and White (1988), using back-propagation learning algorithms for feedforward networks. However, the back-propagation network-based inverse kinematics solutions are typically not accurate enough for practical applications even for 2 or 3 DoF robots. Attempts to apply back-propagation directly to systems with more DoF have not been very successful (Kozakiewicz, Ogiso and Miyake, 1991; Daunicht, 1991), since these systems typically exhibit high-order nonlinearities and hence very slow learning rate and unacceptable learning accuracy. To exploit both ANN efficiency and numerical accuracy, Ahamad and Guez (1990) proposed a hybrid approach that used the ANN outputs as a initial solution for iterative numerical Newton-Raphson method, which resulted in a reduced number of iterations of the numerical method.

All the papers cited above concentrate on position-based inverse kinematic control where only position (location) information needs to be converted. In velocity-based inverse kinematic control or Jacobian control (Fu, Gonzalez and Lee, 1987), both position and velocity information needs to be transformed from Cartesian space to joint space. Velocity-based inverse kinematics is much more complex than position-based inverse kinematics since the number of input variables is doubled. The network training problem may become intractable with the increase in the dimensionality of the input space since the input space will experience an exponential growth in size (Yeung, 1989). Following the divide-and-conquer principal, Yeung (1989), and Yeung and Bekey (1989) proposed context-sensitive networks which partitioned the set of input variables into two groups. One set is used as the input to the network which approximates the basic mathematic operations being represented (the function network), while the second set determines the setting or context within which the function is determined (context network). They have shown that context-sensitive networks improved learning accuracy and reduced convergence time drastically compared with the standard back-propagation networks. Similar network architecture has been used by Bassi and Bekey (1989) to extend the work to inverse dynamics learning. A complete solution to the inverse dynamics problem has been presented by Miyamoto, Kawato, et al (1988). With a priori detailed knowledge of the dynamics equation for a three DoF robot, they decomposed the network into 26 sub-networks according to the primitive non-linear function terms in the analytic dynamic equations. The performance of the system is excellent due to the simplification of the learning task for sub-networks, which is equivalent to the determination of the coefficients in the dynamics equations.

The Hopfield type recurrent neural network (RNN) architecture (Hopfield and Tank, 1986) has been applied to the velocity-based inverse kinematics problems for robots with redundant DoF. Guo and Cherkassky (1989) implemented the Jacobian control scheme using the Hopfield analogue (continuous-valued) computation model. The states of neurons represent joint velocities of a manipulator, and the connection weights are determined from the current value of the Jacobian matrix. The network energy function is constructed so that its minimum corresponds to the minimum least square error between the actual and desired joint velocities. Simulation shows that the method is capable of solving the inverse kinematics problem for a planar redundant manipulator in real time. In contrast to the feedforward neural network-based inverse kinematics solutions, the RNN-based algorithm, by taking advantage of the kinematic structure of

the specific robot, requires no training and can find quality solutions within a few characteristic time constants of neural circuits. Li and Jiang (1993) extended Guo and Cherkassky's work by integrating the optimising properties of the RNN-based inverse kinematic control and the technique of the Variable Structure Control (VSC). The pseudoinverse Jacobian control scheme was implemented using an RNN algorithm for a planar redundant robot (Wu and Wang, 1994).

Other neural network architecture have also been vigorously investigated for robotic applications. Martinez, et al (1990) have shown that an extension of Kohonen's algorithm (Kohonen, 1984) for the formation of topological correct feature maps, together with an error-correction rule of the Widrow-Hoff type, can learn to control the robot arm and gripper movement by using only the input signals of two cameras. Wu, Jiang and Shiau (1993) used the modified two-layered counter-propagation network (Hecht-Nielsen, 1990) to control a robot's gross motion (first layer) and fine motion (second layer). The counter-propagation network, which combines the Kohonen self-organising feature map with the Grossberg outstar map (Grossberg, 1982), can be a statistically optimal self-programming lookup table for the adaptive control of robots. However, these neural network learning algorithms belong to unsupervised learning therefore they are not associative (Yeung, 1989). This implies that the training of such networks normally requires a large number of training data and the interpolation ability of the trained networks are typically poor. A more comprehensive review on the various ANN architecture and their applications in robot task planning, path planning, and sensor/motor control can be found in Kung and Hwang (1989).

2.6 Conclusions

Previous work on robot calibration and related techniques have been reviewed in this chapter. Although a great deal of research has been done on robot calibration over the past decade, most of the calibration techniques developed are only suitable for robot calibration within well-controlled laboratory environments. Rapid autonomous robot calibrations within a shop floor environment, although highly desirable, are still not available in practice. Efficient and robust data processing techniques and fully automated data collection methods are required to perform on-site calibration on a regular basis and within local workspace. Artificial neural networks are appealing for robot calibration processing including modelling, identification and compensation, due to their computational power, learning abilities, and fault tolerance. Selection of neural

network architecture are critical for their successful applications in robotics and *a priori* model knowledge are useful for designing NN architecture. A measurement method capable of collecting data from a local workspace automatically using portable physical constraints needs to be developed for on-site calibration,

CHAPTER 3

ROBOT KINEMATICS AND KINEMATIC ERROR MODELLING

3.1 Introduction

Robot tasks are normally described in terms of the relative locations (positions and orientations) of the workpieces and equipment which exist in the working environment. The study of kinematics reveals that the relative locations of objects can be defined clearly by attaching co-ordinate frames to each object so that when the object moves, so does the frame. The spatial transformation between robot end-effector location and its individual link geometry and joint movement is established in terms of the assigned Cartesian co-ordinate frames fixed relative to each of the links. The (4 x 4) *Homogenous* transformation matrix introduced by Denavit and Hartenberg (1955) and later adopted by Paul (1982) has become the most common approach to describing spatial transformations in robotics. In this chapter, we will review the Denavit-Hartenberg (D-H) method for robot kinematic modelling, and later develop a kinematic error model which describes the relationship between robot kinematic parameter variations and the predicted end-effector location error, which is the basis for kinematic calibration.

The kinematic error model for a single link is normally derived using the analysis approach of the *homogenous* transformation matrix and then applying it to the entire robot. The analytic expression of the coefficient matrix which gives a linear relationship between the kinematic parameter errors and the end-effector error was derived by Wu (1984) and Veitschegger and Wu (1986). The coefficient matrix has

been termed by Mirman and Gupta (1992) as a special Jacobian matrix which has been derived using a similar approach. A disadvantage of using the analysis approach is that the derivation requires a lot of mathematical operations and geometric interpretation is not obvious. Vuscovic (1989) presented a geometric expression for the special Jacobian matrix and termed it differently as kinematic sensitivities, but no derivation process was given. In section 3.3, a geometric approach to deriving the kinematic model is developed based on the theory of rigid body kinematics with moving (translating and rotating) co-ordinate frames. The geometric approach to formulating the special Jacobian matrix is straightforward and the geometric interpretation of the Special Jacobian is useful for identifying model singularities. The D-H model singularity for consecutive parallel revolute joints is then discussed. Finally, a modified D-H notation which overcomes the model singularity is introduced for use in following chapters.

3.2 Kinematic Modelling Using Denavit-Hartenberg Model

For robot manipulators with general kinematic structure of linkages, their complex spatial orientation and position can be specified by allocating kinematic frames to each of the robot links and then specifying transformation from one link to another. Denavit and Hartenberg (1955) interpret the sequential transformation from one link to another as a multiplicative operation of (4 x 4) matrix:

$$\mathbf{T}_n = \mathbf{A}_1 * \mathbf{A}_2 * \ldots * \mathbf{A}_i * \ldots * \mathbf{A}_n \tag{3.1}$$

where $\mathbf{T}_n$ is a (4 x 4) homogenous transformation matrix which has the form:

$$\mathbf{T}_n = \begin{bmatrix} n_x & s_x & o_x & p_x \\ n_y & s_y & o_y & p_y \\ n_z & s_z & o_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

and in terms of its vector components, $\mathbf{n} = [n_x\ n_y\ n_z]^T$, $\mathbf{s} = [s_x\ s_y\ s_z]^T$, $\mathbf{o} = [o_x\ o_y\ o_z]^T$ are three unit vectors specifying the orientation of the **x**, **y** and **z** axis of the co-ordinate frame associated with $\mathbf{T}_n$ with respect to a reference frame, while $\mathbf{p} = [p_x\ p_y\ p_z]^T$ specifies the position of the origin of that frame in a reference frame. For robot manipulators with n-links, the co-ordinate frame associated with $\mathbf{T}_n$ is normally attached to the robot end-effector frame while the reference frame is the robot base

frame. Therefore $\mathbf{T}_n$ determines the robot end-effector orientation and position completely in the robot base co-ordinate frame.

$\mathbf{A}_i$ is a (4 x 4) homogenous transformation matrix which represents the spatial transformation between the (i-1) frame fixed to the (i-1) link and i-th frame fixed to the i-th link of the robot. Its contents depend on the specific kinematic structure of the i-th link of the robot manipulator and the assignment of the kinematic parameters. The D-H parameters are defined through allocation of co-ordinate frames to each link using a set of rules to locate the origin of the frame and the orientation of the axes. The D-H frame assignment procedures are summarised as follows.

The process is begun by identifying the axis of motion for each link. Next, the common normal between consecutive joint axes is then identified. The origin of co-ordinate frame i (attached to the i-th link) is then located at the intersection of joint axis (i+1) and the common normal between axis (i+1) and i. The **z**-axis of co-ordinate system i points along the axis of joint (i+1) and the x axis is aligned with the common normal. Once the **x**, **z**-axes for frame i are determined, the **y**-axis can be decided using right hand rule. After the frame system is assigned to the link, the kinematic parameters are then defined to describe the geometric relation between consecutive frames. Figure 3.1 shows the D-H parameter assignment for link i with revolute joints. The parameter assignment for prismatic joint follows the same rule, the only difference with the revolute joint is that a different interpretation of the joint variable applies.

With reference to Figure 3.1, the transformation between frame (i-1) and frame i can be interpreted as the following sequential steps:

- rotate frame (i-1) about $\mathbf{z}_{i-1}$ by an angle θ_i, the joint angle;
- translate along $\mathbf{z}_{i-1}$ a distance d_i, the link offset;
- translate along the rotated $\mathbf{x}_{i-1}$, a distance a_i, the link length;
- rotate about $\mathbf{x}_i$ the twist angle α_i.

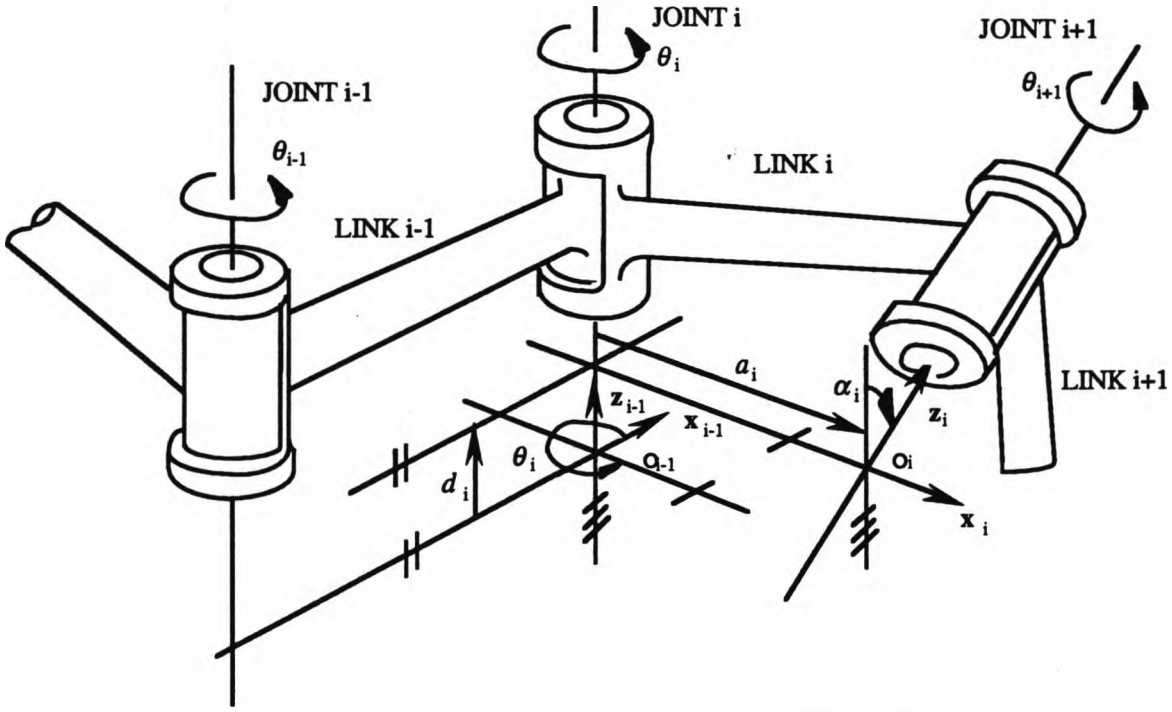


Figure 3.1. Denavit-Hartenberg Parameters for a Revolute Joint

Using Paul's notation of the primitive transformations (Paul, 1982), the above transformation procedure can be written in the matrix form such that the homogenous transformation matrix $\mathbf{A}_i$ is defined as:

$$\mathbf{A}_i = \mathbf{Rot}(\mathbf{z}_{i-1}, \theta_i)\mathbf{Trans}(0, 0, d_i)\mathbf{Trans}(a_i, 0, 0)\mathbf{Rot}(\mathbf{x}_i, \alpha_i) \qquad (3.3)$$

where **Rot**(.) and **Trans**(.) are the primitive transformation matrices of rotation and translation:

$$\mathbf{Rot}(\mathbf{z}_{i-1}, \theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Trans}(0, 0, d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Trans}(a_i, 0, 0) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Rot}(\mathbf{x}_i, \alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Expanding equation 3.3, the general form of the homogenous transformation matrix $\mathbf{A}_i$ is:

$$\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

As shown in Equation 3.4, the homogenous transformation is a function of the link geometry such as the link length a_i, the twist angle α_i, the offset d_i, and the joint angle θ_i. This equation may be used as a recursive transformation relating the position and orientation of one frame with respect to the previous one. Using Equation 3.4 in Equation 3.1, let $i = 1, 2, ..., n$, and frame 0 representing robot base frame and n is the number of robot links, then $\mathbf{T}_n$ represents the position and orientation of robot end-effector frame with respect to the base frame, which is the function of $4n$ kinematic parameters. The position and orientation of robot end-effector frame in the Cartesian base frame can also be represented in the more compact vector form using a six-element vector $\mathbf{x} = [\mathbf{p}^T, \mathbf{w}^T]^T$, rather than a (4 x 4) matrix using 12 significant elements as in Equation 3.1, where $\mathbf{p} = [p_x, p_y, p_z]^T$ is a position vector which takes the first three row elements of the last column of the homogenous matrix $\mathbf{T}_n$, and $\mathbf{w}$ is the orientation vector which has various forms using three independent angle elements to represent the orientation of co-ordinate frame. Typical orientation representations include Roll-Pitch-Yaw and Eulers angles. The conversion between Euler angles and an orientation transformation matrix $[\mathbf{n}, \mathbf{s}, \mathbf{o}]$ written in MATLAB M-files (Mathworks Inc., 1992a) is shown in Appendix 1. Robot end-effector position and orientation is written in vector form:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \hline \mathbf{w} \end{bmatrix} = \mathbf{f}(\mathbf{a}, \mathbf{d}, \alpha, \theta) \tag{3.5}$$

where $\mathbf{a} = [a_1, a_2, ..., a_n]$ is the link offset vector, $\mathbf{d} = [d_1, d_2, ..., d_n]$ is the link length vector, $\alpha = [\alpha_1, \alpha_2, ..., \alpha_n]$ is the twist angle vector, $\theta = [\theta_1, \theta_2, ..., \theta_n]$ is the joint angle vector. and **f**(.) is a non-linear function mapping robot kinematic parameters to robot end-effector position and orientation, which can be obtained using the above homogenous transformation. Normally (**a, d,** α) are the robot geometric parameters which are specified by a robot manufacturer, while θ is the controllable joint variables.

The process to find the end-effector position and orientation, given robot joint variable, is called *forward kinematics* in robotics, whilst the process to find joint variables, given the end-effector position and orientation, is called the *inverse kinematics*. For robot calibration problems, both forward and inverse kinematics are involved. Generally the modelling and identification phase of calibration are considered as forward kinematics, that is, given robot joint variable and geometric errors, to estimate the end-effector position and orientation errors. The implementation phase of calibration is the inverse process which compensates the end-effector error in the robot joint encoders. The forward kinematics of PUMA 560 Robot written in MATLAB M-files (MathWorks, 1992a) is appended in Appendix 1. In general, inverse kinematics involves a numerical procedure since **f**(.) in Equation 3.5 is a multivariate non-linear function. Closed-form analytic inverse solutions are available for some industrial robots with simple-form kinematics. The analytic inverse solution for Puma 560 robot written in MATLAB M-files is shown in Appendix 2. In the next section, a kinematic error model will be derived which estimates the end-effector position and orientation, given geometric parameter and joint variable errors.

3.3 Kinematic Error Model and Special Jacobian Matrix

For reasons described in the previous chapters, the actual kinematic parameters will deviate from nominal values specified by the robot manufacturer, which in turn causes robot end-effector positioning inaccuracy since the robot is controlled by a kinematic control model which is based on nominal parameters. Let $\rho^0 = [\mathbf{a}^0, \mathbf{d}^0, \alpha^0, \theta^0]$ be the nominal kinematic parameters and $\Delta\rho = [\Delta\mathbf{a}, \Delta\mathbf{d}, \Delta\alpha, \Delta\theta]$ be the kinematic parameter error vector which is a small perturbation from the nominal value. The end-effector

position and orientation error $\Delta\mathbf{x}$ due to the kinematic error can then be obtained using Equation 3.5:

$$\Delta\mathbf{x} = \mathbf{f}(\mathbf{a}^0+\Delta\mathbf{a}, \mathbf{d}^0+\Delta\mathbf{d}, \alpha^0+\Delta\alpha, \theta^0+\Delta\theta) - \mathbf{f}(\mathbf{a}^0, \mathbf{d}^0, \alpha^0, \theta^0) \tag{3.6}$$

Expanding Equation 3.6 using Taylor series around nominal kinematic parameters, and ignoring second and higher order terms, Equation 3.6 becomes:

$$\Delta\mathbf{x} = \frac{\partial\mathbf{f}}{\partial\mathbf{a}}\Delta\mathbf{a} + \frac{\partial\mathbf{f}}{\partial\mathbf{d}}\Delta\mathbf{d} + \frac{\partial\mathbf{f}}{\partial\alpha}\Delta\alpha + \frac{\partial\mathbf{f}}{\partial\theta}\Delta\theta = \mathbf{J}\Delta\rho \tag{3.7}$$

where $\mathbf{J} = [\frac{\partial\mathbf{f}}{\partial\mathbf{a}}, \frac{\partial\mathbf{f}}{\partial\mathbf{d}}, \frac{\partial\mathbf{f}}{\partial\alpha}, \frac{\partial\mathbf{f}}{\partial\theta}]$ and is called a special Jacobian matrix to distinguish the ordinary Jacobian matrix $\mathbf{J}_\theta = \frac{\partial\mathbf{f}}{\partial\theta}$ as defined by Paul (1982). Using the position and orientation vector $\mathbf{p}$ and $\mathbf{w}$ to replace $\mathbf{f}(.)$ as in Equation 3.5, the special Jacobian matrix can be portioned into its position and orientation component as follows:

$$\mathbf{J}_p = \left[\frac{\partial\mathbf{p}}{\partial\mathbf{a}}, \frac{\partial\mathbf{p}}{\partial\mathbf{d}}, \frac{\partial\mathbf{p}}{\partial\alpha}, \frac{\partial\mathbf{p}}{\partial\theta}\right] \tag{3.8}$$

$$\mathbf{J}_w = \left[\frac{\partial\mathbf{w}}{\partial\mathbf{a}}, \frac{\partial\mathbf{w}}{\partial\mathbf{d}}, \frac{\partial\mathbf{w}}{\partial\alpha}, \frac{\partial\mathbf{w}}{\partial\theta}\right] \tag{3.9}$$

As can be seen from Equation 3.7, the special Jacobian matrix plays a vital role in transforming individual link kinematic error to the end-effector positioning inaccuracy. The published work (Veitschegger and Wu, 1986; Mirman and Gupta, 1992) on the derivation of the special Jacobian matrix is based on analytic methods which involve complex and abstract mathematical operations. A geometric approach to deriving the detailed structure of the special Jacobain matrix is given below which is straightforward and has direct physical interpretations.

Firstly the theory of absolute movement with respect to the stationary base frame and the relative movement with respect to the moving frame should be introduced. The relationship between the base frame and the moving frame (rotating and translating) was established in Fu, Gonzalez and Lee (1987) (A more thorough vectorial treatment of rigid body kinematics is referred to the textbook by Easthope (1964)). With reference to Figure 3.2, let $\mathbf{v}_0$ and ω_0 be the translation and rotation speed of the frame $o^*x^*y^*z^*$ with respect to the base frame $ox_0y_0z_0$, and $\mathbf{v}^*$ and ω^* be the translation and rotation speed of the frame $o_px_py_pz_p$ with respect to the moving frame $o^*x^*y^*z^*$. The

origin points of the frame $o^*x^*y^*z^*$ and frame $o_px_py_pz_p$ are specified by the vectors $\mathbf{p}_0$, $\mathbf{p}$ respectively with respect to the base frame, and let $\mathbf{p}^*$ be the vector with respect to the base frame connecting o^* and o_p. Then the translation and rotation speed $\mathbf{v}$ and $\mathbf{w}$ of the frame $o_px_py_pz_p$ with respect to the base frame is (Fu, Gonzalez and Lee, 1987):

$$\mathbf{v} = \mathbf{v}^0 + \mathbf{v}^* + \omega \times \mathbf{p}^* \tag{3.10}$$

$$\omega = \omega_0 + \omega^* \tag{3.11}$$

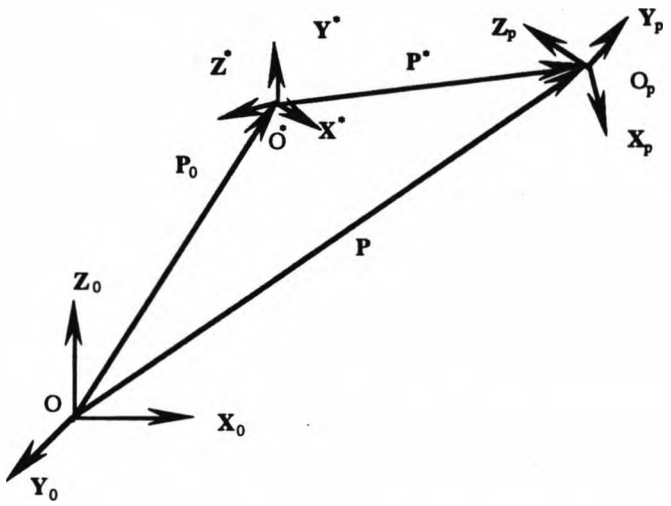


Figure 3.2. Relationship between in Moving Co-ordinate Frames and Base Frame

Multiplying both sides of the Equation (3.10) and (3.11) by the infinitesimal time change dt, then we obtain the infinitesimal translation and rotation changes correspondingly:

$$d\mathbf{p} = d\mathbf{p}_0 + d^*\mathbf{p}^* + d\mathbf{w} \times \mathbf{p}^* \tag{3.12}$$

$$d\mathbf{w} = d\mathbf{w}_0 + d^*\mathbf{w} \tag{3.13}$$

where $d\mathbf{p}_0$ and $d\mathbf{w}_0$ are the differential translation and rotation change of the frame $o^*x^*y^*z^*$ with respect to the base frame respectively, $d^*\mathbf{p}^*$ and $d^*\mathbf{w}$ are the differential translation and rotation change of the frame $o_px_py_pz_p$ with respect to the frame $o^*x^*y^*z^*$ respectively, and the $d\mathbf{p}$ and $d\mathbf{w}$ are the differential translation and rotation change of the frame $o_px_py_pz_p$ with respect to the base frame $ox_0y_0z_0$ respectively.

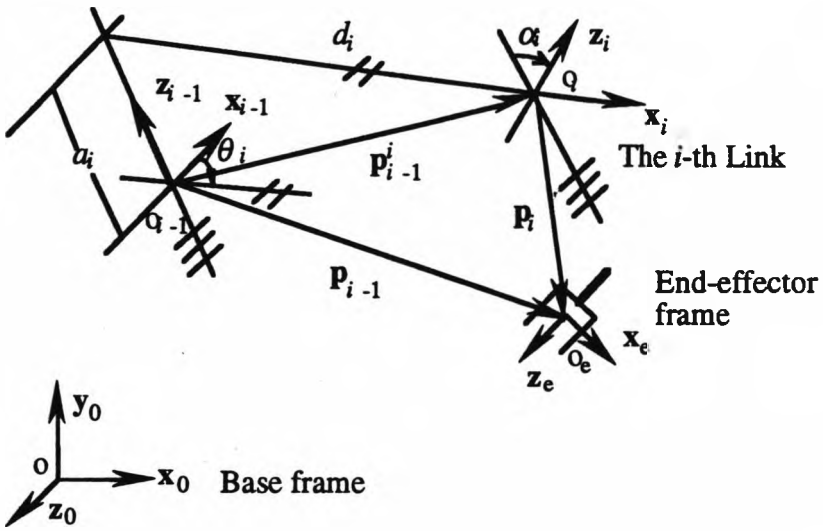


Figure 3.3. Relationship between the differential change of the end-effector frame and the differential changes in the *i*-th link parameters.

Next we apply the relationship of the differential translation and rotation change between moving frames and base frame to the derivation of the special Jacobain matrix. With reference to the Figure 3.3, we assign the starred moving frame in Figure 3.2 to the (i-1)-th frame of the robot, and the frame $o_p x_p y_p z_p$ in Figure 3.2 to the robot end-effector frame. To develop the relationship between the i-th link parameter changes and the end-effector position and orientation change, the end-effector frame is rigidly attached to the i-th link by fixing all the link movements after the i-th link. Using Equation 3.12 and 3.13, we have:

$$d\mathbf{p} = d\mathbf{p}^{i-1} + d^*\mathbf{p}_{i-1} + d\mathbf{w} \times \mathbf{p}_{i-1} \tag{3.14}$$

$$d\mathbf{w} = d\mathbf{w}^{i-1} + d^*\mathbf{w} \tag{3.15}$$

where $d\mathbf{p}$ and $d\mathbf{w}$ represent the differential translation and rotation changes of the end-effector frame with respect to the base frame respectively, $d\mathbf{p}^{i-1}$ and $d\mathbf{w}^{i-1}$ represent the differential translation and rotation change of the (i-1)-th frame with respect to the base frame respectively, and $d^*\mathbf{p}_{i-1}$ and $d^*\mathbf{w}$ are differential translation and rotation of the end-effector frame with respect to the (i-1)-th frame respectively.

Recalling the definition of the D-H parameters and the fact that the end-effector frame is rigidly attached to the i-th link, the relationship between the differential change

of the end-effector frame and the differential changes in the i-th link parameters can be described in the following manner.

1) Allowing the joint variable θ_i a differential change $d\theta_i$ (rotating around $\mathbf{z}_{i-1}$), then the relative rotation with respect to the (i-1)-th frame $d^*\mathbf{w} = d\theta_i\mathbf{z}_{i-1}$, and the relative translation with respect to the (i-1)-th frame $d^*\mathbf{p}_{i-1} = d\theta_i\mathbf{z}_{i-1} \times \mathbf{p}_{i-1}$. From Equation 3.14 and 3.15, we have:

$$d\mathbf{p} = d\mathbf{p}^{i-1} + d\mathbf{w}^{i-1} \times \mathbf{p}_{i-1} + d\theta_i\mathbf{z}_{i-1} \times \mathbf{p}_{i-1} \tag{3.16}$$

$$d\mathbf{w} = d\mathbf{w}^{i-1} + d\theta_i\mathbf{z}_{i-1} \tag{3.17}$$

Differentiating Equation 3.16 and 3.17 with respect to θ_i, and note that $d\mathbf{p}^{i-1}$ and $d\mathbf{w}^{i-1}$ are independent of $d\theta_i$ therefore their derivatives with θ_i are equal to zero; and that the derivative of vector $\mathbf{p}_{i-1}$ with respect to θ_i is equal to zero due to the end-effector is rigidly attached to the i-th frame, we have:

$$\frac{\partial \mathbf{p}}{\partial \theta_i} = \mathbf{z}_{i-1} \times \mathbf{p}_{i-1} \tag{3.18}$$

$$\frac{\partial \mathbf{w}}{\partial \theta_i} = \mathbf{z}_{i-1} \tag{3.19}$$

2) Allowing joint variable a_i a differential change da_i (along the $\mathbf{z}_{i-1}$), then the relative rotation with respect to the (i-1)-th frame $d^*\mathbf{w} = \mathbf{0}$, and the relative translation with respect to the (i-1)-th frame $d^*\mathbf{p}_{i-1} = da_i\mathbf{z}_{i-1}$. From Equation 3.14 and 3.15, we have:

$$d\mathbf{p} = d\mathbf{p}^{i-1} + da_i\mathbf{z}_{i-1} + d\mathbf{w}_{i-1} \times \mathbf{p}_{i-1} \tag{3.20}$$

$$d\mathbf{w} = d\mathbf{w}^{i-1} \tag{3.21}$$

Differentiating Equation 3.20 and 3.21 with respect to a_i and noting that $d\mathbf{p}^{i-1}$ and $d\mathbf{w}^{i-1}$ are independent of da_i therefore their derivatives with a_i equals to zero, and that the derivative of vector $\mathbf{p}_{i-1}$ with respect to a_i is equal to zero due to that the end-effector is rigidly attached to the i-th frame, we have:

$$\frac{\partial \mathbf{p}}{\partial a_i} = \mathbf{z}_{i-1} \tag{3.22}$$

$$\frac{\partial \mathbf{w}}{\partial a_i} = \mathbf{0} \tag{3.23}$$

3) Let dd_i be the differential change in parameter d_i (along the axis $\mathbf{x}_i$), similar to the derivation of the Equation 3.22 and 3.23, we have:

$$\frac{\partial \mathbf{p}}{\partial d_i} = \mathbf{x}_i \tag{3.24}$$

$$\frac{\partial \mathbf{w}}{\partial d_i} = \mathbf{0} \tag{3.25}$$

4) Let $d\alpha_i$ be the differential change in parameter α_i (rotate about the axis $\mathbf{x}_i$), recalling that the i-th frame is fixed to the i-th link and the end-effector frame is rigidly attached to the i-th link, the relative differential rotation of end-effector frame with respect to the (i-1)-th frame equals the relative rotation of the i-th frame with respect to the (i-1)-th frame, i.e. $d^*\mathbf{w} = d\alpha_i\mathbf{x}_i$. Note that the rotation is about the axis through the origin o_i, then the relative translation of o_i with respect to the (i-1)-th frame equals $-d\alpha_i\mathbf{x}_i \times \mathbf{p}_{i-1}^i$, therefore $d^*\mathbf{p}_{i-1} = -d\alpha_i\mathbf{x}_i \times \mathbf{p}_{i-1}$. Using the above relations in Equation 3.14 and 3.15, we have:

$$d\mathbf{p} = d\mathbf{p}^{i-1} + d\mathbf{w}^{i-1} \times \mathbf{p}_{i-1} + d\alpha_i\mathbf{x}_i \times (\mathbf{p}_{i-1} - \mathbf{p}_{i-1}^i) \tag{3.26}$$

$$d\mathbf{w} = d\mathbf{w}^{i-1} + d\alpha_i\mathbf{x}_i \tag{3.27}$$

Differentiating Equation 3.26 and 3.27 with respect to α_i, noting that $d\mathbf{p}_{i-1}$ and $d\mathbf{w}_{i-1}$ are independent on $d\alpha_i$, their derivatives with respect to α_i equal zero; $(\mathbf{p}_{i-1} - \mathbf{p}_{i-1}^i) = \mathbf{p}_i$; and that the derivative of vectors $\mathbf{p}_{i-1}$ and $\mathbf{p}_i$ with respect to α_i are equal to zeros due to that the end-effector is rigidly attached to the i-th frame, we have:

$$\frac{\partial \mathbf{p}}{\partial \alpha_i} = \mathbf{x}_i \times \mathbf{p}_i \tag{3.28}$$

$$\frac{\partial \mathbf{w}}{\partial \alpha_i} = \mathbf{x}_i \tag{3.29}$$

From Equations 3.8 and 3.9, we have the basic structure of the special Jacobian matrix as shown by the Equations 3.18-19; 3.22-23; 3.24-25; and 3.28-29. Following

Equation 3.7, the relationship between position and rotation error of end-effector frame and its individual link parameter errors can be expressed as follows:

$$\Delta \mathbf{x} = \begin{bmatrix} \Delta \mathbf{p} \\ \Delta \mathbf{w} \end{bmatrix} = \sum_{i=1}^{n} \Delta a_i \begin{bmatrix} \mathbf{x}_i \\ \mathbf{0} \end{bmatrix} + \Delta d_i \begin{bmatrix} \mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix} + \Delta \alpha_i \begin{bmatrix} \mathbf{x}_i \times \mathbf{p}_i \\ \mathbf{x}_i \end{bmatrix} + \Delta \theta_i \begin{bmatrix} \mathbf{z}_{i-1} \times \mathbf{p}_{i-1} \\ \mathbf{z}_{i-1} \end{bmatrix} \tag{3.30}$$

Comparing Equation 3.30 with Equation 3.7, the special Jacobian matrix **J** can also be written in the partitioned form whose i-th columns are:

$$(\mathbf{J}_a)_i = \begin{bmatrix} \mathbf{x}_i \\ \mathbf{0} \end{bmatrix}, (\mathbf{J}_d)_i = \begin{bmatrix} \mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix}, (\mathbf{J}_\alpha)_i = \begin{bmatrix} \mathbf{x}_i \times \mathbf{p}_i \\ \mathbf{x}_i \end{bmatrix}, (\mathbf{J}_\theta)_i = \begin{bmatrix} \mathbf{z}_{i-1} \times \mathbf{p}_{i-1} \\ \mathbf{z}_{i-1} \end{bmatrix} \tag{3.31}$$

where $\mathbf{J}_a = \dfrac{\partial \mathbf{f}}{\partial \mathbf{a}}$, $\mathbf{J}_d = \dfrac{\partial \mathbf{f}}{\partial \mathbf{d}}$, $\mathbf{J}_\alpha = \dfrac{\partial \mathbf{f}}{\partial \alpha}$, $\mathbf{J}_\theta = \dfrac{\partial \mathbf{f}}{\partial \theta}$ are the Jacobian matrices with regard to the specific parameters. The vectors $\mathbf{x}_i$, $\mathbf{y}_i$, $\mathbf{z}_i$ are x, y z axis of the frame i associated with the homogenous transformation $\mathbf{T}_i$ which can be computed recursively using Equation 3.1. And the $\mathbf{p}_i$ is the vector connecting the origin of end-effector frame and the origin of the i-th frame with respect to the base frame, which is determined using the following recursive procedure:

$$\mathbf{p}_n = \mathbf{0} \tag{3.32}$$

$$\mathbf{p}_{i-1} = \mathbf{p}_i + d_i \mathbf{x}_i + a_i \mathbf{z}_{i-1}, \quad i = n, n\text{-}1, \ldots 1, \tag{3.33}$$

The ordinary Jacobian matrix ($\mathbf{J}_\theta$) and special Jacobian matrix (**J**) for Puma 560 robot were implemented using MATLAB M-files which are listed in Appendix 3 and 4. From Equation 3.30 and 3.31, we see that robot end-effector inaccuracy vector is a linear combination of the column vectors of the special Jacobian matrix. It is straightforward from linear algebra (Landesman and Hestenes, 1992) that the Jacobian matrix is singular (not full rank) if its column vectors are linearly related. If the Jacobian matrix is singular then the kinematic parameter errors (coefficients in Equation 3.30) can't be uniquely determined in Equation 3.30 and those parameters are defined as unidentifiable. The strict definition and proof of parameter identifiability is given by Bennet and Hollerbach (1991). From the conditions of matrix singularity and Equation 3.31, the conditions for parameter identifiability can be described by the following theorem:

Theorem 3.1 (*Identifiability*): Kinematic parameters are unidentifiable if and only if there exists constants c_i and k_i, not all zero, such that

$$\sum_{i=1}^{n} c_i \mathbf{z}_{i-1} + k_i \mathbf{x}_i = \mathbf{0} \tag{3.34}$$

for all configurations (Bennet and Hollerbach, 1991).

From Equation 3.34, the singularity of the D-H model with two consecutive parallel joint axes can be easily identified. Two consecutive parallel joint axes imply

$$\mathbf{z}_i - \mathbf{z}_{i-1} = \mathbf{0} \tag{3.35}$$

Thus Equation 3.34 is true. The geometric interpretation of this singularity is that there is no unique common normal for parallel joint axes, therefore the link parameter d_i and d_{i+1} can not be uniquely identified.

The inherent D-H model singularity for two consecutive joint axes can be avoided using a modified D-H convention originally proposed by Hayati (1983). The modified model uses an extra rotation parameter around the y joint axis to avoid the use of common normal for parallel joint axes. Post-multiplying Equation 3.3 a rotation transformation **Rot**(**x**, β_i), the Homogenous transformation $\mathbf{A}_i$ in Equation 3.4 becomes:

$$\mathbf{A}_i = \begin{bmatrix} \cos\theta_i \cos\beta_i - \sin\theta_i \sin\alpha_i \sin\beta_i & -\sin\theta_i \cos\alpha_i & \cos\theta_i \sin\beta_i + \sin\theta_i \sin\alpha_i \cos\beta_i & a_i \cos\theta_i \\ \sin\theta_i \cos\beta_i + \cos\theta_i \sin\alpha_i \sin\beta_i & \cos\theta_i \cos\alpha_i & \sin\theta_i \sin\beta_i - \cos\theta_i \sin\alpha_i \cos\beta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i \cos\beta_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.36}$$

It is known that only four parameters are required to specify a co-ordinate frame from the previous frame. When consecutive axes are not parallel, the value of β_i is defined to be zero, while for the case when consecutive axes are parallel, d_i is the variable chosen to be zero. Similar to the derivation of Equation 3.29, the special Jacobian matrix with regard to the parameter β written in its column vector form is:

$$(\mathbf{J}_\beta)_i = \begin{bmatrix} \mathbf{y}_i \times \mathbf{p}_i \\ \mathbf{y}_i \end{bmatrix} \tag{3.37}$$

If the axes of joints (i-1) and i are parallel, then $(\mathbf{J}_\beta)_i$ is used to replace the $(\mathbf{J}_d)_i$ to formulate the special Jacobian matrix.

3.4 Chapter Summary

The D-H modelling technique was reviewed for robot kinematics modelling. The special Jacobian matrix was derived using a geometric approach based on the theory of rigid body movement with rotating frame. The geometric approach is more straightforward than analytic derivation since it has explicit physical interpretation. The linear error model based on robot kinematics is then formulated. The inherent model singularity for the standard D-H convention was discussed, and a modified D-H notation to overcome the singularity was introduced. All the models derived in this chapter have been implemented and validated using the experimental data from a six DoF Puma robot. This chapter serves as the basis for kinematic model-based calibration presented in the following chapters.

CHAPTER 4

KINEMATIC IDENTIFICATION USING RECURRENT NEURAL NETWORK PROCESSING

4.1 Introduction

Kinematic identification is the process of estimating the set of model parameters by minimising the deviations between the poses (positions and orientations) computed by the theoretic model and the measured poses. Therefore three basic ingredients are required for the system identification problem; a mathematical model, measured data and the set of variables that needs to be estimated. A robot kinematic model based on the D-H model convention has been introduced in Chapter 3. Using the vector formulation as Equation 3.5, the l-th robot end-effector location vector $\mathbf{x}_l$ is represented as a non-linear function of the kinematic parameter vector:

$$\mathbf{x}_l = \mathbf{f}(\rho) \tag{4.1}$$

where $\rho = [\mathbf{a}, \mathbf{d}, \alpha, \theta_l]^T$ and $\mathbf{a}, \mathbf{d}, \alpha$ represent manipulator link length, link offset, and twist angle respectively, θ_l is the l-th joint variable which is changing from one configuration to another, $l = 1, 2, ..., M$, and M is the number of robot configurations. The computed vectors of Equation 4.1 are then compared with the actual measured pose vectors $\mathbf{x}_l^a$ to obtain the workspace inaccuracy vector:

$$\Delta\mathbf{x}_l(\Delta\rho) = \mathbf{x}_l^a - \mathbf{f}(\rho^0 + \Delta\rho) \tag{4.2}$$

$\Delta \mathbf{x}_l$ is the non-linear function of the kinematic errors $\Delta\rho = [\Delta\mathbf{a}, \Delta\mathbf{d}, \Delta\alpha, \Delta\theta]^T$ and ρ^0 is the nominal kinematic parameter. The kinematic errors are then identified by a non-linear optimisation procedure which minimises the workspace inaccuracy vector in the least mean square sense:

$$\min_{\Delta\rho} \sum_{l=1}^{M} [\Delta \mathbf{x}_l(\Delta\rho)]^T \mathbf{Q} [\Delta \mathbf{x}_l(\Delta\rho)] \tag{4.3}$$

where $\mathbf{Q}$ is a positive diagonal weight matrix used to adjust the balance between the end-effector position and orientation accuracy. By linearizing the inaccuracy model (4.2) using the first order Taylor series around the nominal kinematic parameters ρ^0, the inaccuracy vector is then represented as a linear function of kinematic errors:

$$\Delta \mathbf{x}_l = \mathbf{J}_l \Delta\rho \tag{4.4}$$

where $\mathbf{J}_l = [\frac{\partial \mathbf{f}}{\partial \mathbf{a}}, \frac{\partial \mathbf{f}}{\partial \mathbf{d}}, \frac{\partial \mathbf{f}}{\partial \alpha}, \frac{\partial \mathbf{f}}{\partial \theta}]$ is a special Jacobian matrix relating the differential changes in kinematic errors to the changes in workspace, which was derived in Chapter 3. The kinematic errors are then identified iteratively using a linear least square method:

$$\Delta \mathbf{x} = \mathbf{J} \Delta\rho \tag{4.5}$$

$$\Delta\rho = [\mathbf{J}^T\mathbf{J}]^{-1}\mathbf{J}^T\Delta\mathbf{x} \tag{4.6}$$

where $\Delta\mathbf{x} = [\Delta\mathbf{x}_1\ \Delta\mathbf{x}_2\ \ldots\ \Delta\mathbf{x}_M]^T$ is the aggregated workspace inaccuracy vector at different robot configurations, $\mathbf{J} = [\mathbf{J}_1\ \mathbf{J}_2\ \ldots\mathbf{J}_M]^T$ is the aggregated special Jacobian matrix, M is the number of robot configurations at which measurement data is collected.

Both linear and non-linear least square optimisation methods as described above have been commonly used to identify geometric and non-geometric errors. To make the identification accurate and fast, many issues such as the choice of algorithms, selection of measurement points and the number of measurements have been addressed by other researchers (Pathre and Driels, 1990; Driels, Swayze and Potter, 1993). Since kinematic identification is a highly complex numerical problem which involves a large number of variables to be estimated, normally the standard optimisation methods require a long convergence time, and the measurement points must be distributed widely in the workspace to ensure the numerical stability. These identification algorithms are usable

for robot off-line calibration performed in a laboratory environment where computation time is not critical and there are no physical constraints on data collection. It is not the case for robot on-site calibration performed in an industrial environment, which may impose severe limitations on data collection and identification capabilities. In this chapter, a Hopfield type recurrent neural network (RNN) based algorithm has been developed for the robot kinematic identification problem. The RNN-based algorithm is computationally more efficient and robust compared with the numerical optimisation algorithms and therefore is suitable for robot on-site calibration processing. Calibration results for a six DoF Puma 560 robot are presented using the new calibration processing method.

4.2 Hopfield Recurrent Neural Network

A Hopfield net is a neural network composed of a layer of fully interconnected artificial neurons. Each neuron of the network is connected to every other neuron in the network. The architecture of the Hopfield network differs significantly from the feedforward network in that it belongs to the class of recurrent (or feedback) neural networks in which dynamics play an important role. The dynamics of such networks are described by a system of non-linear ordinary differential equations and by an associated energy (called the Lyapunov, potential, or simply network energy) function which is minimised during the computation process. Hence the Hopfield network is dynamic by nature and is categorised as a searching (or optimisation) type of network.

There are two types of the Hopfield model: binary model and continuous model (Hopfield and Tank, 1985). The continuous model of the Hopfield network is analogous to an analogue circuit model of an operational amplifier with resistive connections and additional capacitors (Figure 4.1). The dynamics of the neuron circuit is governed by the following non-linear ordinary differential equations:

$$I_i + \sum_{i=1}^{n} T_{ij} v_j = \frac{u_i}{R_i} + c_i \frac{du_i}{dt} \tag{4.7}$$

$$v_i = g_i(u_i) \tag{4.8}$$

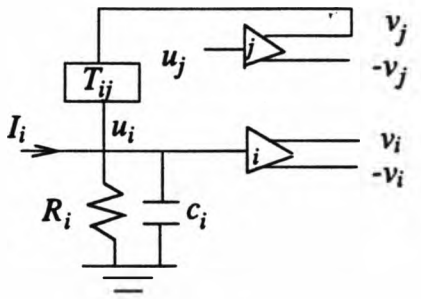


Figure 4.1. Hopfield Neuron Circuit

where I_i is input current of the i-th neuron (amplifier), u_i is the input voltage and v_i is the output voltage; $g_i = g(x) = \tanh(\beta x)$ is the output function which is usually taken to be related to the hyperbolic tangent function[1], β is the amplifier gain constant. $T_{ij} = 1/R_{ij}$, R_{ij} is the connection resister between amplifier i and j, c_i is the input capacitor. Figure 4.2 depicts the Hopfield network which consists of n fully interconnected artificial neurons described in Figure 4.1. Hopfield (1985) discovered a Lyapunov function for a network of n neurons characterised by Equation 4.7 and 4.8 which measures the total energy represented by the network with respect to the network outputs:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij} v_i v_j - \sum_{i=1}^{n} I_i v_i + \sum_{i=1}^{n} \frac{1}{R_i} \int_0^{v_i} g_i^{-1}(v)dv \qquad (4.9)$$

when the gain of the monatomic increasing activation function is sufficiently high, the last term of the Energy function can be ignored. It can be shown that the changes in the network output governed by the network dynamics such as Equation 4.7 and 4.8 will always cause the network to evolve toward a minimum of the energy function. The stable states of the network therefore correspond to the local minima of the energy function.

[1] It is known that the shape of the output function (or activation function) is not important to guarantee the network convergence as long as it is monotonically increasing with high positive gain constant. More often than not, the output function is simplified as a linear function with high gain constant.

Because the network of neurons will seek to minimise the energy function, one may design a Hopfield type neural network for function minimisation by associating variables in an optimisation problem with variables in the energy function. Developing a neural network to seek solutions to an optimisation problem becomes the task of selecting appropriate values for the connection strengths T_{ij} and the external inputs I_i so that the desired network behaviour results. Hopfield and Tank (1985, 1986) have illustrated the use of energy functions by configuring the network to find good solutions to difficult optimisation problems such as the travelling salesman problem which is of the *np*-complete class. The Hopfield type simple "neural" optimisation networks have been used (Tank and Hopfield, 1986) to find globally optimal solutions for a class of less complicated optimisation problems such as A/D converter and a linear programming problem which have no local minima in their solution spaces (in the vicinity of specific initial conditions). Since kinematic errors are only small perturbations from the nominal parameter values, we show in the following sections how the kinematic identification can be solved rapidly by using the Hopfield recurrent neural network.

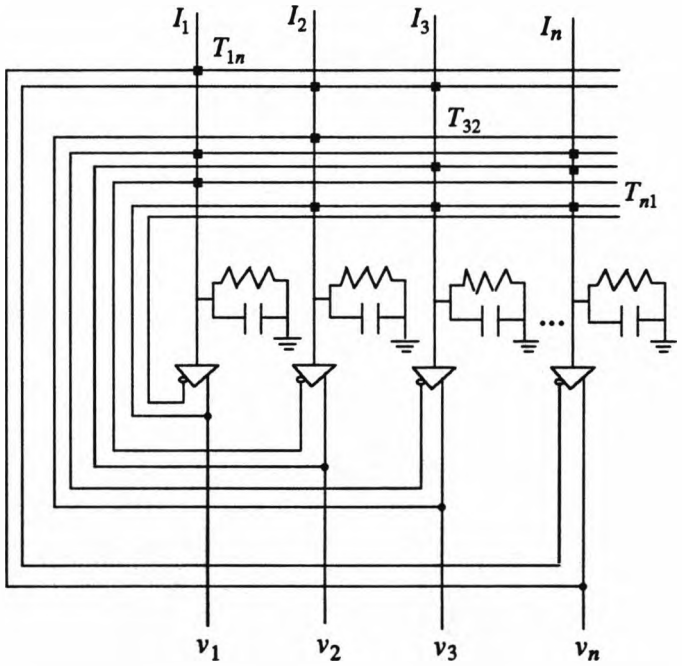


Figure 4.2. Hopfield analogue neural circuit model (Hopfield and Tank, 1986). Black square at intersections represent resistive (weight) connections (with conductance T_{ij}).

4.3 RNN-based Kinematic Identification Algorithm

The key to the application of the Hopfield type neural networks is the formulation of the network energy function. In our application, the quadratic form of robot workspace inaccuracy is constructed as a network energy function so that a decrease of network energy corresponds to a decrease of robot positioning inaccuracy in workspace. The kinematic parameter errors are chosen as the neural circuit variables. From Equation 4.4, the linear residual error model for M measurement points, writing in the compact matrix form, is:

$$\mathbf{e}(\Delta\rho) = \Delta\mathbf{x}^0 - \mathbf{J}\Delta\rho \tag{4.10}$$

where $\mathbf{e}$ is the residual positioning error vector in workspace which is the linear function of kinematic error $\Delta\rho$, $\Delta\mathbf{x}^0$ is the initial value of the aggregated inaccuracy vector and $\mathbf{J}$ is the aggregated special Jacobian matrix as defined above. Kinematic identification is equivalent to determining the kinematic error vector by minimising the residual positioning error vector in the least square sense. Using the Euclidean norm, the energy function then can be constructed as follows:

$$E = \frac{1}{2}\left[\Delta\mathbf{x}^0 - \mathbf{J}\Delta\rho\right]^T\left[\Delta\mathbf{x}^0 - \mathbf{J}\Delta\rho\right] \tag{4.11}$$

Expanding Equation 4.11, and rearranging it in the standard form of the network energy, we have:

$$E = -\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}\Delta\rho_i\Delta\rho_j - \sum_{j=1}^{n} I_j\Delta\rho_j + \tfrac{1}{2}\sum_{i=1}^{m}(\Delta x_i^0)^2 \tag{4.12}$$

where

$$T_{ij} = -\sum_{s=1}^{m} J_{si}J_{sj} \tag{4.13}$$

$$I_j = \sum_{i=1}^{m} J_{ij}\Delta x_i^0 \tag{4.14}$$

T_{ij} and I_j determine the network connection weights and the input currents respectively based on the nominal kinematic model and the workspace inaccuracy vector. The $\Delta\rho_j$ is

the j-th kinematic parameter error to be identified which corresponds to the j-th neuron state. n is the number of kinematic parameters while m is the dimension of vector $\Delta \mathbf{x}^0$.

The energy function gradient with respect to the i-th neuron state is obtained from Equation 4.12:

$$\frac{\partial E}{\partial(\Delta\rho_i)} = -(\sum_{j=1}^{n} T_{ij}\Delta\rho_j + I_i) \tag{4.15}$$

The time evolution of neuron states should follow the opposite direction of the energy function gradient, so we have the neuron circuit dynamics equation:

$$\frac{d(\Delta\rho_i)}{dt} = \mu_i(\sum_{j=1}^{n} T_{ij}\Delta\rho_j + I_i) \tag{4.16}$$

where μ_i is the i-th diagonal element of the positive diagonal coefficient matrix (or learning rate) μ which is chosen to ensure the stability and convergence speed of the neural circuit. Given the initial condition of the neuron states ($\Delta\rho_i = 0$, $i = 1, ..., n$), the above ordinary differential equation (ODE) determines the neuron state trajectories, and hence the kinematic errors to be identified (the stable states of the neurons). Equation 4.16 is actually a linear differential equation of high gain which can be solved by any standard ODE methods such as Runge-Kutta method, Euler's method, etc (MathWorks Inc., 1992b). To improve the robustness against measurement noise and numerical perturbations, non-linearity can also be incorporated into Equation 4.16 through the introduction of the neuron "sigmoid" to the network as in Equation 4.7 and 4.8. The linear neuron processing units were used here for simplicity.

From Equation 4.15 and 4.16, the time derivative of the energy function is derived as Equation 4.17 which is always non-positive. Therefore the above algorithm is guaranteed to converge to a lower energy level.

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial(\Delta\rho_i)} \frac{d(\Delta\rho_i)}{dt} = \sum_{i=1}^{n} -\mu_i(\sum_{j=1}^{n} T_{ij}\Delta\rho_j + I_i)^2 \tag{4.17}$$

The convergence speed of the Equation 4.17 depends on the choice of the learning rate μ which is shown as follows. Using the definition of the Equation 4.13 and 4.14, the dynamic system Equation 4.16 can be written in compact matrix form:

$$\frac{d(\Delta\rho)}{dt} = \mu(-\mathbf{J}^T\mathbf{J}\Delta\rho + \mathbf{J}^T\Delta\mathbf{x}^0) = -\mu\mathbf{J}^T(\Delta\mathbf{x} - \Delta\mathbf{x}^0) \tag{4.18}$$

where $\Delta\mathbf{x} = \mathbf{J}\Delta\rho$. Using Equation 4.18 in Equation 4.17, we have:

$$\frac{dE}{dt} = -(\Delta\mathbf{x} - \Delta\mathbf{x}^0)^T\mathbf{J}\mu\mathbf{J}^T(\Delta\mathbf{x} - \Delta\mathbf{x}^0) \tag{4.19}$$

Theorem 4.1. Let λ_J and λ_μ be the minimum eigenvalue of symmetric matrix $\mathbf{JJ}^T$ and positive diagonal matrix μ respectively, then the time derivative of the energy function Equation 4.19 satisfies the following inequality:

$$\frac{dE}{dt} \le -\lambda_\mu\lambda_J \left\|\Delta\mathbf{x} - \Delta\mathbf{x}^0\right\|^2 \tag{4.20}$$

where $\|\cdot\|$ is a Euclidean norm.

Proof:

We introduce the Rayleigh quotient (Landesman and Hestenes, 1992):

$$R(\mathbf{v}) = \frac{\mathbf{v}^T\mathbf{A}\mathbf{v}}{\mathbf{v}^T\mathbf{v}} \tag{4.21}$$

where $\mathbf{v}$ is any vector excluding $\mathbf{v} = \mathbf{0}$, $\mathbf{A}$ is a symmetric matrix. The Rayleigh quotient satisfies that

$$\lambda_{\min} \le R(\mathbf{v}) \le \lambda_{\max} \tag{4.22}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ is the minimum and maximum eigenvalues of $\mathbf{A}$.

The following inequality is obtained by setting $\mathbf{A} = \mu$, $\mathbf{v} = \mathbf{J}^T(\Delta\mathbf{x}-\Delta\mathbf{x}^0)$ in the expression of $R(\mathbf{v})$ and using inequality 4.22:

$$(\Delta\mathbf{x} - \Delta\mathbf{x}^0)^T\mathbf{J}\mu\mathbf{J}^T(\Delta\mathbf{x} - \Delta\mathbf{x}^0) \ge \lambda_\mu(\Delta\mathbf{x} - \Delta\mathbf{x}^0)^T\mathbf{J}\mathbf{J}^T(\Delta\mathbf{x} - \Delta\mathbf{x}^0) \tag{4.23}$$

In Equation 4.21, let $\mathbf{A} = \mathbf{JJ}^T$, $\mathbf{v} = (\Delta\mathbf{x}-\Delta\mathbf{x}^0)$ and using 4.22, then we have:

$$(\Delta\mathbf{x} - \Delta\mathbf{x}^0)^T\mathbf{J}\mathbf{J}^T(\Delta\mathbf{x} - \Delta\mathbf{x}^0) \ge \lambda_J(\Delta\mathbf{x} - \Delta\mathbf{x}^0)^T(\Delta\mathbf{x} - \Delta\mathbf{x}^0) = \lambda_J\left\|\Delta\mathbf{x} - \Delta\mathbf{x}^0\right\|^2 \tag{4.24}$$

From 4.23 and 4.24, and noting that μ is positive definite thus λ_μ is positive, we have:

$$(\Delta\mathbf{x} - \Delta\mathbf{x}^0)^T\mathbf{J}\mu\mathbf{J}^T(\Delta\mathbf{x} - \Delta\mathbf{x}^0) \ge \lambda_\mu\lambda_J\left\|(\Delta\mathbf{x} - \Delta\mathbf{x}^0)\right\|^2 \tag{4.25}$$

From 4.19 and 4.25, it is straightforward to have 4.20. The proof is completed.

From Theorem 4.1 we can see that

• the system has a quadratic convergence speed in terms of the residual error vector in workspace ($\Delta \mathbf{x} - \Delta \mathbf{x}^0$).

• by increasing λ_μ, we can achieve very fast convergence of the energy function to its minimum. If the learning rate μ is set to be a constant k,[2] then the energy function is linearly decreasing in time with the scalar gain of k. This observation agrees that in Cichocki and Unbehauen (1993).

Comparing the discrete-time steepest-descent algorithm in which the controlling parameter μ should be bounded in a small range to ensure the stability of the algorithm, in the continuous-time system the learning rate μ can be set theoretically arbitrarily large without affecting the stability of the algorithm (Cichocki and Unbehauen, 1993). It is an important advantage of using the continuous-time neural system which will be shown through numerical examples in the following sections. Another advantage of the RNN-based method is the potential implementation of parallel computation which leads to on-line identification of kinematic parameters. It only takes a few characteristic time constants of the neuron circuit and is independent of the robot DoF and the number of parameters to be identified.

After kinematic errors have been obtained through Equation 4.16, the kinematic parameters are updated in the forward kinematic model and the neuron inputs and weight connections are updated correspondingly. The above procedure is repeated until the algorithm converges to the prescribed accuracy. The Hopfield type recurrent neural network can converge to the global optimal given the specific initial conditions which are in the vicinity of its optimal solutions (Tank and Hopfield, 1986). Since the kinematic errors are only small perturbations from the nominal parameters, the above procedures are guaranteed to converge to the global optimal provided the kinematic errors are small. Normally three iterations are sufficient to get the desirable results.

---

[2] All diagonal elements of the diagonal matrix μ are equal to k, then $\lambda_\mu = k$.

4.4 Pose Measurement Using a CMM

The experimental set-up consists of a Ferranti Merlin 750 precision co-ordinate measuring machine (CMM) and a Puma 560 robot as illustrated in Figure 4.3. A local calibration volume is chosen in the common working volume of the CMM and the robot. The calibration volume is a parallelepiped measuring 200 (mm) in width, 400 (mm) in length, and 200 (mm) in height. This is justified by the fact that high positioning accuracy is only required in the robot local working area in which fine motion operations (like assembly) are executed. The measurement points are uniformly distributed in the calibration volume with four points in length, and three points each in the width and height dimension. The measurement grid ($3 \times 4 \times 3$) for data acquisition is shown in Figure 4.4. Eight different orientations of the end-effector were placed at each measuring point so that a range of different configurations were measured. The total number of measurements thus is 288 ($= 3 \times 4 \times 3 \times 8$). This data set is used for robot calibration and evaluation.

Figure 4.3. Experimental Set-up for Data Collection

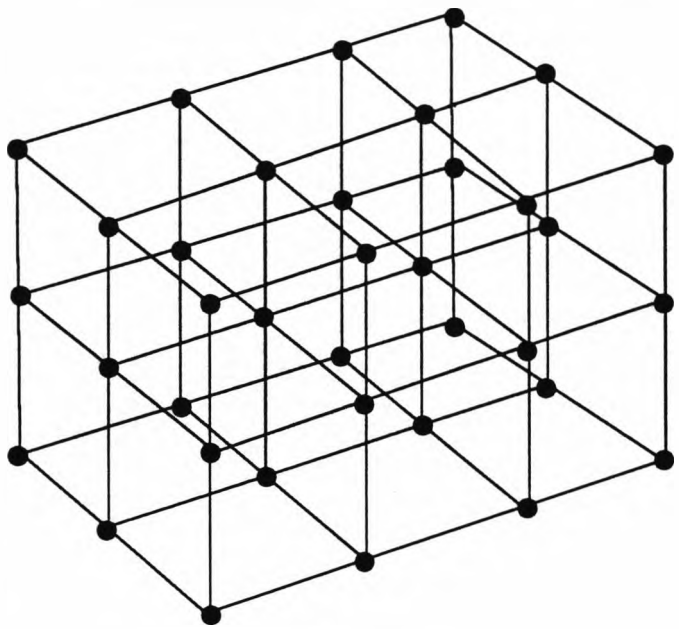


Figure 4.4. Measurement grid for data acquisition

The schematic of the measurement set-up is shown in Figure 4.5. The CMM has an accuracy of 4 μm which is accurate enough for robot calibration. Measurements are made by manually moving the CMM one axis at a time until the touch probe mounted on the CMM contacts an object to be measured. When the touch probe is triggered, the x, y and z co-ordinates of the tip point of the probe are recorded and transferred to the personal computer by the CMM controller. The Amstrad personal computer is used to record the manipulator configuration, collect the CMM data, and perform some data processing. The robot end-effector is equipped with a measuring cube so that the end-effector position and orientation can be obtained from the positions of the touch points on the cube. A more detailed description of the CMM measuring system can be found in Rea (1992). Pose measuring using CMM is very time consuming and took skilled operators several days work to collect the data required. The high cost of the CMM and its bulky volume preclude its use in robot on-site calibration in a shop-floor environment. A novel approach suitable for on-site application is developed in the next chapter. However, the precise full pose data collected using the CMM is very useful for the purpose of proof of concept calibration and the calibration evaluation. Below we give the principles of obtaining robot end-effector position and orientation inaccuracy data from the positions of the CMM probe touch points.

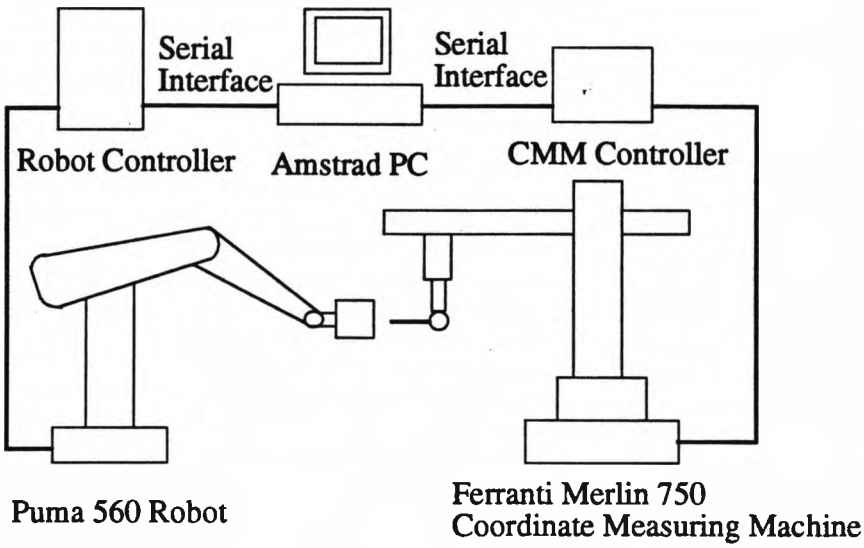


Figure 4.5. Schematic of measurement set-up

The robot end-effector consists of a calibrated measuring cube with a dimension $L \times L \times L$. and its co-ordinate system $X_eY_eZ_e$ is shown in Figure 4.6. The position and orientation of the measuring cube can be described by a homogenous transformation matrix as introduced in Chapter 3. The homogenous transformation of the measuring cube in the world co-ordinate system is determined if the position vector of the cube centre point $\mathbf{p}_c$ and three orientation vectors $\mathbf{n}_c$, $\mathbf{s}_c$ and $\mathbf{o}_c$ are determined in the world co-ordinate system. The world co-ordinates of the touch points on the measuring cube are estimated and reported by the CMM. Let the position vectors of the touch points be $\mathbf{r}_{ni}$, $\mathbf{r}_{si}$,$\mathbf{r}_{oi}$,where the *n, s, o* subscripts represent the touch surface of the cube and *i* denotes the number of the touch points. Clearly at least three different touch points are needed to determine the normal vector of the side surface of the cube. Therefore three touch points were measured on each of the three side surface of the cube to determine the position and orientation of the cube:

$$\mathbf{n}_c = \frac{(\mathbf{r}_{n3} - \mathbf{r}_{n2}) \times (\mathbf{r}_{n2} - \mathbf{r}_{n1})}{\left\| (\mathbf{r}_{n3} - \mathbf{r}_{n2}) \times (\mathbf{r}_{n2} - \mathbf{r}_{n1}) \right\|} \tag{4.26}$$

$$\mathbf{s}_c = \frac{(\mathbf{r}_{s3} - \mathbf{r}_{s2}) \times (\mathbf{r}_{s2} - \mathbf{r}_{s1})}{\left\| (\mathbf{r}_{s3} - \mathbf{r}_{s2}) \times (\mathbf{r}_{s2} - \mathbf{r}_{s1}) \right\|} \tag{4.27}$$

$$\mathbf{o}_c = \frac{(\mathbf{r}_{o3} - \mathbf{r}_{o2}) \times (\mathbf{r}_{o2} - \mathbf{r}_{o1})}{\left\| (\mathbf{r}_{o3} - \mathbf{r}_{o2}) \times (\mathbf{r}_{o2} - \mathbf{r}_{o1}) \right\|} \tag{4.28}$$

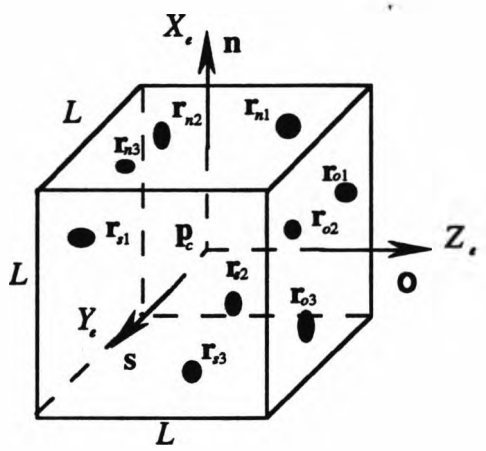


Figure 4.6. The end-effector (measuring cube) co-ordinate system

The orientation vectors must satisfy the orthogonal condition $\mathbf{o}_c = \mathbf{n}_c \times \mathbf{s}_c$. Noting that the projection of the difference vector between touch point and the cube centre point in the direction of the surface normal equals the half side length of the cube (L), we have the following scalar equations which are used to determine the position vector of the cube centre point:

$$(\mathbf{r}_{ni} - \mathbf{p}_c) \,.\, \mathbf{n}_c = 0.5L \tag{4.29}$$

$$(\mathbf{r}_{si} - \mathbf{p}_c) \,.\, \mathbf{s}_c = 0.5L \tag{4.30}$$

$$(\mathbf{r}_{oi} - \mathbf{p}_c) \,.\, \mathbf{o}_c = 0.5L \tag{4.31}$$

where $i = 1, 2, 3$. The position vector $\mathbf{p}_c = [p_{cx}, p_{cy}, p_{cz}]^T$ has three unknowns therefore is uniquely determined by the three scalar equations above. More scalar equations (more touch points) can be used to determine the position vector in the least square sense. After the position and orientation vectors are determined, the actual measured end-effector homogenous transformation matrix $\mathbf{T}_a$ is obtained:

$$\mathbf{T}_a = \begin{bmatrix} \mathbf{n}_c \, \mathbf{s}_c \, \mathbf{o}_c \, \mathbf{p}_c \\ 0 \; 0 \; 0 \; 1 \end{bmatrix} \tag{4.32}$$

Comparing with the nominal commanded end-effector transformation $\mathbf{T}_n$, the additive differential transformation is:

$$\mathbf{dT} = \mathbf{T}_a - \mathbf{T}_n, \tag{4.33}$$

Then the multiplicative differential transformation is (Paul, 1982):

$$\Delta\mathbf{T} = d\mathbf{T} \cdot \mathbf{T}_a^{-1} \tag{4.34}$$

Ideally, $\Delta\mathbf{T}$ has the structure of an upper 3 X 3 skew-symmetric matrix:

$$\Delta\mathbf{T} = \begin{bmatrix} 0 & -\delta z & \delta y & dx \\ \delta z & 0 & -\delta x & dy \\ -\delta y & \delta x & 0 & dz \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.35}$$

In real cases, the actual $\Delta\mathbf{T}$ obtained has the following general form due to numerical errors.

$$\Delta\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.36}$$

Then the elements of the Equation 4.35 can be obtained through the following procedure:

$$\delta x = \frac{1}{2}(t_{32} - t_{23}) \tag{4.37}$$

$$\delta y = \frac{1}{2}(t_{13} - t_{31}) \tag{4.38}$$

$$\delta z = \frac{1}{2}(t_{21} - t_{12}) \tag{4.39}$$

$$dx = t_{14} \tag{4.40}$$

$$dy = t_{24} \tag{4.41}$$

$$dz = t_{34} \tag{4.42}$$

The position and orientation inaccuracy vector is then subtracted from Equation 4.35:

$$d\mathbf{x} = \begin{bmatrix} d\mathbf{p} \\ d\mathbf{w} \end{bmatrix} = \begin{bmatrix} (dx, dy, dz)^T \\ (\delta x, \delta y, \delta z)^T \end{bmatrix} \tag{4.43}$$

The postional and orientational inaccuracy, representing in length (Euclidean norm), are:

$$dr = \|d\mathbf{p}\| = \sqrt{dx^2 + dy^2 + dz^2} \tag{4.44}$$

$$dtr = \|d\mathbf{w}\| = \sqrt{\delta x^2 + \delta y^2 + \delta z^2} \tag{4.45}$$

4.5 Kinematic Identification Results for a PUMA 560 Robot

The Puma 560 robot is a six DoF manipulator with six revolute joints. The link co-ordinate frame assignment for the Puma robot using the D-H convention is shown in Figure 4.7. Robot end-effector frame transformation represented in the world co-ordinate frame can be obtained through the following sequential homogenous transformations:

$$\mathbf{T}_e = \text{BASE}*\mathbf{A}_1*\mathbf{A}_2*...*\mathbf{A}_6*\text{FLANGE}*\text{TOOL} \tag{4.46}$$

where

1) BASE represents the transformation between the world co-ordinates $\mathbf{x}_w\mathbf{y}_w\mathbf{z}_w$ and the first link co-ordinate frame $\mathbf{x}_0\mathbf{y}_0\mathbf{z}_0$ on joint 1 fixed in the robot base.

2) $\mathbf{A}_i$ (i = 1, 2, .., 6) represents the transformation between joint coordinate frame i and i-1 as defined in Chapter 3.

3) FLANGE represents a transformation between the last joint co-ordinate frame nominally located on the last joint axis to a co-ordinate frame located on the manipulator's flange used for mounting the end-effector or tool (the measuring cube in this case).

4) TOOL is the transformation from the mounting point on the manipulator's FLANGE to the tool frame located on the tool centre point (TCP frame).

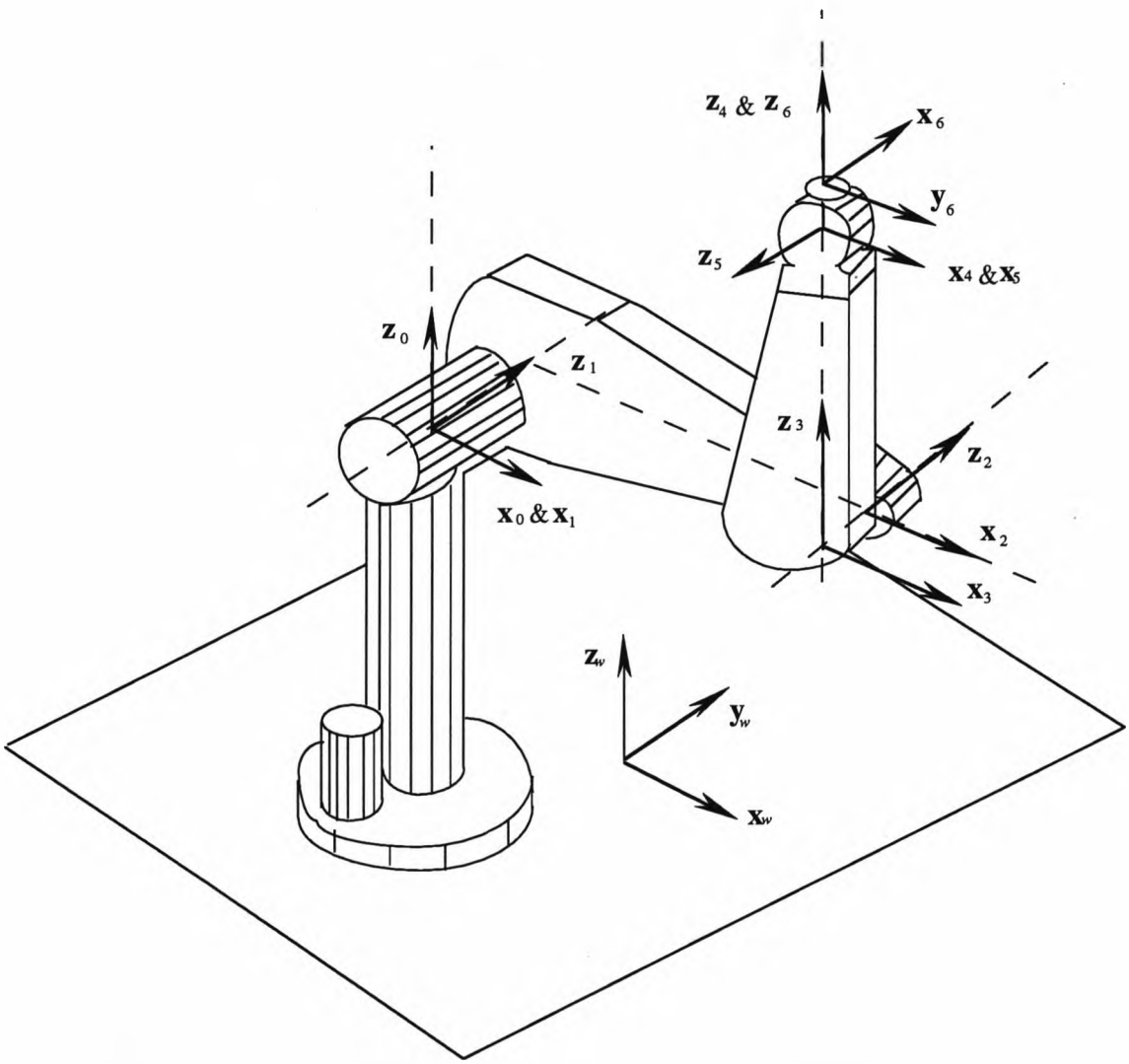


Figure 4.7. Puma Co-ordinate Frame Assignment

The Puma robot is installed in the workspace which is common with the CMM so that its first link frame is aligned with the CMM reference frame (i.e. the frame $\mathbf{x}_w\mathbf{y}_w\mathbf{z}_w$ is aligned with the frame $\mathbf{x}_0\mathbf{y}_0\mathbf{z}_0$). By this choice of frame arrangement, the CMM measurements can be converted to the robot measurements in its base frame by a pure translation transformation **Trans**(x_0,y_0,z_0), where x_0,y_0,z_0 are relative position co-

ordinates between the origins of the world frame and the robot base frame. Since the first link co-ordinate frame, according to the D-H convention, is inside the robot, the alignment of the robot base frame with the CMM is accomplished by using a locating dowel and precision holes in the robot base. We assume that the robot base is exactly located with the world frame therefore no errors exist in the BASE transformation. The effects of the errors in the BASE transformation will be investigated using a modified D-H modelling convention.

Due to parameter redundancy of the $\mathbf{A}_6$, FLANGE and TOOL transformation, the $\mathbf{A}_6$ and FLANGE are normally combined into a single transformation $\mathbf{A}_6$. Since the tool mounted to the flange is a high-precision measuring cube, the TOOL transformation will always be assumed to be error-free during calibration and all the errors in the last link will be compensated using kinematic parameters in $\mathbf{A}_6$. The assumption of an error-free TOOL is made because it is impossible to distinguish the effects of errors in the last link from errors in the TOOL by measuring a reference point located on the TOOL (Veitschegger and Wu, 1988).

Based on the above assumptions that BASE and TOOL transformation are error-free, the causes for robot positioning inaccuracy are basically kinematic parameter errors in $\mathbf{A}_i$ (i = 1, 2, ..., 6). Using the D-H modelling notation, four parameters a_i, d_i, α_i, and θ_i are needed to specify one transformation, therefore there are 24 parameters in total which need to be identified. The nominal values of these parameters are defined as in Table 4.1. Joint variables θ_i are determined by the nominal inverse kinematic model which change from one configuration to another. Another six parameters are involved in TOOL transformation, the first three are rotational parameters (roll-pitch-yaw) and the last three are translation parameters. The standard D-H model has the singularity problem for the consecutive parallel joint axes as introduced in Chapter 3. Since kinematic modelling is not the focus of this chapter, the standard D-H model is used firstly to verify the RNN-based kinematic identification algorithm.

Table 4.1 Nominal Parameters of a Puma 560 Robot Using D-H Model

No.	a_i (mm)	d_i (mm)	α_i (rad.)	TOOL
1	0.0	0.0	$-\pi/2$	0.0 (rad.)
2	431.8	149.09	0.0	0.0 (rad.)
3	-20.32	0.0	$\pi/2$	0.0 (rad.)
4	0.0	433.07	$-\pi/2$	0.0 (mm)
5	0.0	0.0	$\pi/2$	0.0 (mm)
6	0.0	56.25	0.0	55.0 (mm)

After the kinematic model has been established, the actual measurements are used for kinematic identification. One hundred and twenty poses have been randomly chosen from the collected data set for identification processing. The RNN-based kinematic identification algorithm was implemented using MATLAB® (MathWorks Inc., 1992a) on a Hewlet-Packard 9000 workstation. The neural dynamic equations were solved by calling SIMULINK™ (MathWorks Inc., 1992b) ordinary differential equation (ODE) solver. Given the learning rate as a high gain constant $\mu = 10^6$, and the initial conditions of neural states $\Delta\rho = \mathbf{0}$, the kinematic errors (Table 4.2) were identified rapidly (in less than two seconds of simulation time and in about 20 (μs) of the circuit settling time). Figure 4.8 shows the time evolution trajectory of the kinematic parameter errors during the first iteration of the identification process, which exhibits efficient and robust convergence. The identification process stopped after three iterations, when further iterations will not obtain much improvements on the final residual errors. The average position error in length decreased from 3.33 (mm) to 0.70 (mm) for the 120 calibration points, and to 0.68 (mm) for 100 independent test points after updating the kinematic parameters by the identified errors. Table 4.3 compares the residual position and orientation errors before and after kinematic calibration based on the 100 independent test points from the collected data set. The position accuracy has improved after

calibration by a factor of about 5, and the orientation improvement is less significant due to the scaling problem. The balance between the position and orientation residual error can be adjusted by a weight matrix **Q** as defined in Equation 4.3. For most robot applications, position accuracy is more critical than orientation accuracy, therefore position accuracy improvement is emphasised in the following work.

For comparison, the non-linear and linear square optimisation approaches (using Equation 4.3 and 4.6) were also used for identification based on the same data set. After about two hours computation (1200 iterations) using quasi-Newton searching strategy (MathWorks Inc., 1992c), the non-linear optimisation approach converged to results that were identical to the neural net solution, while the linear optimisation approach failed to converge due to the singularity of the identification Jacobian using the standard D-H model. As observed by Cichocki and Unbehauen (1993), the computation efficiency and robustness of the RNN approach compared with numerical optimisation methods are largely due to the use of systems of ordinary differential equations rather than the difference equations as used in conventional optimisations. The advantage of converting an optimisation problem into a system of differential equations are outlined as follows (Cichocki and Unbehauen, 1993):

- Due to the massively parallel operations and due to the better convergence properties in comparison with iterative schemes, the simulation of a system of differential equations enable us to solve many optimisation problems in real time. The better convergence properties of the continuous-time systems are due to the fact that some controlling parameters (learning rates) can be set arbitrarily large without affecting the stability of the system in contrast to discrete-time systems where the corresponding controlling parameters must be bounded in a small range or else the system will be unstable. For example, the learning rate in the above simulation is set as a high gain constant ($\mu = 10^6$) which enables the system to converge in the order of (μs) settling time.

- A dynamic system implemented on the basis of differential equations usually exhibits more robustness (insensitivity) to certain parameter variations and it tends to retain information better through time.

- In the simulation of continuous-time dynamic systems more sophisticated and faster simulation techniques than simple first-order difference equations can be used (Mathworks Inc., 1992b).

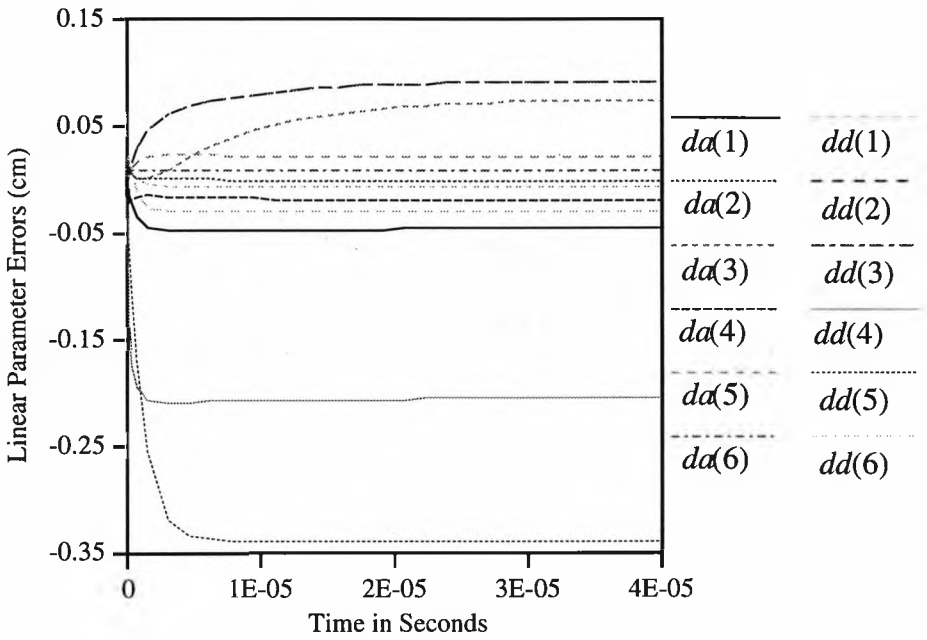
- Theoretical ordinary differential equation techniques often offer better understanding of the convergence conditions of the corresponding iterative algorithms.

Table 4.2. Identified Kinematic Parameter Errors of the PUMA 560 (D-H Model)

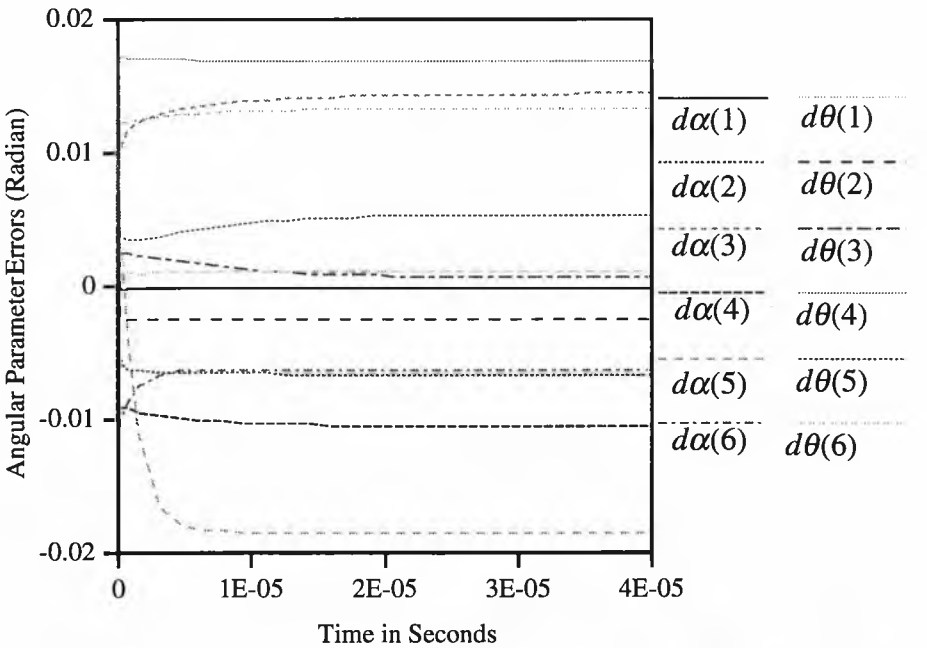
Link No.	Δa_i (mm)	Δd_i (mm)	$\Delta \alpha_i$ (rad.)	$\Delta \theta_i$ (rad.)
1	-0.478	-0.301	-0.0003	0.0009
2	-0.023	0.909	-0.0068	-0.0025
3	0.719	0.909	0.0143	0.0004
4	-0.222	-2.026	-0.0107	0.0167
5	0.204	-3.400	-0.0187	0.0053
6	0.064	-0.041	-0.0067	0.0132

Table 4.3 Residual Error Comparisons Using the Standard D-H model

	Before Calibration		After Calibration	
(in length)	position (mm)	orien. (degree)	position (mm)	orien. (degree)
average	3.30	2.63	0.68	1.19
standard dev.	1.15	0.41	0.46	0.46
maximum	4.74	3.21	1.83	2.30



(a). Linear Parameter Error Identification



(b). Angular Parameter Error Identification

Figure 4.8. Time Evolution of Kinematic Error Identification

The standard D-H model is adopted to perform the concept-proof calibration using the RNN-based kinematic identification algorithm. The D-H model singularity and the assumption of error-free BASE may be attributable to the large residual errors after kinematic calibration as seen from the Table 4.3. As introduced in Chapter 3, the D-H model singularity can be avoided by a modified D-H notation which uses a rotation parameter β to replace the common normal parameter d for the two consecutive parallel joint axes. The joints 2 and 3 of the Puma robot are the consecutive joints with parallel joint axes, therefore the common normal parameter d_2 is replaced by the rotation parameter β_2, and the values of the d_2 and β_i ($i \neq 2$) are set to zero during the identification process. Following Veitschegger and Wu (1988), robot BASE parameters can also be included in the identification by using an introduced transformation. Since the BASE frame is aligned with the world frame, an extra transformation BOFF was introduced to change the relative arrangement of the world frame and robot base frame. From Equation 4.46, we have:

$$\text{BOFF}*\mathbf{T}_e = \text{BOFF}*\text{BASE}*\mathbf{A}_1*\mathbf{A}_2*...*\mathbf{A}_6*\text{FLANGE}*\text{TOOL} \qquad (4.47)$$

where BOFF represents an introduced translation and/or rotation transformation. By selecting BOFF such that the world co-ordinate frame origin lies on the manipulator's base mounting surface, the errors in the transformed BASE (BOFF*BASE) will correspond to the errors within the manipulator's physical base. An extra rotation $\mathbf{Rot}(x, 90^\circ)$ was made so that the $\mathbf{z}_w$ and $\mathbf{z}_0$ are perpendicular to each other. Therefore BOFF = $\mathbf{Trans}(x_0,y_0,z_0)*\mathbf{Rot}(x, 90^\circ)$, where x_0 = -450 (mm), y_0 = -145 (mm), z_0 = -260 (mm) are relative position co-ordinates between the origin points of the world frame and the robot base frame. Through this introduced transformation, the robot BASE (link 0) can be described using the standard four D-H parameters. The nominal parameters of the Puma robot (including the BASE) are redefined in Table 4.4 using the modified D-H model.

Kinematic identification was performed based on the modified D-H model using the RNN-based algorithm. The identified parameter errors are listed in Table 4.5. It is seen that there exists large translation errors in robot base parameters which may not be ignored for kinematic calibration. The residual errors comparison of the end-effector positioning accuracy before and after the kinematic calibration was made in Table 4.6 based on the 100 test points. It shows that the average residual error of robot end-effector position was reduced from 3.3 (mm) before calibration to 0.19 (mm) after

calibration, which indicates an improvement factor of more than 10, comparing the improvement factor of about 5 by using a standard D-H model and ignoring the BASE errors. The achieved positioning accuracy of 0.19 (mm) is of the same order of the Puma robot repeatability which is the limit of the robot calibration. The results show that robot positioning accuracy can be improved significantly through kinematic calibration only. Although the measurements are limited to the local volume in the workspace, hence produced an ill-conditioned identification Jacobian[3], the RNN-based identification algorithm is able to identify the kinematic errors efficiently and accurately. The identification accuracy is further improved by using a modified D-H model. Figure 4.9 plots the residual position error distribution after calibration for the 100 test points. The figure consists of 5 bars that represents the number of points for which the position error fell within the range indicated on the horizontal axis. For example, the first bar from the right shows that only one point out of 100 points had a position error in the range between 0.4 (mm) and 0.5 (mm). The figure illustrate that the position errors of most test points lies around the mean of 0.19 (mm).

It can be seen from Table 4.6 that the residual orientation error is rather large after calibration. The simple identity scaling matrix **Q** used in the calibration is attributable to the large residual orientation error. If more accurate orientation is required, proper weighting on the orientation component of the pose vector can be used. For example, choosing **Q** = diag(1, 1, 1, 20, 20, 20), the residual error comparison is listed in Table 4.7, which shows that the average residual error of orientation is reduced to 0.79 degree, at the expense of position accuracy (average position error is up to 0.82 (mm) in this case). Since robot end-effector orientation errors can be corrected by using passive mechanical fixtures, in most cases robot end-effector position accuracy is more critical than orientation accuracy, therefore the simple identity scaling on position and orientation is appropriate, and no weighting on the orientation component will be used in the following work.

[3] The condition number of the identification Jacobian using standard D-H model is 3.1968×10^{17}

Table 4.4 Nominal Parameters of a Puma Robot Using a Modified D-H Model

Link No.	a_i (mm)	d_i (mm)	α_i (rad.)	β_i (rad.)	TOOL
0	0.0	0.0	0.0	0	—
1	0.0	0.0	$-\pi/2$	0	0.0 (rad.)
2	431.8	149.09	0.0	0	0.0 (rad.)
3	-20.32	0	$\pi/2$	$\pi/2$	0.0 (rad.)
4	0.0	433.07	$-\pi/2$	0	0.0 (mm)
5	0.0	0.0	$\pi/2$	0	0.0 (mm)
6	0.0	56.25	0.0	0	55.0 (mm)

Table 4.5. Identified Kinematic Errors of the Puma robot (a modified D-H Model)

Link No.	Δa_i (mm)	Δd_i (mm)	$\Delta \alpha_i$ (rad.)	$\Delta \theta_i$ (rad.)	$\Delta \beta_i$ (rad.)
0	0.302	-3.356	-0.003	0.0005	0
1	0.174	1.112	-0.0011	0.0004	0
2	0.157	0	-0.0031	-0.0026	-0.011
3	0.048	1.174	0.0045	0.0010	0
4	-0.033	-0.719	-0.0043	0.0110	0
5	-0.169	-2.447	-0.0209	0.0046	0
6	-0.031	0.455	0.0002	0.0111	0

Table 4.6 Residual Error Comparisons Using the Modified D-H model

	Before Calibration		After Calibration	
(in length)	Position (mm)	Orien.(degree)	Position (mm)	Orien.(degree)
average	3.30	2.63	0.19	1.31
standard dev.	1.15	0.41	0.097	0.45
maximum	4.74	3.21	0.40	2.49

Table 4.7 Residual Error Comparisons Using the Modified D-H model

	Before Calibration		After Calibration	
(in length)	position (mm)	orien. (degree)	position (mm)	orien. (degree)
average	3.30	2.63	0.82	0.79
standard dev.	1.15	0.41	0.43	0.30
maximum	4.74	3.21	2.06	1.71

(Using weight matrix **Q** = diag(1, 1, 1, 20, 20, 20) in the residual pose vector (4.4))

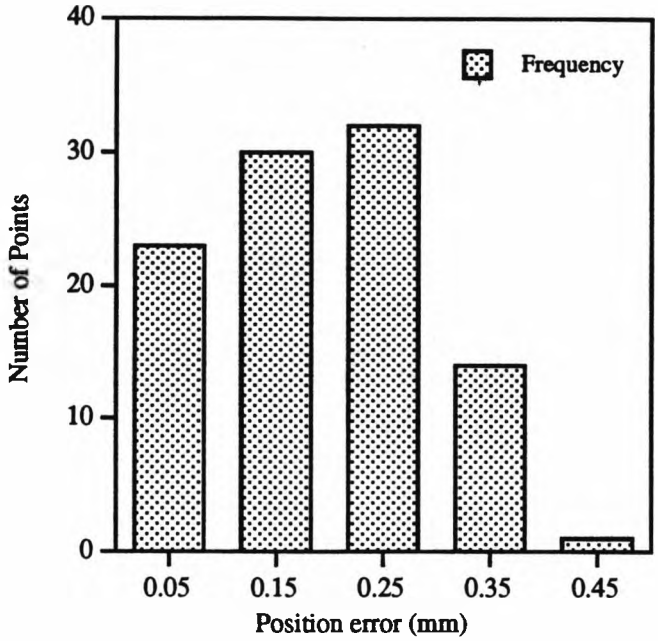


Figure 4.9. The Residual Position Errors Distribution for the 100 Test Points

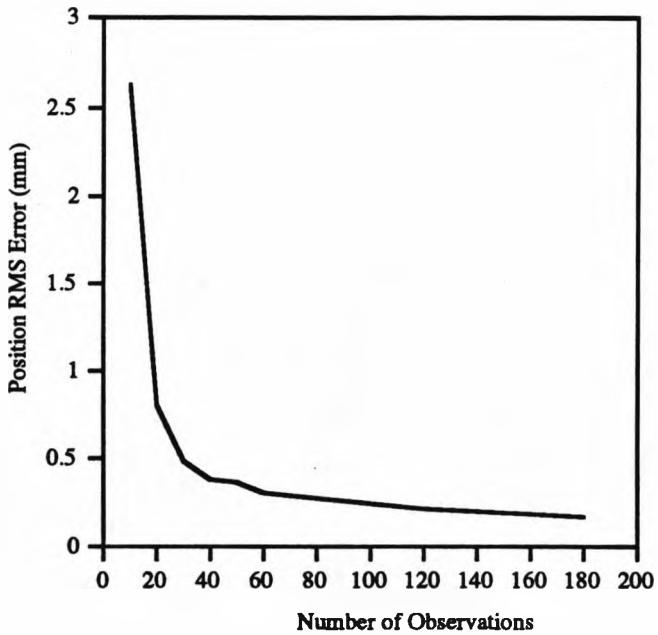


Figure 4.10. The Relationship Between the Observations and Final RMS Errors

The number of measurement points is an important factor which affects the kinematic identification efficiency and accuracy. An insufficient number of measurements (insufficient excitation) will produce an ill-conditioned or singular identification Jocobian which in turn causes numerical problems during the identification process. Too many measurements will increase the data collection cost, computation time, and sometimes even degrade the quality of identification solution. The number of measurements ($M = 120$) used in this calibration is determined based on extensive numerical experiments. Figure 4.10 shows the relationship between the number of measurements and the final residual position errors for the 100 test points. The final RMS (Root Mean Square) error decreased with increasing number of observations. But the decrease of the RMS after certain number of observations is not significant. The number of 120 is decided as an adequate number of measurements for this calibration based on the Figure 4.10.

4.6 Chapter Summary

The Hopfield type recurrent neural network (RNN)-based kinematic identification algorithm has been developed and experimentally evaluated in this chapter. The quadratic form of robot inaccuracy was constructed as the network energy function. The network connection weights and input currents are determined by nominal kinematic model and parameters. The network converges rapidly (in a few characteristic time constants of the neural circuit) to the optimal solutions which minimises the network energy function. The final states of the neuron variables correspond to the kinematic parameters to be identified. If robot inaccuracy data can be collected on-line, robot kinematic identification can be performed in real time by using the RNN-based real time optimisation technique. Due to the use of ordinary differential equations (ODE) in the simulation, the RNN-based algorithm also exhibit numerical robustness over conventional least square methods. For parallel implementation, the computation time of the RNN-based algorithm is independent of the number of robot DoF and the number of parameters to be identified. Therefore, the RNN-based algorithm is especially attractive for a robot with multiple DoF (redundant robot) and requiring to maintain its own calibration in real-time.

CHAPTER 5

AUTONOMOUS CALIBRATION USING A TRIGGER PROBE

5.1 Introduction

It is inevitable that a robot will have its links bent, base moved, or some components repaired during its service lifetime. A robot needs to have periodic re-calibration to maintain its positioning accuracy. In such situations it is desirable not to have to resort the use of special-purpose calibration equipment to update the model for robot control. An ultimate goal would be for the robot to be able to calibrate its internal model in real time (Bennett, Geiger, Hollerbach, 1991). In Chapter 4, a RNN-based kinematic identification algorithm was developed which is capable of performing kinematic identification in real-time. However, as reviewed in Chapter 2, measurement is one of the most difficult aspects of robot calibration which consumes the most time and effort involved in the calibration process. Typically, existing calibration systems have large volume and very stringent installation requirements, and the robot has to be removed from its normal working environment in order to perform the calibration. Moving the robot leads to a loss of base information, which will affect the accuracy of the calibrated model seriously since robot base location is as significant as the identification of robot kinematic parameters (Driels and Swayze, 1994). Due to the high cost of calibration equipment, the need for trained skilled operators, and the long production down-time, extensive calibration is still an expensive procedure, and becomes unacceptable when it has to be applied repeatedly. The development of a low-cost, easy-to-use calibration tool suitable for on-site autonomous calibration is the subject of this chapter.

Autonomous calibration is defined as an automated process that determines the model parameters by using only the robot's *internal sensors* (Bennett, Geiger and Hollerbach, 1991). The basic observation, that a mobile closed-loop kinematic chain can be formed either by connecting the end-effector of two open-chain mechanisms or by adding additional links or joints between the end-effector and ground, serves as the basis for several other researchers' work (Tang and Mooring, 1992; Goswami, Quaid and Peshkin, 1993; Driels and Swayze, 1994). The concept of robot self-calibration using a defined task constraint is appealing, since it dispenses with the need for pose measurement equipment. There are however some disadvantages to this method. In general either another robot or some special mechanical fixtures are needed to accomplish the kinematic closure, which requires painstaking efforts to set up. The method is therefore not autonomous in the sense that it requires extensive human intervention during experimental set-up and calibration.

In this chapter, we present a new robot kinematic calibration scheme which can be implemented autonomously and is suitable for on-site calibration in an industrial environment. The known shape of an object is used to obtain robot kinematic constraint equations instead of using known reference locations in workspace. Gripping a simple trigger probe-*Renishaw probe* (Figure 5.2), the robot uses the probe as its extended link to touch constraint planes in its workspace (the locations of the constraint planes are not necessarily known exactly). Only the robot joint readings and the Cartesian position values (reported by the controller) are recorded for identification while the tip-point of the probe is touching the constraint planes from various configurations. Neither external sensor measurements nor human intervention are required in the calibration, hence the calibration process is fully autonomous. A linear identification model has been derived from the consistency conditions of a plane, and is presented in the next section. A RNN-based kinematic identification algorithm based on the Chapter 4 is given in section 5.3. In section 5.4, we describe the data collection method used and the experimental set-up. Both simulation and experimental results for a PUMA 560 robot are given in section 5.5. The cross evaluation results using an external global measuring device are also presented. Discussions and conclusions are given in the final section.

5.2 Formulation of the Kinematic Identification Model

The objective of kinematic identification is to identify the actual kinematic parameters of the robot manipulator. Let $\Delta\rho$ be the kinematic errors which are assumed to be small perturbations from the nominal values specified by the robot manufacturer, where $\Delta\rho$ is an n by 1 vector, and n is the number of kinematic parameters. Without external measurements, the actual position $\mathbf{p}_l = [x_l\, y_l\, z_l]^T$ of the robot end-point remains unknown but must be near the nominal position $\mathbf{p}_l^0$ predicted by the robot controller. Using the Taylor series to the first order, we have:

$$\mathbf{p}_l = \mathbf{p}_l^0 + \mathbf{J}_l\Delta\rho = [x_l^0\, y_l^0\, z_l^0]^T + [\mathbf{j}_l^x\, \mathbf{j}_l^y\, \mathbf{j}_l^z]^T\Delta\rho \qquad (5.1)$$

where $\mathbf{J}_l$ is a 3 by n matrix which is the positional component of robot special Jacobian, which can be calculated with robot joint readings and nominal kinematic parameters; $\mathbf{j}_l^x$, $\mathbf{j}_l^y$ *and* $\mathbf{j}_l^z$ is the x, y and z component of $\mathbf{J}_l$ respectively, and l is the subscript index representing different touch points.

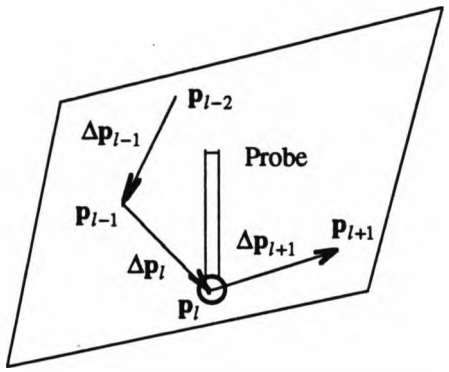


Figure 5.1. Constraint conditions for co-planar points

Although the exact locations of touch points are unknown, they are constrained to lie on a plane. The consistency condition of a plane leads to the construction of the identification model (Figure 5.1). The difference vector between two consecutive touch points is:

$$\Delta\mathbf{p}_l = \mathbf{p}_l - \mathbf{p}_{l-1} = [\Delta x_l^0\, \Delta y_l^0\, \Delta z_l^0]^T + [\Delta\mathbf{j}_l^x\, \Delta\mathbf{j}_l^y\, \Delta\mathbf{j}_l^z]^T\Delta\rho \qquad (5.2)$$

where

$$\Delta x_l^0 = x_l^0 - x_{l-1}^0,\ \Delta y_l^0 = y_l^0 - y_{l-1}^0,\ \Delta z_l^0 = z_l^0 - z_{l-1}^0;$$

$\Delta \mathbf{j}_l^x = \mathbf{j}_l^x - \mathbf{j}_{l-1}^x$, $\Delta \mathbf{j}_l^y = \mathbf{j}_l^y - \mathbf{j}_{l-1}^y$, $\Delta \mathbf{j}_l^z = \mathbf{j}_l^z - \mathbf{j}_{l-1}^z$

The difference vectors are normalised to roughly unit vectors by dividing the vectors using their nominal length.

$$\Delta \mathbf{p}_l = \frac{\mathbf{p}_l - \mathbf{p}_{l-1}}{\|\mathbf{p}_l - \mathbf{p}_{l-1}\|} \approx \frac{\mathbf{p}_l - \mathbf{p}_{l-1}}{\sqrt{(\Delta x_l^0)^2 + (\Delta y_l^0)^2 + (\Delta z_l^0)^2}} \tag{5.3}$$

The necessary and sufficient condition for touch points $\mathbf{p}_{l-2}$, $\mathbf{p}_{l-1}$, $\mathbf{p}_l$, and $\mathbf{p}_{l+1}$ to lie on one constraint plane is that the volume of the parallelepiped formed by the difference vector $\Delta \mathbf{p}_{l-1}$, $\Delta \mathbf{p}_l$, and $\Delta \mathbf{p}_{l+1}$ must be equal to zero, i.e., the determinant of the matrix formed by the difference vectors must be zero. Using (5.2), we have Equation (5.4):

$$\left\| \left[\Delta \mathbf{p}_{l-1} \; \Delta \mathbf{p}_l \; \Delta \mathbf{p}_{l+1} \right] \right\| = \begin{vmatrix} \Delta x_{l-1}^0 + \Delta \mathbf{j}_{l-1}^x \Delta\rho & \Delta x_l^0 + \Delta \mathbf{j}_l^x \Delta\rho & \Delta x_{l+1}^0 + \Delta \mathbf{j}_{l+1}^x \Delta\rho \\ \Delta y_{l-1}^0 + \Delta \mathbf{j}_{l-1}^y \Delta\rho & \Delta y_l^0 + \Delta \mathbf{j}_l^y \Delta\rho & \Delta y_{l+1}^0 + \Delta \mathbf{j}_{l+1}^y \Delta\rho \\ \Delta z_{l-1}^0 + \Delta \mathbf{j}_{l-1}^z \Delta\rho & \Delta z_l^0 + \Delta \mathbf{j}_l^z \Delta\rho & \Delta z_{l+1}^0 + \Delta \mathbf{j}_{l+1}^z \Delta\rho \end{vmatrix} = 0 \tag{5.4}$$

Ignoring the second and higher order of $\Delta\rho$, Equation (5.4) can be written as:

$$\begin{vmatrix} \Delta \mathbf{j}_{l-1}^x \Delta\rho & \Delta x_l^0 & \Delta x_{l+1}^0 \\ \Delta \mathbf{j}_{l-1}^y \Delta\rho & \Delta y_l^0 & \Delta y_{l+1}^0 \\ \Delta \mathbf{j}_{l-1}^z \Delta\rho & \Delta z_l^0 & \Delta z_{l+1}^0 \end{vmatrix} + \begin{vmatrix} \Delta x_{l-1}^0 & \Delta \mathbf{j}_l^x \Delta\rho & \Delta x_{l+1}^0 \\ \Delta y_{l-1}^0 & \Delta \mathbf{j}_l^y \Delta\rho & \Delta y_{l+1}^0 \\ \Delta z_{l-1}^0 & \Delta \mathbf{j}_l^z \Delta\rho & \Delta z_{l+1}^0 \end{vmatrix} + \begin{vmatrix} \Delta x_{l-1}^0 & \Delta x_l^0 & \Delta \mathbf{j}_{l+1}^x \Delta\rho \\ \Delta y_{l-1}^0 & \Delta y_l^0 & \Delta \mathbf{j}_{l+1}^y \Delta\rho \\ \Delta z_{l-1}^0 & \Delta z_l^0 & \Delta \mathbf{j}_{l+1}^z \Delta\rho \end{vmatrix} + \begin{vmatrix} \Delta x_{l-1}^0 & \Delta x_l^0 & \Delta x_{l+1}^0 \\ \Delta y_{l-1}^0 & \Delta y_l^0 & \Delta y_{l+1}^0 \\ \Delta z_{l-1}^0 & \Delta z_l^0 & \Delta z_{l+1}^0 \end{vmatrix} = 0 \tag{5.5}$$

Prior to the expansion of the equation (5.5), let us introduce a more compact notation. We define a function detm(X_1, X_2, X_3) which generalises determinant calculation to the situation where one of the three variables X_1, X_2, X_3 is a 3 by M matrix (where M is an integer larger than one), and the other two variables are 3 by 1 vector. The detm(X_1, X_2, X_3) will return a matrix of 1 by M, whose element resulting from the determinant of the matrix formed by the corresponding *column vector* of the matrix variable and the other two vector variables. For example:

$$\mathbf{detm}\left(\begin{bmatrix} x_1 & \begin{bmatrix} x_{21} & x_{22} \end{bmatrix} & x_3 \\ y_1 & \begin{bmatrix} y_{21} & y_{22} \end{bmatrix} & y_3 \\ z_1 & \begin{bmatrix} z_{21} & z_{22} \end{bmatrix} & z_3 \end{bmatrix} \right) = \begin{bmatrix} \begin{vmatrix} x_1 & x_{21} & x_3 \\ y_1 & y_{21} & y_3 \\ z_1 & z_{21} & z_3 \end{vmatrix} & \begin{vmatrix} x_1 & x_{22} & x_3 \\ y_1 & y_{22} & y_3 \\ z_1 & z_{22} & z_3 \end{vmatrix} \end{bmatrix} \tag{5.6}$$

Using the definition of detm(•), Equation (5.5) is written as:

$$(\text{detm}\begin{pmatrix} \Delta j_{l-1}^x & \Delta x_l^0 & \Delta x_{l+1}^0 \\ \Delta j_{l-1}^y & \Delta y_l^0 & \Delta y_{l+1}^0 \\ \Delta j_{l-1}^z & \Delta z_l^0 & \Delta z_{l+1}^0 \end{pmatrix} + \text{detm}\begin{pmatrix} \Delta x_{l-1}^0 & \Delta j_l^x & \Delta x_{l+1}^0 \\ \Delta y_{l-1}^0 & \Delta j_l^y & \Delta y_{l+1}^0 \\ \Delta z_{l-1}^0 & \Delta j_l^z & \Delta z_{l+1}^0 \end{pmatrix} + \text{detm}\begin{pmatrix} \Delta x_{l-1}^0 & \Delta x_l^0 & \Delta j_{l+1}^x \\ \Delta y_{l-1}^0 & \Delta y_l^0 & \Delta j_{l+1}^y \\ \Delta z_{l-1}^0 & \Delta z_l^0 & \Delta j_{l+1}^z \end{pmatrix})\Delta\rho$$
$$+\begin{vmatrix} \Delta x_{l-1}^0 & \Delta x_l^0 & \Delta x_{l+1}^0 \\ \Delta y_{l-1}^0 & \Delta y_l^0 & \Delta y_{l+1}^0 \\ \Delta z_{l-1}^0 & \Delta z_l^0 & \Delta z_{l+1}^0 \end{vmatrix} = 0 \qquad (5.7)$$

Note that in equation (5.7) $\Delta\mathbf{J}_l = [\Delta j_l^x\ \Delta j_l^y\ \Delta j_l^z]^T$ is a 3 by n matrix, as well as $\Delta\mathbf{J}_{l-1}$ and $\Delta\mathbf{J}_{l+1}$.

Denoting $\Delta X_l = \begin{vmatrix} \Delta x_{l-1}^0 & \Delta x_l^0 & \Delta x_{l+1}^0 \\ \Delta y_{l-1}^0 & \Delta y_l^0 & \Delta y_{l+1}^0 \\ \Delta z_{l-1}^0 & \Delta z_l^0 & \Delta z_{l+1}^0 \end{vmatrix}$ and

$$\mathbf{H}_l = \text{detm}\begin{pmatrix} \Delta j_{l-1}^x & \Delta x_l^0 & \Delta x_{l+1}^0 \\ \Delta j_{l-1}^y & \Delta y_l^0 & \Delta y_{l+1}^0 \\ \Delta j_{l-1}^z & \Delta z_l^0 & \Delta z_{l+1}^0 \end{pmatrix} + \text{detm}\begin{pmatrix} \Delta x_{l-1}^0 & \Delta j_l^x & \Delta x_{l+1}^0 \\ \Delta y_{l-1}^0 & \Delta j_l^y & \Delta y_{l+1}^0 \\ \Delta z_{l-1}^0 & \Delta j_l^z & \Delta z_{l+1}^0 \end{pmatrix} + \text{detm}\begin{pmatrix} \Delta x_{l-1}^0 & \Delta x_l^0 & \Delta j_{l+1}^x \\ \Delta y_{l-1}^0 & \Delta y_l^0 & \Delta j_{l+1}^y \\ \Delta z_{l-1}^0 & \Delta z_l^0 & \Delta j_{l+1}^z \end{pmatrix}$$

We have a linear system:

$$\mathbf{H}_l\Delta\rho + \Delta X_l = 0 \qquad (5.8)$$

From the above derivation, we see that every four consecutive touch points will decide a constraint equation (5.8). Using (m+3) consecutive touch points, a linear system consisting of m constraint equations is obtained:

$$\mathbf{H}\Delta\rho + \Delta\mathbf{X} = \mathbf{0} \qquad (5.9)$$

where $\mathbf{H} = [\mathbf{H}_1\mathbf{H}_2,...,\mathbf{H}_m]^T$ and $\Delta\mathbf{X} = [\Delta X_1\Delta X_2,...,\Delta X_m]^T$

In Equation 5.9, the coefficient matrix **H** and $\Delta\mathbf{X}$ can be calculated based on the *difference* of nominal positions predicted by the controller, and the *difference* of special Jacobian for each pair of consecutive touch points (only joint readings are required for the computations). The only unknown remaining is the kinematic error to be identified. Equation 5.9 serves as the linear identification model of the new method.

The derivation of the above model assumes the general case in which neither the position nor the orientation of the constraint plane is known accurately. As a special case, assuming that we have the knowledge of the orientation of the constraint plane with respect to the robot base co-ordinate system, for example, the constraint plane is

aligned with the robot base x-y plane, then the z component of $\Delta\mathbf{p}_i$ will be zero. From (5.2), we have:

$$\Delta\mathbf{j}_i^z\Delta\rho + \Delta z_i^0 = 0 \tag{5.10}$$

Comparing the above equation with the constraint Equation 5.8, we see that the coefficient matrix ΔX_i is simply the difference of z component of two consecutive touch points, and the $\mathbf{H}_i$ simply the difference of z component of special Jacobian at two consecutive touch points. Similarly, we have the constraint equations for the cases where the constraint plane is aligned with robot base y-z or x-z planes:

$$\Delta\mathbf{j}_i^x\Delta\rho + \Delta x_i^0 = 0 \tag{5.11}$$

or

$$\Delta\mathbf{j}_i^y\Delta\rho + \Delta y_i^0 = 0 \tag{5.12}$$

Although the exact location of the constraint plane is not necessarily known, care must be exercised in placing the constraint plane in the workspace. Considerations include:

- the robot configurations enabling desirable and safe touch on the plane;
- the workspace in which accuracy is critical;
- the optimal identification configurations of the robot;
- the workcell layout and kind of constraint plane available.

As robot base axes are always aligned with respect to some reference planes in the workcell, the assumption made earlier, in which the constraint plane is aligned with the robot base co-ordinates, has practical significance. In the case that external constraint planes are used, it is also easy to align the plane with the robot base axis using the probe and VAL II [4] axis motion function. To maximise the range of robot movements, it is desirable to have the robot touch the constraint planes separately which lie perpendicular to the robot's base axes. The linear identification model in this case can be constructed according to Equations 5.10-5.12. The calculations are simplified and the calibration results can be evaluated directly in the special case.

[4] For UNIMATION PUMA robot in this case

5.3 RNN-based Identification Algorithm

In Chapter 4, a RNN-based kinematic identification algorithm was developed which exhibited numerical efficiency and robustness. A similar algorithm is presented here for the linear identification problem, which might be ill-conditioned due to the limited range of robot movement during data collection. The resolution of Equation 5.9 is equivalent to the minimisation of the following energy function in the least square sense:

$$E = \frac{1}{2}(\|\mathbf{H}\Delta\rho + \Delta\mathbf{X}\|^2 + \alpha\|\Delta\rho\|^2) \tag{5.13}$$

where α is a positive scalar coefficient for regulation (Cichocki and Unbehauen, 1993), and $\| \|$ is a Euclidean norm. Comparing Equation 5.13 with 4.11, an extra regulation (penalty) term is added to improve the conditioning of identification Jacobian, which is near singular due to the partial pose information used in this case, and the limited movement ranges for data collection. The physical meaning of the regulation (penalty) term is to ensure a small norm of the identified kinematic error vector.

Writing Equation. 5.13 in the form of Hopfield network energy, we have:

$$E = -\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}T_{ij}\Delta\rho_i\Delta\rho_j - \sum_{j=1}^{n}I_j\Delta\rho_j + \tfrac{1}{2}\sum_{i=1}^{m}(\Delta X_i)^2 \tag{5.14}$$

where

$$T_{ij} = -\sum_{s=1}^{m}H_{si}H_{sj} + \alpha\delta_{ij}, \tag{5.15}$$

where $\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$

and

$$I_j = -\sum_{i=1}^{m}H_{ij}\Delta X_i, \tag{5.16}$$

T_{ij} and I_j determine the network connection weights and input currents respectively based on the nominal kinematic model and parameters. The $\Delta\rho_j$ is the j-th component of kinematic parameters to be identified which corresponds to the j-th neuron state. n is the dimension of the vector of kinematic error while m is the row number of coefficient

matrix **H**. Following the procedure of the derivation of Equation 4.16, the neural dynamic equation is given as:

$$\frac{d(\Delta\rho_i)}{dt} = \mu_i(\sum_{j=1}^{n} T_{ij}\Delta\rho_j + I_i) \tag{5.17}$$

where the net connection weight and input current are defined as (5.15) and (5.16), and μ_i is the same as defined in Chapter 4. The resolution of the ordinary differential equation (ODE) (5.17) determines the kinematic parameter errors to be identified.

5.4 Data Collection

Renishaw Probes were originally used for accurate workpiece set up and workpiece measurement for CNC lathes. The probe is in effect an omni-directional switch that triggers when the probe contacts the workpiece from any direction (Renishaw Metrology, Ltd, 1983). For our application, a special tool changer was made to hold the probe (Figure 5.3). The trigger signal is transmitted as an input to control the robot. The switch is kept on while the probe is in contact with the object. The tip-point of the probe is a ruby ball so that the contact point from any direction is a constant distance from the centre of the tip. The probe has 12.5° over-travel in ±X, Y direction (which is equivalent to about 22 mm over-travel for a 100 mm long probe stylus), and 6.5 mm over-travel in Z direction, which allows a certain probing speed and misplacement of workpieces. The trigger force in the X, Y direction is set at 10 (g) and 15 (g) in the Z direction. The probe has repeatability of 1 (μm) thus can be used for high precision measurements. The data collection procedure written in pseudo-code is as follows;

```
Repeat until the maximum number of touch points is reached {
        MoveTo(START);
        Point = RandomConfiguration(MAX, MIN);
        MoveTo(Point);
        while(ProbeSignal == OFF){
                ProbeBy(XSTEP, YSTEP, ZSTEP);
                }
        while(ProbeSignal == ON){
                ProbeBy(-XSTEP/10, -YSTEP/10, -ZSTEP/10);
                }
        RecordData(JointValues, CartesianValues);
        }
```

The robot moves from a start point to a point above the constraint plane where its configuration is randomly generated within the robot movement ranges, from which a desirable and safe touch on the constraint plane is ensured[5]. From that point, the robot probes the plane by moving in small steps toward the constraint plane until the probe signal is on. As there is some over-travel of the stylus, the stop point is not the point of the first touch due to the probing speed. Therefore fine tuning is needed to retract the first touch point. Whilst still in contact with the plane, the fine tuning process begins by moving the tip-point away from the plane in steps of one tenth of the probing steps. Then the joint values of the robot and the corresponding Cartesian co-ordinates are recorded for post-processing. The above process is repeated until the desired number of touch points has been reached.

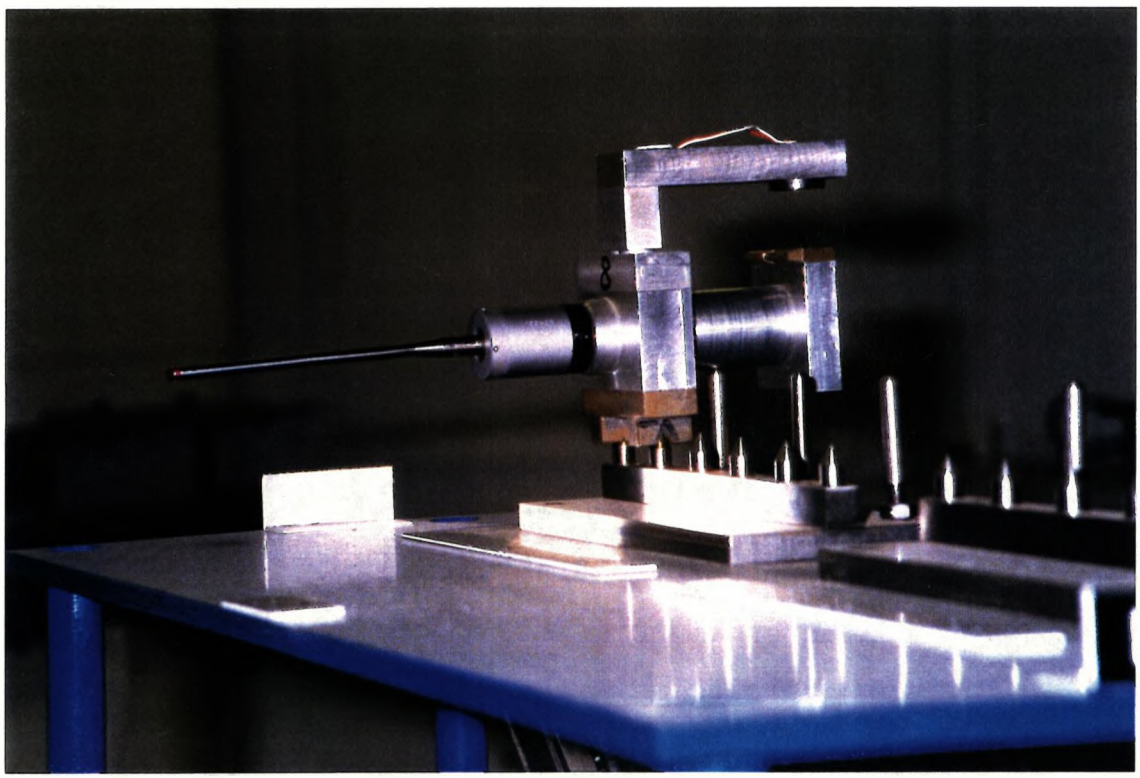


Figure 5.2. The Trigger Probe

[5] This can be done either on-line using VAL II random generator, given the intervals of Cartesian coordinates, or through off-line planning process using robot simulation package.

The probing and fine tuning direction is along the normal to the constraint plane. The probing steps are normally set less than 1 (mm) hence the fine tuning steps are less than 0.1 (mm), thus achieves a measurement accuracy in the order of 0.1 (mm) which is sufficient for robot calibration (if more accurate measurement is required, smaller probing steps can be set). In the special case when the constraint planes are aligned with robot base planes, the robot only probes in one axis direction, the movements along the other two axis directions are set to zero. The data collection program for probing z-constraint plane written in VAL II are listed in the appendix 5.

5.5 Results for a Puma 560 Robot

The PUMA 560 robot is a six DoF manipulator with six revolute joints. There are in total 24 kinematic parameters, using Denavit-Hartenberg (D-H) notation, to describe the kinemtaic model. Since joints 2 and 3 of the Puma robot are consecutive joints with parallel joint axes, a modified D-H model is used to avoid a model singularity (Veitschegger and Wu, 1988). The common normal parameter error Δd_2 is replaced by a rotation parameter $\Delta\beta_2$, and the value of Δd_2 and the values of $\Delta\beta_i$ ($i \neq 2$) are fixed to zero during the identification process. In general, another six parameters including three each of rotation {roll-pitch-yaw} and translation parameters are needed for TOOL transformation. The nominal parameters for the Puma robot are listed in Table 5.1. Since only position information are used for calibration, only three position parameters of the TOOL transformation are identifiable. To eliminate parameter redundancy between the TOOL and the last link, the TOOL parameters are incorporated into the last link. The three positional parameters of the tool are represented by the three parameters d_6, a_6 and θ_6 of the last link, and α_6 is not identifiable. The total number of identifiable kinematic parameters is therefore 23.

Table 5.1 Nominal Parameters of a Puma 560 Robot

No.	a_i (mm)	d_i (mm)	α_i (rad.)	β_i (rad.)	TOOL
1	0.0	0.0	$-\pi/2$	0	0.0 (rad.)
2	431.8	149.09	0.0	0.0	0.0 (rad.)
3	-20.32	0.0	$\pi/2$	0	0.0 (rad.)
4	0.0	433.07	$-\pi/2$	0	5 (mm)
5	0.0	0.0	$\pi/2$	0	0.0 (mm)
6	0.0	56.25	0.0	0	302.34 (mm)

5.5.1 Simulation Results

The simulation program was built to test the proposed calibration algorithm using MATLAB® running on Hewlet-Packard 9000 workstation. The flowchart of the simulation program was shown in Figure 5.3. To maximise the robot joint movements, three constraint planes were simulated to be placed perpendicular to each base axis: x = -550 (mm); y = 300 (mm); z = -450 (mm). Sixty random configurations in Cartesian space for each constraint plane were generated satisfying the constraint conditions. Then the nominal inverse kinematic model was used to find the corresponding joint values for each of the Cartesian configurations. If there was no error in the kinematic model, it would be found that the positions achieved by controlling those joint values would perfectly match those constraint conditions. However, by inducing small errors in the parameters, the achieved positions by the 'actual' robot will be different from the commanded ones thus deviate from the constraint planes. An iterative inverse Jacobian method was adopted to find the corresponding joint compensations so that the positions achieved by the 'actual' robot were identical to those commanded ones. The updated joint values then feed to the nominal forward kinematic model to simulate the positions reported by the robot controller. The positions reported by the nominal forward kinematic model will not satisfy the plane constraint conditions. These position values, together with the robot special Jacobian, are used to calculate coefficient matrix **H** and $\Delta\mathbf{X}$ in Equation (5.9). Then the recurrent neural network is applied to identify the

kinematic errors. The identified errors are then used to update the forward kinematic model. This process is repeated until the discrepancy is decreased to the desired level. Finally the identified errors are compared with the induced errors to evaluate the simulation results.

The ordinary differential Equation (5.17) was resolved by calling the SIMULINK™ ODE solver. Choosing the coefficient $\alpha = 0.1$, and the learning rate $\mu = 10^6$, it took less than one second of simulation time, and about 20 (μs) of the circuit settling time for the system to converge. Table 5.2 lists the identified kinematic errors using the identification model where neither orientation nor position of the constraint planes are known. The induced errors were randomly produced in the range of ±0.15 (mm) for linear parameters and in the range of ±0.015 radians for rotary parameters (the values in parenthesis). Comparing the induced errors with the identified, we can see that the angular errors are almost identical and the linear errors are similar but have small residual errors. Table 5.3 compares the deviations from the constraint planes before and after calibration. *dx* represents the deviations from the *x*-axis constraint plane $x = -550$ (mm), *dy* represents the deviations from the *y*-axis constraint plane $y = 300$ (mm), and *dz* represents the deviations from the *z*-axis constraint plane $z = -450$ (mm). The statistical analysis is based on the 40 calibration points on each of the constraint planes. It is seen that the induced errors, though small, will produce a maximum deviation from the constraint plane of up to 13.2 (mm). After identifying the kinematic errors, the deviations from the constraints planes are close to zero. Small residual errors exist in the *x*-axis and *z*-axis constraint planes (*dx* and *dz*), but the standard deviations of the residual errors are very small, which implies that the calibrated positions shift parallel from the constraint plane a small amount. Increasing the magnitude of the induced errors will result in an invalid identification which leads to the identified model pointing to a plane parallel to the constraint plane (the constraint condition still holds). This is due to the fact that no information of the relative position and orientation between the constraint planes and robot base frame are provided in the identification. The general model is suitable for robot on-site re-calibration on a periodical basis where the parameter changes from the previous calibration are relatively small. Information about the relative relation between the constraint planes and robot base (such as the constraint planes aligned with the base) will enable the model to identify larger kinematic errors.

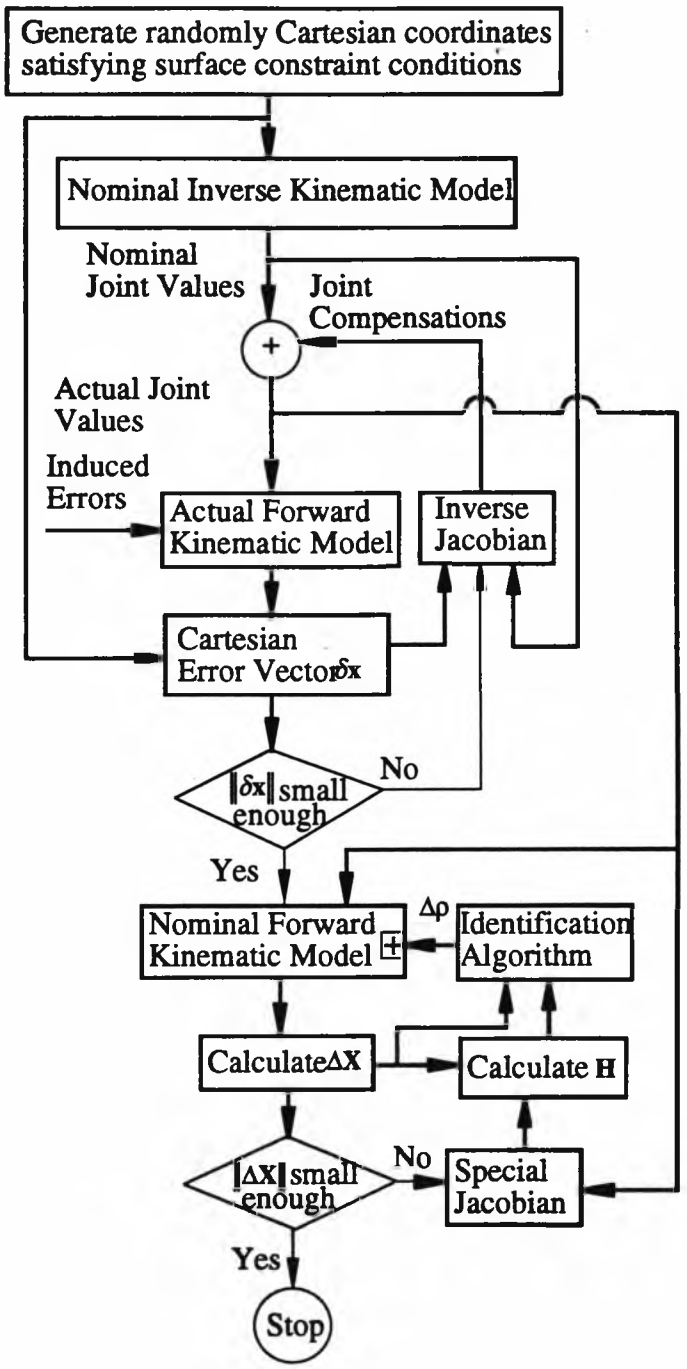


Figure 5.3. Simulation Program Flowchart

Table 5.2. Induced & Identified Kinematic Errors

Link No.	Δa_i (mm)	Δd_i (mm)	$\Delta \alpha_i$ (rad.)	$\Delta \theta_i$ (rad.)	$\Delta \beta_i$ (rad.)
1	0.087	0.055	0.0081	-0.0011	0
(induced)	(0.087)	(0.000)	(0.0080)	(-0.0019)	(0)
2	0.049	0	-0.0026	0.0059	0.0131
(induced)	(0.069)	(0)	(-0.0027)	(0.0060)	(0.0131)
3	-0.036	0.075	-0.0122	-0.0039	0
(induced)	(-0.004)	(0.011)	(-0.0123)	(-0.0039)	(0)
4	-0.061	-0.147	-0.0014	-0.0070	0
(induced)	(-0.053)	(-0.07)	(-0.0014)	(-0.0070)	(0)
5	0.015	0.029	0.0022	-0.0074)	0
(induced)	(0.041)	(0.016)	(0.0022)	(-0.0076)	(0)
6	0.146	-0.039	0	-0.0084	0
(induced)	(0.146)	(0.040)	(0)	(-0.0070)	(0)

Table 5.3. Accuracy Comparisons for Calibration Points

	Before Calibration			After Calibration		
(mm)	*dx*	*dy*	*dz*	*dx*	*dy*	*dz*
avg.	-3.756	-6.8745	-6.2575	-0.2049	-0.0876	-0.2646
stdev.	2.6299	2.7775	2.6459	0.0411	0.0429	0.0132
max.	6.5839	12.7363	13.1583	0.2681	0.1916	0.3098

(Simulation results with induced errors)

Table 5.4 lists the identification results using the identification model where the orientations of the constraint planes are known (aligned with the robot base axes). The induced errors were randomly produced in the range of ±2 (mm) for linear parameters and in the range of ±0.02 radians for angular parameters (the values in parenthesis). Both the linear and angular parameter errors identified are almost identical to those induced in this case. Comparing with the assumed positions of the constraint planes, the corresponding position component of the calibration points can be evaluated directly, which is listed in Table 5.5. The x component evaluation (dx) are based on the forty calibration points which lie on the constraint plane x = -550 (mm). The y, z component of position evaluation (dy and dz) are based on the calibration points which lie on the constraint plane y = 300 (mm), and z = -450 (mm) respectively. It is shown that the positioning accuracy has been improved significantly by identifying the induced errors, the maximum deviation from the constraint plane being decreased from up to 15 (mm) to below 0.2 (mm).

Figure 5.4 plots the x, y, and z component of the positioning deviations from the corresponding constraint plane. The first 40 points are the x component deviations from the constraint plane x = -550 (mm), the next 40 are y component deviations from the y = 300 (mm) constraint plane and the last 40 are z component deviations from the z = -450 (mm) constraint plane for the 40 calibration points on the corresponding constraint plane respectively. Note that only relative position information (the nominal position difference between two consecutive touch points) is used for calibration. But the calibrated model can predict accurately the absolute positions of the constraint planes. There exists a small residual error in the x component deviations from the x-axis constraint plane after calibration (the average error is 0.1114 mm, Table 5.5). It is due to the fact that the linear position information between robot base and the constraint planes are not provided in the identification. The big value of induced errors used in the simulation resulted in the calibrated model pointing to the positions shifted parallel from the constraint plane. This small parallel shift will be diminished by decreasing the induced errors. Since the assumed errors are larger than those for actual robots we generally dealt with in on-site calibration, the identification algorithm used in simulation is suitable for practical applications.

Table 5.4 . Induced & Identified Kinematic Errors

Link No.	Δa_i (mm)	Δd_i (mm)	$\Delta \alpha_i$ (rad.)	$\Delta \theta_i$ (rad.)	$\Delta \beta_i$ (rad.)
1	1.737	0.038	-0.0036	-0.0119	0
(induced)	(1.696)	(0.130)	(-0.0036)	(-0.0120)	(0)
2	1.175	0	-0.0012	-0.0175	0.0106
(induced)	(1.256)	(0)	(-0.0011)	(-0.0175)	(0.0106)
3	0.248	0.373	0.0160	-0.0005	0
(induced)	(0.236)	(0.374)	(0.0160)	(-0.0005)	(0)
4	0.179	1.560	-0.0106	-0.0048	0
(induced)	(0.195)	(1.662)	(-0.0106)	(-0.0048)	(0)
5	-0.788	-0.167	-0.0096	0.0076	0
(induced)	(-0.955)	(-1.367)	(-0.0130)	(0.0080)	(0)
6	-1.893	-1.932	0	0.0147	0
(induced)	(-1.894)	(-1.835)	(0)	(0.0138)	(0)

Table 5.5. Accuracy Comparisons for Calibration Points

	Before Calibration			After Calibration		
(mm)	*dx*	*dy*	*dz*	*dx*	*dy*	*dz*
avg.	-5.0646	1.9947	8.1754	0.1114	-0.0648	0.0051
stdev.	3.8930	3.6056	2.3923	0.0034	0.0093	0.0053
max.	14.8847	10.0196	13.0042	0.1186	0.1110	0.0148

(Simulation results with induced errors)

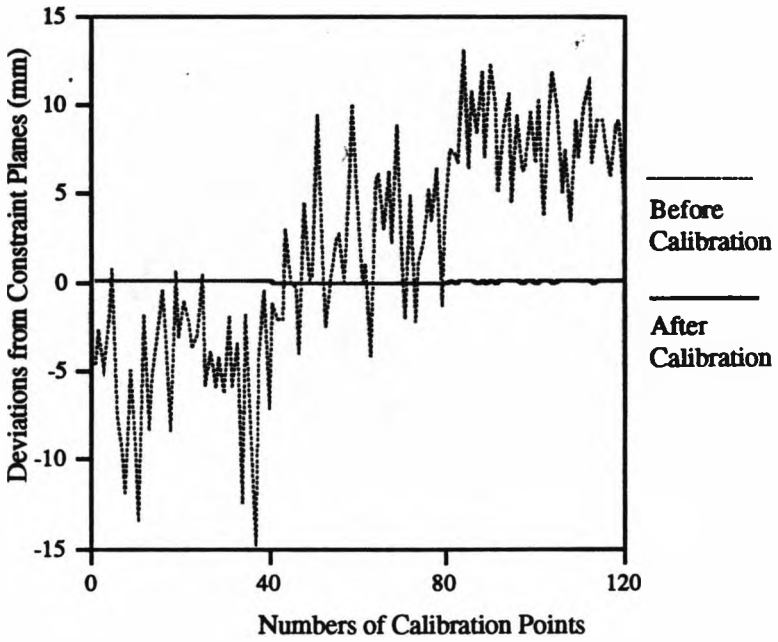


Figure 5.4. Simulation Result with Induced Errors

5.5.2 Experimental Results

To verify the proposed calibration scheme, the experimental set-up was used for data collection (Figure 5.5). A calibrated flat plate measuring 530 (mm) long and 250 (mm) wide was placed perpendicular to robot base x, y-axes for data collection. The robot x-y base plane was aligned precisely with the granite worktable of a co-ordinate measuring machine (CMM), therefore the worktable surface was used as the z-axis constraint plane. The positions of the x, y, z-axes constraint planes in the robot base frame, according to average values of the robot controller's readings, are at x = -652.070 (mm); y = 491.337 (mm); and z = -470.558 (mm). The worktable surface has a reachable area for the robot of about 80 (cm) by 110 (cm) which allows a wide range of robot movements in the x, y direction. The robot movement ranges are restricted such that the probe touch points lie on the constraint planes. The flat plate has a flatness of about ±0.001 (mm) and the flatness of the granite worktable surface is in the order of 5 (μm) which are accurate enough for robot calibration. The data collection procedure was

implemented in VAL-II and it took about five seconds to collect one data point. One hundred touch points on each of the plane were collected. Sixty pairs randomly chosen from each of the 100 pairs were used for calibration and the remaining points were used for independent test. Choosing a learning rate $\mu = 10^6$, the dynamic system (5.17) converges to its minimum rapidly. The trajectory of the kinematic identification in the first iteration is illustrated in Figure 5.6, which exhibits efficient and robust convergence (in about 20 μs for circuit implementation and less than 1 second in simulation), where linear parameters are in centimetres and angular parameters in radians. Table 5.5 lists the identified kinematic errors using the RNN-based identification algorithm based on the experimental data.

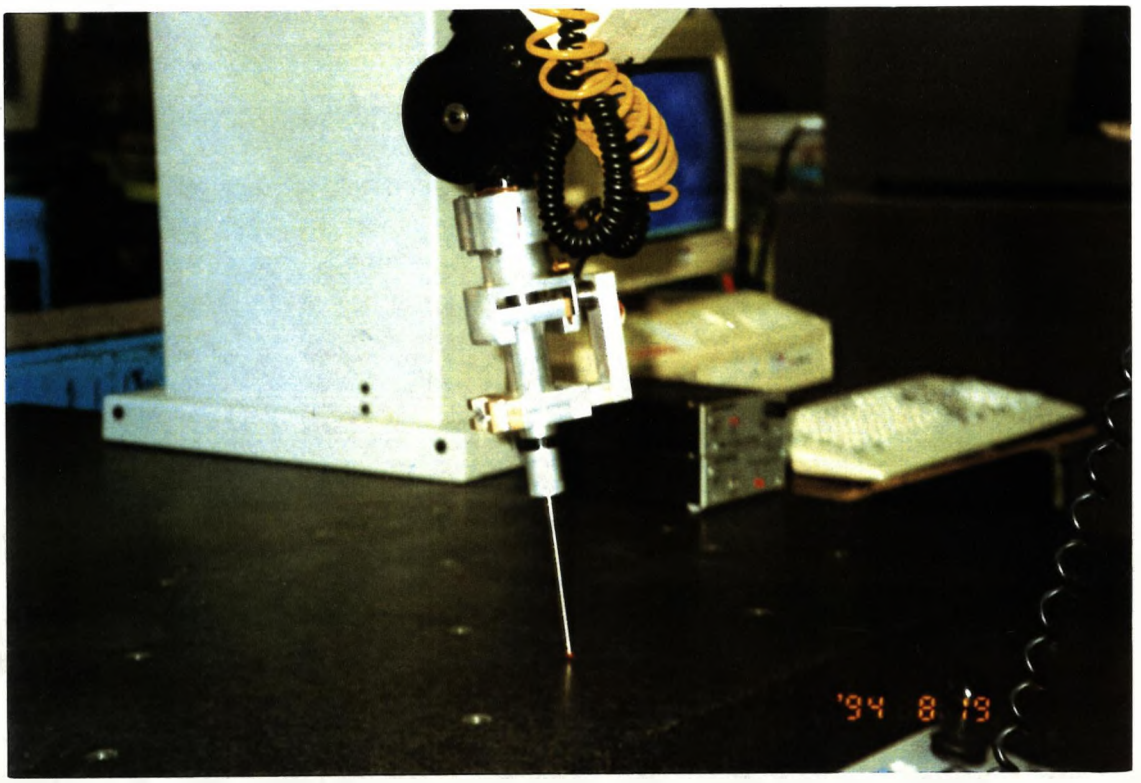


Fig.5.5 (a). Robot touch worktable surface (z-aixs constraint plane) using a probe

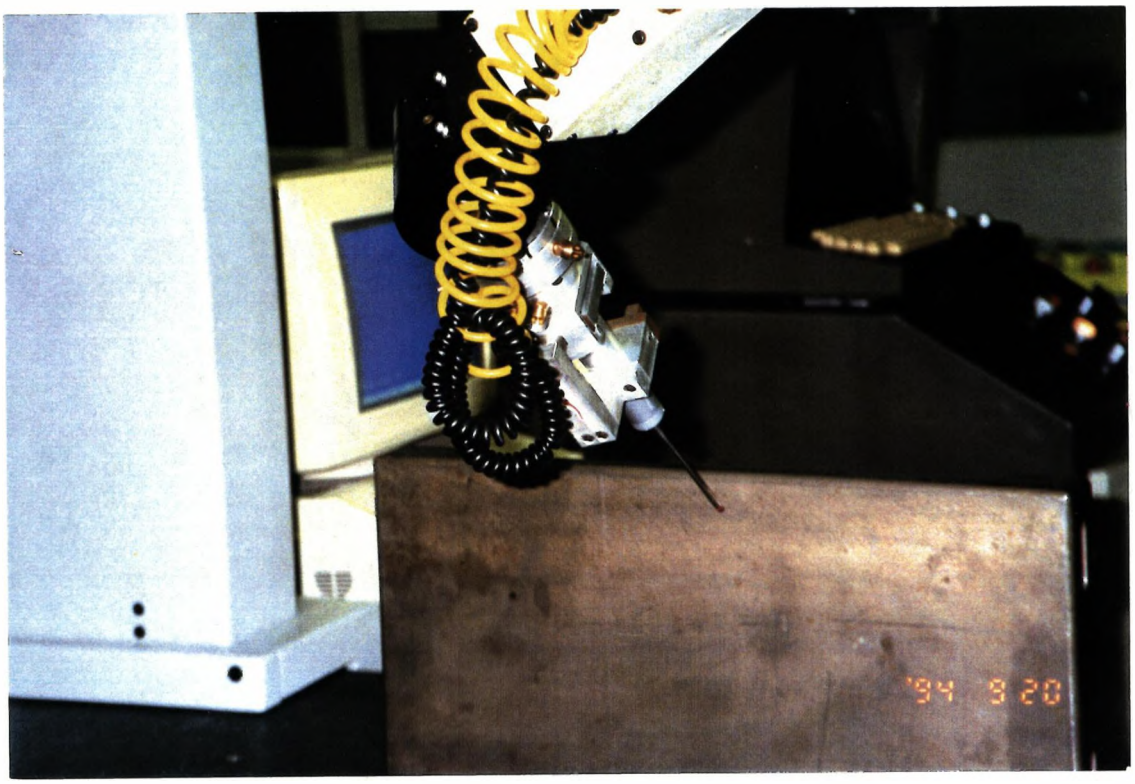


Fig. 5.5 (b). The robot touch an aligned (y-axis) constraint plane using a probe
Figure 5.5. Experimental Set-up for Data Collection

Figure 5.7 plots the x, y, and z component of the differences between each two consecutive touch points on the same constraint plane (the first 39 points are the difference of x components obtained from the 40 test points on the x-axis constraint plane, and the next 39 are y component differences obtained from the 40 test points on the y-axis plane and the last 39 are z component differences obtained from the 40 test points on the z-axis plane). The dashed lines represent the differences of positions predicted by the un-calibrated model in the robot controller while the solid lines represent the differences of positions predicted by the updated model using the identified errors. The symmetry of the graph is due to the use of the differences between consecutive points. It is shown that the calibrated model works well for test data points as well. Therefore the calibrated model is valid not only for the calibration points but also for the test points.

Using the reference positions of the constraint planes perpendicular to the base axis, we can evaluate directly the positioning accuracy achieved by this calibration. The accuracy comparisons based on the forty test data points on each of the three constraint

planes are given in Table 5.7, where *dx, dy* and *dz* represent the *x, y* and *z* component deviations from the *x, y* and *z*-axes constraint plane respectively. The average of absolute error after calibration has improved to below 0.3 (mm). The *z* component deviations from the *x*-*y* plane (*z*-axis constraint plane) before and after this calibration are illustrated in Figure 5.8 based on the 100 collected data from the *z*-axis constraint plan. The *x, y*-axes represent the *x, y* coordinates of those touch points on the *z*-axis constraint plane (which shows that the touch points on the *z*-axis constraint plane lie in the area of 400 x 200 mm^2), and the *z*-axis represents the *z* coordinate differences between the model predicted and the actual position of the *z*-axis constraint plane. Figure 5.8 (a) shows the *z*-axis constraint plane predicted by the robot model before calibration, while Figure 5.8 (b) shows the *z*-constraint plane predicted by the model after calibration. Figure 5.8 (b) is much closer to the actual shape and position of the *z*-axis constraint plane than Figure 5.9 (a). It has shown that the deviations from the *z*-axis constraint plane has been decreased significantly after calibration. We can see that this approach achieves an accuracy improvement comparable to other calibration methods using sophisticated external measurements. The maximum inaccuracy after calibration is of the order of robot repeatability for the test points on the constraint planes (Table 5.7).

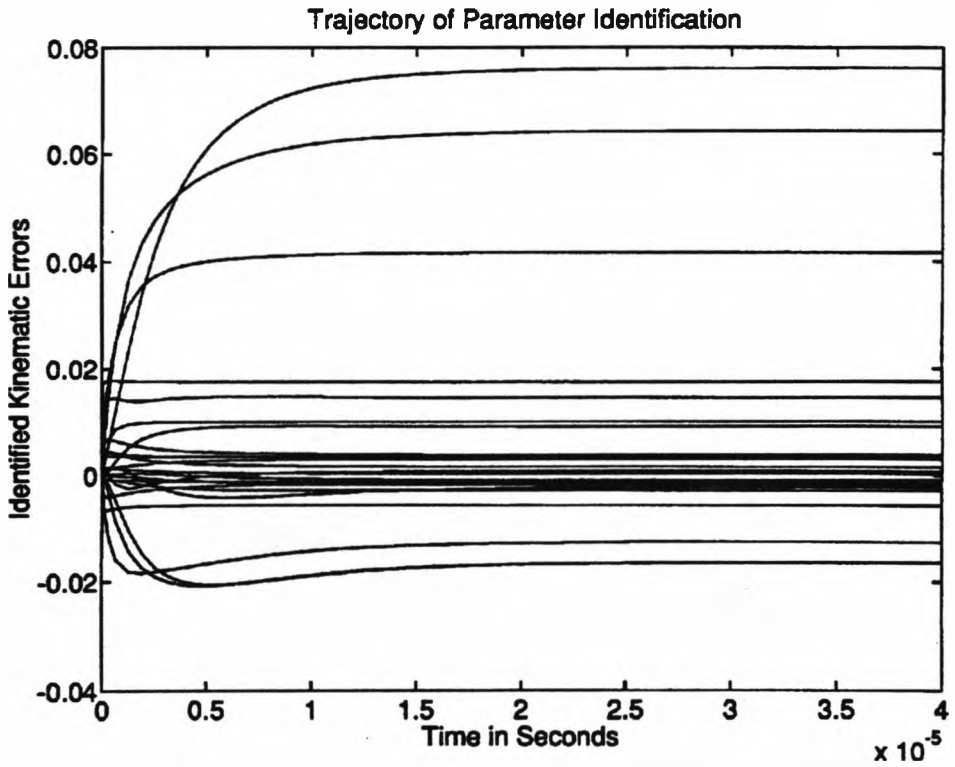


Figure 5.6. Time Evolution of Kinematic Errors during Identification

Table 5.6 Identified Errors of a Puma 560 Robot

Link No.	Δa_i (mm)	Δd_i (mm)	$\Delta \alpha_i$ (rad.)	$\Delta \theta_i$ (rad.)	$\Delta \beta_i$ (rad.)
1	1.027	-0.010	0.0018	0.0017	0
2	0.688	0	-0.0023	-0.0048	-0.0013
3	0.453	0.128	0.0007	0.0024	0
4	0.139	-0.252	0.0056	0.0172	0
5	-0.126	-0.441	-0.0061	0.0041	0
6	0.068	-0.348	0	0.0082	0

Table 5.7 Accuracy Comparisons Based on Test Points

	Before Calibration			After Calibration		
(mm)	*dx*	*dy*	*dz*	*dx*	*dy*	*dz*
average	1.017	4.058	1.890	0.190	0.259	0.223
stdev.	0.165	0.658	0.306	0.031	0.042	0.036
max.	2.218	4.866	3.924	0.558	0.573	0.571

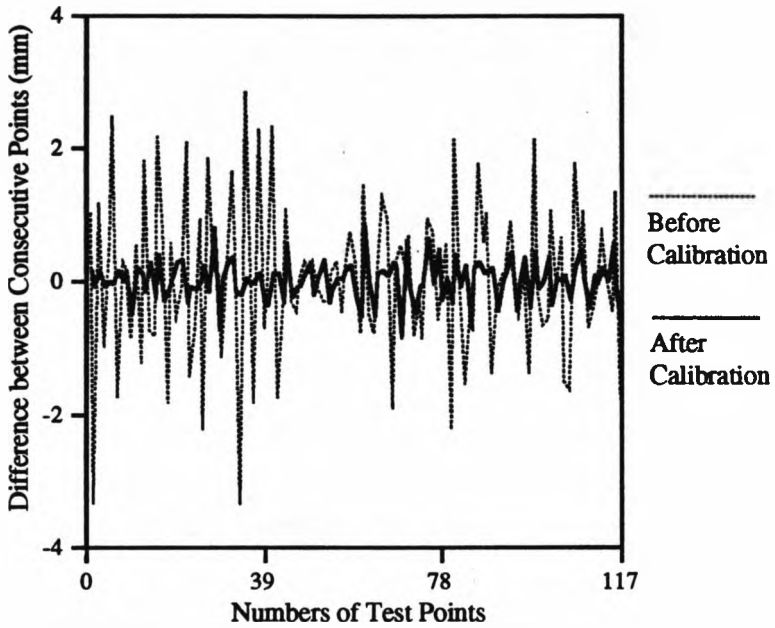
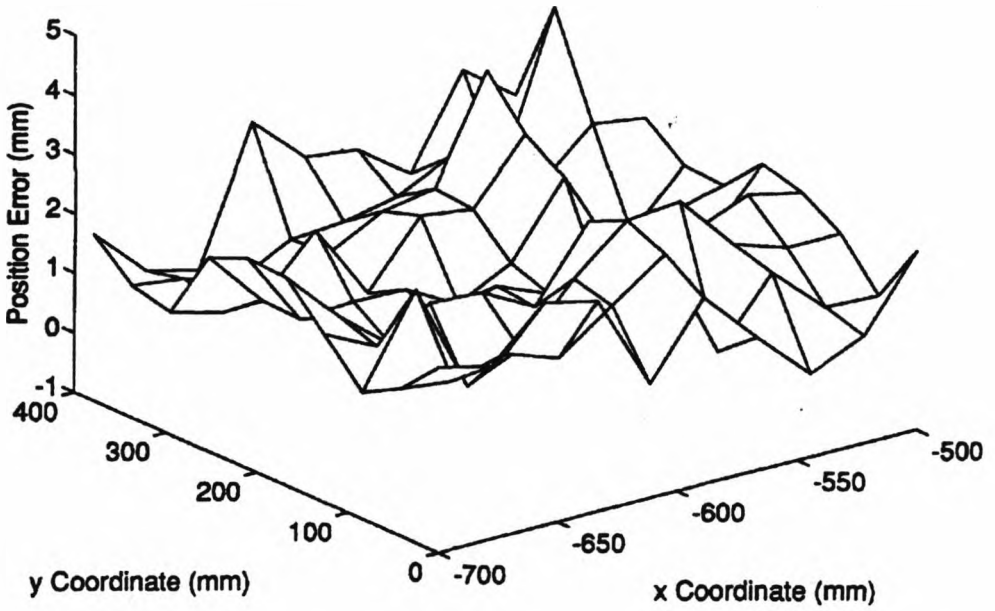
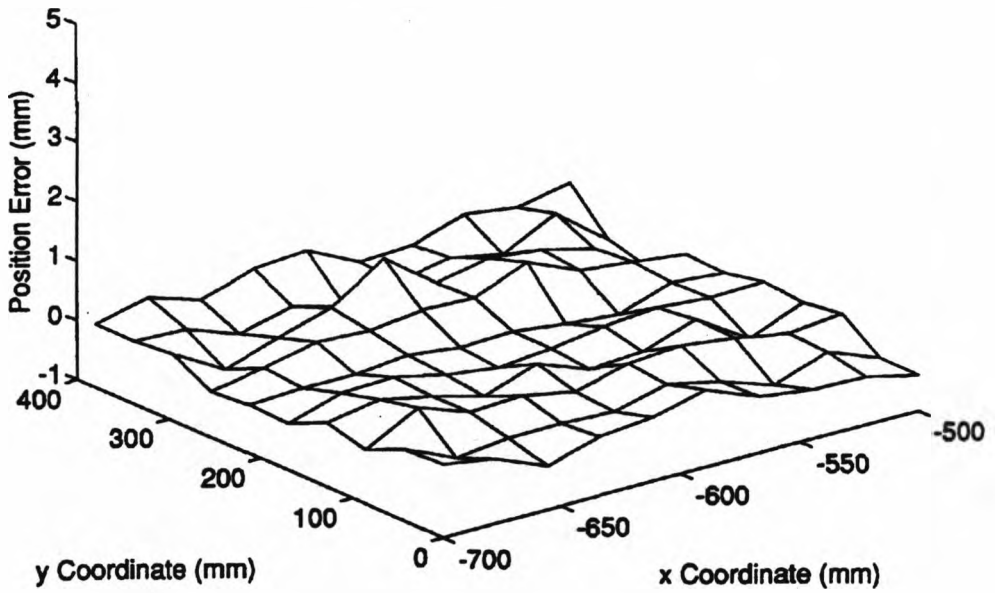


Figure 5.7 Test Result with Experimental Data



(a). z-axis Constraint Plane Perceived by the Robot Before Calibration



(b) z-axis Constraint Plane Perceived by the Robot After Calibration

Figure 5.8. z-axis Constraint Plane Perceived by the Robot

5.5.3 Cross-Evaluation using CMM

In the above sections, we have shown through simulation and experimentation that a robot can calibrate its kinematic control model using the measurements of its *internal sensors* only. However, for the evaluation of the calibration results, the internal sensor measurements are inappropriate and some external global measurements are needed. Furthermore, although we show through experimentation that the positioning accuracy of the test points on the constraint planes are improved, it needs to be verified that the positioning accuracy of those points beyond the constraint plane are also improved. Therefore a precision co-ordinate measuring machine (CMM) was used to obtain the actual locations achieved by the robot end-effector. A measuring cube was mounted in the end-effector to obtain the full pose information of the end-effector as described in Chapter 3. Two hundred and eighty eight points uniformly distributed in a volume of 200 (mm) by 400 (mm) by 200 (mm) in Cartesian space and 45 (degrees) by 90 (degrees) by 135 (degrees) in orientation space were collected. The measurement volume was located near the area where the constraint plates were placed. Since the actual measurements of the CMM were with respect to a reference point, the reference point was calibrated before the collected data were used. The calibration of the reference point could be considered as the robot base calibration and was performed by minimising the total inaccuracy of the whole 288 points in the least square sense. To evaluate the robot kinematic model before and after calibration, the effects of the robot base and the tool were eliminated, since the robot was set-up at different time for calibration and for evaluation and used the different end-effector tools.

Both the actual measured positions and orientations of the end-effector were compared with the locations reported by the kinematic model without calibration, and by the kinematic model updated using the identified errors of Table 5.5. The comparison results are listed in Table 5.8 which are based on twenty test points randomly chosen from the whole data set. After compensating robot base error, the average position error in length was decreased from 3.75 (mm) to 2.15 (mm). The average error decreased further from 2.15 (mm) to 0.76 (mm) after robot kinematic calibration. The standard deviation decreased from 1.15 to 0.19. Although only position information was used in calibration, the orientation accuracy was improved as well due to the robot wrist parameters being calibrated using the tool offset. The accuracy improvement of the cross-evaluation is encouraging, considering that the set-up for

calibration and evaluation was changed and the evaluation volume was beyond the constraint planes used for calibration.

Table 5.8 Cross-Evaluation Results Using CMM

	Before Calibration		After Base Calibration		After Base & Robot Calibration	
in length	Pos.(mm)	Ori.(deg.)	Pos.(mm)	Ori.(deg.)	Pos.(mm)	Ori.(deg.)
average	3.7509	2.5743	2.15	1.57	0.76	0.84
stdev.	1.1506	0.4227	0.52	0.42	0.19	0.37
max.	4.7437	3.1496	2.86	2.45	1.14	1.41

5.6 Chapter Summary and Discussions

A new autonomous robot calibration scheme has been developed in this chapter. *Renishaw probes* were originally used for workpiece set-up and measurements for CNC lathes. We applied these cost effective sensors successfully for robot on-site calibration in an industrial application environment. Instead of taking partial or complete pose measurements for robot calibration, the tip-point of the probe was constrained to a plane movement and only robot internal joint measurements were used for kinematic identification. Neither external measurements nor accurate fixture set-up are needed for such a calibration. The recurrent neural network-based parameter identification algorithm is used for calibration processing. Both simulation and experimental results for a Puma robot show that robot positioning accuracy can be improved to the level of robot repeatability.

The six dimensional robot kinematic error model was projected into one positional dimension in this study. The full kinematic parameters can be calibrated as long as the inaccuracy function of the specific dimension component contains all the kinematic parameters. If the specific component is not sufficient to identify all kinematic parameters, more than one constraint plane can be placed in different positional axis

separately for the data collection. Proper tool offsets are also needed to make the robot wrist parameters identifiable. The general kinematic identification model does not require the exact knowledge of the constraint plane locations but the model can only identify small kinematic errors. The orientation knowledge of the constraint planes (aligned with robot base planes) enable the model to identify reasonably large kinematic errors for practical applications. The alignment of the constraint plane with the robot base axis can be easily performed with the robot and a trigger probe. Future work will investigate the optimal placement of the constraint plane in the constrained environments so that the robot has the optimal identification configurations. The study could lead to the construction of a portable mechanical fixture for robot on-site calibration. Constraint surfaces other than planes may also be suitable for the proposed calibration method (such as a spherical ball) as long as the surfaces have known shapes and are suitable for touching with the probe. The effect of measurement noises such as the flatness of constraint planes on identification accuracy need to be studied by simulation in future work.

CHAPTER 6

GENERIC ACCURACY MODELLING USING FEEDFORWARD NEURAL NETWORKS

6.1 Introduction

Accuracy modelling is an important aspect for robot calibration which uses some mathematical tools to represent the functional relationship between robot end-effector positioning error and model parameter errors of each individual link. Either analytic or numerical modelling techniques are used to estimate robot end-effector accuracy as a function of robot joint configurations. In previous chapters, robot end-effector inaccuracy was modelled as an analytic function of robot kinematic or geometric errors. If geometric error modelling is not sufficient, non-geometric errors such as transmission error, compliance, gear backlash, etc. can be added into the model (Whitney, Lozinski and Rourke, 1986; Judd and Knasinski, 1990) and the calibration technique developed remains valid. Another objective of the study of robot accuracy modelling is to understand the interactions between errors which is useful for diagnosing various error sources. To diagnose robot error sources is difficult since different errors in individual links may cause identical errors in the robot end-effector.

As argued by Everett (1993), however, existing geometric and non-geometric error models are ad-hoc and are not suitable for investigating the causes of errors. Existing models are not complete and are unable to model all the phenomena that contribute to robot end-effector inaccuracy. The assignment of non-geometric parameters in the models are ad-hoc since the importance of non-geometric factors in affecting the

accuracy vary from one particular robot to another, and the added non-geometric parameters may have dependency on the existing model parameters which might lead to invalid identification results. Instead, a general error function for each parameter, based on the expansion of Fourier series, was proposed by Everett (1993). The error function provides a more systematic way to represent both geometric and non-geometric errors than previous models. While the general error function is useful for analysing the contributions of various errors, and understanding the error characteristics of a robot, the size of the general error model makes it impractical for calibration purposes because the terms that need to be identified are numerous and the identification algorithm becomes intractable. No numerical identification algorithm is given for the general error model in Everett's work.

In this chapter, a generic accuracy function is defined based on the concept of Everett's error function. Instead of defining each parameter error using the expansion of Fourier series, the generic accuracy function defines robot end-effector inaccuracy by a series of trigonometric functions of joint variables expanded by Fourier series. The accuracy function is generic in that it accounts for various error sources and it can apply to any type of robot. The functional relationship between end-effector inaccuracy and joint variables is useful for the design of robot accuracy compensators using feedforward neural networks. Multi-layered feedforward neural networks are known to be universal function approximators which are theoretically capable of approximating any continuous function to an arbitrary accuracy. Regarding robot joint configuration as inputs and end-effector inaccuracy as outputs, multi-layered feedforward neural networks can be used to learn the non-linear mapping between robot joint configuration and end-effector inaccuracy. However, the design of such a network is not straightforward due to the complexity of the robot accuracy problem, which involves multiple joints and various error sources. The generic accuracy function serves as the basis for the design of the neural network architecture. While maintaining the completeness of Everett's general error model, the neural network learning technique provides a practical solution to the calibration problem using the generic accuracy model.

In the next section, we explain that robot end-effector inaccuracy can be represented by a series of trigonometric polynomials of robot joint variables through the discussion of the error function, therefore an expansion of Fourier series can be employed as a generic accuracy function for completeness. In section 6.3, a higher-order neural

network architecture-Pi-sigma network (Ghosh and Shin, 1992)-is introduced. The Pi-sigma network is capable of generating a higher-order polynomial representation efficiently and dynamically, which provides potential for practical implementation of the generic accuracy model. To illustrate this, Section 6.4 takes a one degree-of-freedom manipulator as an example which shows that the trained network is equivalent to the simulated analytic higher-order error function. In Section 6.5, experimental data for a six DoF Puma robot is used for the network training, and compared with the network training using the commonly used back-propagation algorithm.

6.2 A Generic Accuracy Function

Typical error sources for robot inaccuracy can be classified into two types; geometric errors and non-geometric errors. Geometric errors are normally defined as static parameter deviations which are constant for all robot configurations. Kinematic parameter errors discussed in the previous chapters are geometric errors. Non-geometric errors are dependent on the robot configurations (model terms are functions of the joint variables). Everett (1993) represents both geometric and non-geometric errors systematically as an error function which is based on the expansion of Fourier series. The configuration-independent geometric errors can fit exactly the error function when the function only contains the zero'th order of joint variables. A similar fit can be achieved for the configuration-dependent non-geometric errors although the error function order may vary. For example, typical non-geometric errors such as gear transmission error, compliance and gear backlash can be modelled as robot joint offset errors which are trigonometric functions of the joint variables (The detailed expressions of the model are referred to Whitney, Lozinski and Rourke, 1986). The trigonometric function of the actual joint values will therefore contain terms which are trigonometric functions of the trigonometric function of the joint variables, such as $\cos(p\sin(\theta))$, $\sin(p\cos(\theta))$, etc, where p is a coefficient constant and θ is joint variable. Using Taylor series expansion and some trigonometric operations, such functions like cosine of sine can be expressed as a series of higher-order trigonometric function of joint variable. For example:

$$\cos(p\sin(\theta)) = A_0 + A_1\cos(2\theta) + A_2\cos(4\theta) + \ldots \qquad (6.1)$$

where A_0, A_1, and A_2,... are coefficient constants in the series form of the coefficient p.

Based on the above observation, model parameter errors (both geometric and non-geometric) can be described by the error function which is defined by a Fourier series. The error function is applied to the standard homogenous transformation as an ordinary kinematic parameter, it is similar to the kinematic error except that it allows the error to change with joint variables. However, the development of the relationship between end-effector error and each error function by expanding the function is impractical for multi-degree of freedom robots, because the size of the model grows exponentially with the number of DoF and the order of the error function. Figure 6.1 illustrates the relationship between robot end-effector error $\Delta\mathbf{T}$ and each link transformation error $\Delta\mathbf{A}_i$ for a six DoF robot (i = 1,2,..., 6), where $\mathbf{A}_i$ is the standard Homogenous transformation for link i which is the first order trigonometric function of joint variable θ_i as defined in Chapter 3, $\Delta\mathbf{A}_i$ contains error functions which may be expanded to a certain order of the trigonometric functions of joint variables as discussed above. If $\Delta\mathbf{A}_i$ is purely geometric then it is the zero'th order of the joint variables, the model of the end-effector error $\Delta\mathbf{T}$ can be established as described in Chapter 3, which is however the sixth order of the trigonometric functions of joint variables. The higher-order terms of the trigonometric functions in $\Delta\mathbf{T}$ are developed due to the fact that individual link errors are transmitted through each link transformation to the end-effector error, and there is coupling effect between link transformations. When $\Delta\mathbf{A}_i$ is expanded to the higher-order of the trigonometric functions of the joint variables, the order of end-effector error $\Delta\mathbf{T}$ in terms of the trigonometric functions of the joint variables will grow considerably. For example, if $\Delta\mathbf{A}_i$ is expanded to the second order of the trigonometric functions of joint variables, the order of $\Delta\mathbf{T}$ will be 2^6 (64) due to the multiplicative transmission of the errors.

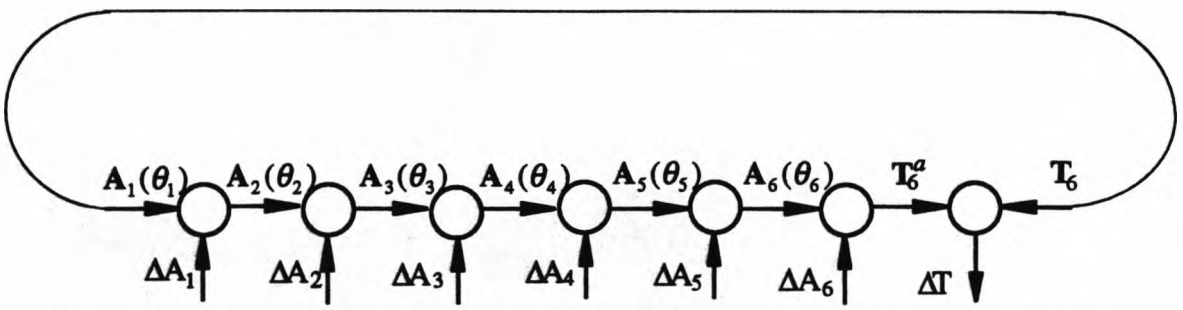


Figure 6.1. Transmission of Individual Link Transformation Errors to End-Effector Error

Through the above discussion, we see that robot end-effector errors due to link parameter errors (represented by an error function) can be represented by a series of trigonometric functions of joint variables with varying orders. For the purpose of robot accuracy compensation, it would be very useful to develop a generic accuracy model that establishes the functional relationship between robot end-effector error and joint variables, taking all the error sources into account. Similar to the definition of the error function which describes each parameter error using an expansion of the Fourier series, we define a generic accuracy function which defines robot end-effector positioning error (inaccuracy) by a series of trigonometric functions of joint variables expanded by Fourier series. For example, the generic accuracy function for end-effector error of a one degree of freedom robot can be written as

$$\Delta x_e = a_0 + \sum_{l=1}^{m} a_l \cos(l\theta) + b_l \sin(l\theta) \tag{6.2}$$

where θ is the joint variable and m is the highest order of error; Δx_e represents the e-th component of the robot end-effector error (inaccuracy). For rotary type joints, θ is the joint angle; for prismatic type joints, conversion of linear movement into rotary angle $\theta = q*2\pi/L$ is applied so that the generic accuracy function can be used for prismatic type joints, where q is the joint variable and L is the joint range of the prismatic joint. If m is equal to zero, the end-effector inaccuracy is equal to a constant which can be interpreted as a constant error existing at the robot base. If only purely geometric errors exist, then the first order expansion is sufficient for a one DoF robot to represent the end-effector inaccuracy due to the geometric errors. The kinematic accuracy model based on geometric errors as developed in Chapter 3 is a special case of the generic accuracy function. The generic accuracy function can be expanded incrementally to accommodate various orders of error function and number of DoF.

For a multi-degree of freedom robot, the generic accuracy function becomes rather complicated since it is a multi-dimensional Fourier series. The basis functions of a multi-dimensional Fourier series expand significantly with the degree of freedom and the order of the function. For example, the basic trigonometric system for a two dimensional series are as follows:

$$\begin{aligned} &1, \cos(m\theta_1), \sin(m\theta_1), \cos(n\theta_2), \sin(n\theta_2), \cos(m\theta_1)\cos(n\theta_2), \ldots \\ &\sin(m\theta_1)\cos(n\theta_2), \cos(m\theta_1)\sin(n\theta_2), \sin(m\theta_1)\sin(n\theta_2), \ldots \\ &(m = 1, 2, \ldots; n = 1, 2, \ldots) \end{aligned} \tag{6.3}$$

The structure of the generic accuracy function provides useful information for the design of a robot accuracy model which attempt to approximate the functional relationship between end-effector inaccuracy and joint variables. In the following sections, a feedforward neural network is used to model the accuracy function.

6.3 Neural Network Architecture and Learning Algorithm

6.3.1 Neural Network Architecture

The involvement of neural networks in this case is to construct the non-linear mapping between robot joint space and end-effector inaccuracy in Cartesian space using a NN internal representation. Multi-layered perceptron (MLP) neural networks are capable of approximating any continuous function from one dimensional space to another to an arbitrary accuracy, provided sufficiently many hidden units are available (Hornik, Stinchcombe and White, 1989). However, there are currently no constructive methods available for the design of NN architecture for specific problems. Typically, an internal representation of a MLP network is constructed from a group of first-order units via a learning rule such as back-propagation. The first-order neuron processing units are linear in the sense that they can capture only first-order correlation of inputs. It has been shown that higher-order correlation among input components can be used to construct a higher-order network which exhibits greatly enhanced performance in learning, generalisation and knowledge (symbol) representation (Giles and Maxwell, 1987). This performance is due to the fact that the order or structure of a higher-order neural network can be tailored to the order or structure of a problem, which enables such a network to learn geometrically invariant properties more easily. For example, learning the XOR has been the classic difficult problem for first-order units which requires thousands of iterations of the fastest learning rule to train a hidden unit to perform the XOR function. However, if a second-order correlation term $x_1 * x_2$ between the two inputs x_1 and x_2 is provided, no hidden unit is needed and one iteration of training will converge (Giles and Maxwell, 1987).

From the generic accuracy function as defined above, we can see that robot end-effector inaccuracies are higher-order trigonometric polynomials of joint variables. Therefore, a higher-order neural network which can capture higher-order correlation among input components will be desirable for our application. One straightforward

higher order network can be constructed by using non-linear basis functions, chosen from a *priori* knowledge of the model, as the network inputs, and no hidden layer is needed (such networks are sometimes called functional link networks (Pao, 1989)). A similar network architecture was used by Kawato and Suzuki (1988) for robot inverse dynamics learning where the basis functions were selected from the non-linear terms of dynamics equations. This kind of network has fast and accurate learning ability because it belongs to Widrow-Hoff type linear learning (Mathworks Inc., 1993). Unfortunately, a priori knowledge of models is essential for this kind of network, and the terms required in the input layer become impractically large for higher dimensional and higher-order non-linear mapping problems due to the problem of combinatorial explosion. Take the accuracy modelling problem as an example, if a two DoF robot is considered, the basis functions for network input can be chosen from the generic accuracy function (6.3); if only purely geometric errors are considered, only second-order correlation between the trigonometric functions of joint variables are required, the number of the basis functions is 9. However, the number of basis functions grows exponentially with the dimension of input space and the order of the problem since it belongs to multi-dimensional Fourier series expansion.

A class of higher-order networks, Pi-sigma networks, was introduced by Ghosh and Shin (1992). This network is a fully connected two-layered feedforward network, the weights from a hidden layer to the output are fixed at 1, and only weights between input and the hidden layer are adjustable (Figure 6.2). The Pi-sigma network uses linear summing units in the hidden layer and product units in the output layer to incorporate the approximation capabilities of higher-order networks while greatly reducing network complexity. It has been shown that the size of the network is linear in input size and the order of the network. The total number of adjustable weights for a K-th order Pi-sigma network with N-dimensional inputs is $K*(N+1)$ (Ghosh and Shin, 1992). This also enables the network to be incrementally expandable since the order can be increased by adding another summing unit and associated weights. The highest order of correlation among inputs is equivalent to the number of hidden units used. Figure 6.2 shows a K-th order Pi-sigma network with one output. For multiple outputs, multiple Pi-sigma networks can be used for each component of the output. Due to the higher-order approximation capability and the network simplicity, the Pi-sigma network has been employed to construct a generic model to approximate robot accuracy in this work.

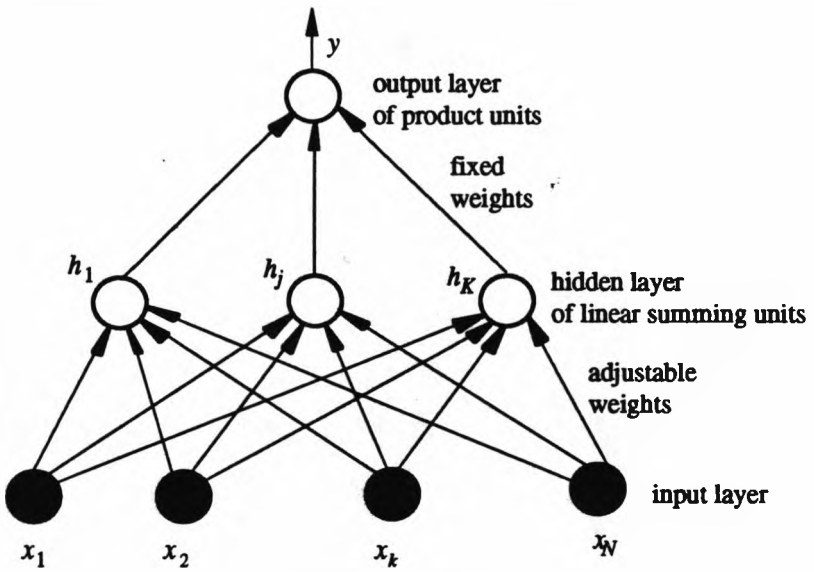


Figure 6.2. A Pi-sigma network with one output (Ghosh and Shin, 1992)

Let $\mathbf{x} = (1, x_1, x_2, ..., x_N)$ be an N+1-dimensional augmented input vector, x_k denotes the $(k+1)$-th component of $\mathbf{x}$. The inputs are weighted by K*$(N+1)$-dimensional augmented weight vectors $\mathbf{w}_j = (b_j, w_{1j}, w_{2j}, ..., w_{Nj})^T$, $j = 1, 2, ..., K$, and summed by a layer of K "linear summing" units where b_j is a bias or the threshold of the jth summing unit and K is the desired order of the network. The output of the j-th summing unit, h_j, is given by:

$$h_j = \mathbf{w}_j^T \mathbf{x} = \sum_{k=1}^{N} w_{kj} x_k + b_j, \qquad j = 1, 2, ..., K \tag{6.4}$$

The output y is given by

$$y = S\left(\prod_{j=1}^{K} h_j\right) \tag{6.5}$$

where $S(\bullet)$ is an appropriate activation function which can be chosen as the sigmoid activation function:

$$S(x) = \frac{1}{1+e^{-cx}} \tag{6.6}$$

or the hyperbolic tangent function:

$$S(x) = \frac{e^{cx} - e^{-cx}}{e^{cx} + e^{-cx}} \tag{6.7}$$

where c is a coefficient constant. If no non-linear activation function is used in the output layer, the network output is actually a higher-order polynomial of input variables. Pi-sigma networks without non-linear activation function are used as the basic building blocks to construct Ridge Polynomial networks which are capable of representing any multivariate polynomial (Shin, 1992). Since our target is to represent the generic accuracy function, no non-linear activation function is used in our network and the inputs are pre-processed using non-linear trigonometric transformations, and therefore the output is the K-th order trigonometric polynomial of robot joint variables. The network learning algorithm provides a natural computational mechanism for the generic accuracy modelling problem.

6.3.2 Network Learning Algorithm

The theoretical proof of Pi-sigma network approximation capability and learning stability has been given in (Ghosh and Shin, 1992). For efficient and stable learning, the asynchronous updating rule has been adopted (Ghosh and Shin, 1992). That is, at each learning epoch, only weights associated with one hidden unit will be chosen to be updated at one time, this process is repeated until all weights associated with each hidden unit are updated once. The weight updating is based-on a LMS (least mean square)-type learning rule, and the batch learning is used to speed up the training process. To derive a network weight updating rule based on the gradient descent algorithm, the network objective is constructed as the sum squared error (SSE) function:

$$\varepsilon^2 = \sum_{p=1}^{Q} (t^p - y^p)^2 \tag{6.8}$$

where superscript p denotes the p-th training pattern, t^p is the desired output for the p-th pattern, y^p is the network output, and summation is over all Q training patterns.

Let the l-th hidden unit be selected for updating, applying gradient descent on the selected weights $\mathbf{w}_l$, we have:

$$\Delta w_{lk} \propto -\frac{\partial \varepsilon^2}{\partial w_{lk}}, \qquad k = 1, 2, ..., N \tag{6.9}$$

Using (6.7) and (6.4) in (6.8), and noting that no non-linear activation function is used in the output layer, we obtain the weight updating rule as follows:

$$\Delta w_i = \eta . \sum_{p=1}^{Q} (t^p - y^p).(\prod_{j \neq i} h_j^p).\mathbf{x}^p \tag{6.10}$$

$$\Delta b_i = \eta . \sum_{p=1}^{Q} (t^p - y^p).(\prod_{j \neq i} h_j^p) \tag{6.11}$$

where t^p is the desired output and y^p is the network output, h_j^p is the hidden unit output and $\mathbf{x}^p$ is the input vector, for the p-th training pattern respectively. All Q training patterns are applied simultaneously to determine the weight changes. The learning rate η is chosen to be a small valued number and changes adaptively according to the sum square error (adaptive learning rate). The learning algorithm for the Pi-sigma network is implemented using MATLAB neural network toolbox (Mathworks Inc., 1993) running on a Hewlet-Packard 9000 workstation.

6.4 Simulation Example for a One DoF Manipulator

Simulation for a one DoF manipulator (Figure 6.3) is performed to illustrate how a Pi-sigma network can be tailored to realise a generic accuracy model. The robot end-effector position (x, y) is dependent on its joint variable θ, for example:

$$x = l * \cos(\theta) \tag{6.12}$$

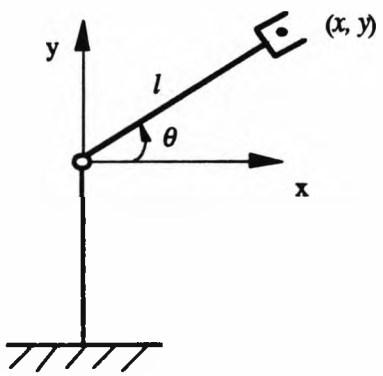


Figure 6.3. One Degree of Freedom Manipulator

The end-effector inaccuracy dx , due to its link geometric errors dl and $d\theta$, is given as:

$$dx = dl*\cos(\theta) - d\theta*l*\sin(\theta) \tag{6.13}$$

The inaccuracy model (6.13) due to purely geometric error is the first-order trigonometric function of joint variable. Assuming the second-order term $p*\sin(2\theta) = 2p*\sin(\theta)*\cos(\theta)$ due to non-geometric errors, and letting coefficient parameter $p = 0.025$, kinematic parameter $l = 10$, $dl = 0.3$, $d\theta = 0.02$, the analytic expression of the inaccuracy model dx is:

$$dx = 0.30\cos(\theta) - 0.2\sin(\theta) + 0.05\sin(\theta)\cos(\theta) \tag{6.14}$$

To realise a second-order trigonometric polynomial (6.14), a second-order Pi-sigma network is designed as shown in Figure 6.4. The input is joint variable encoded by trigonometric functions. Two hidden units are employed to generate second-order correlation and one output unit without activation function is used. The training patterns comprise of 25 pairs of data generated by using the analytic model (6.14) which are uniformly distributed in the joint variable range $[-\pi/2, \pi/2]$. After about 1000 epochs training using the learning algorithm presented above, the network converged with the final RMS (root mean square) error in the order of 10^{-6}. The trained network connection weights are shown in Figure 6.4. Due to the small size of the network, we can expand the trained network analytically:

$$dx = 0.0+ 0.2999\cos(\theta) - 0.2\sin(\theta) + 0.05\sin(\theta)\cos(\theta) + 0.0001\cos^2(\theta) + 0.0\sin^2(\theta) \tag{6.15}$$

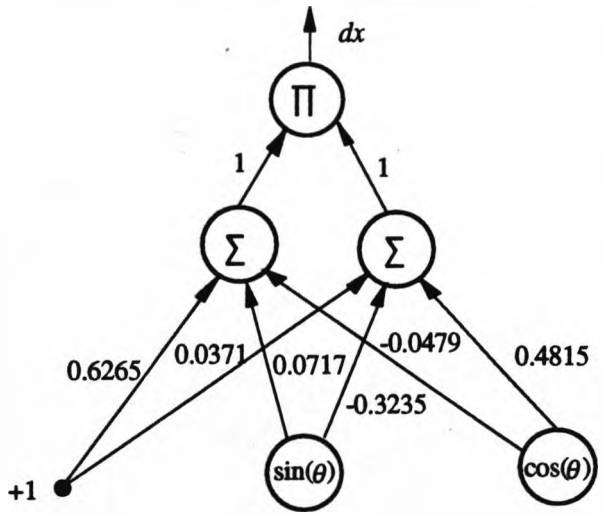


Figure 6.4. NN Representation of Generic Accuracy Model for a One DoF Robot

Comparing (6.15) with (6.14), we can see that the achieved second-order trigonometric polynomial is very close to the simulated accuracy model. Several observations can be made from this simple example:

- It has been shown that the Pi-sigma network can be tailored to represent the generic accuracy function, with pre-processed inputs by using trigonometric functions, and hidden units being selected to fit the order of problem and robot DoF.

- The capability of this simple network architecture of realising higher-order trigonometric polynomial is attractive for our application since it suits the structure of the problem which is represented by a series of trigonometric polynomials. The network training is fast due to only one layer of connection weights being modified and the small network size needed.

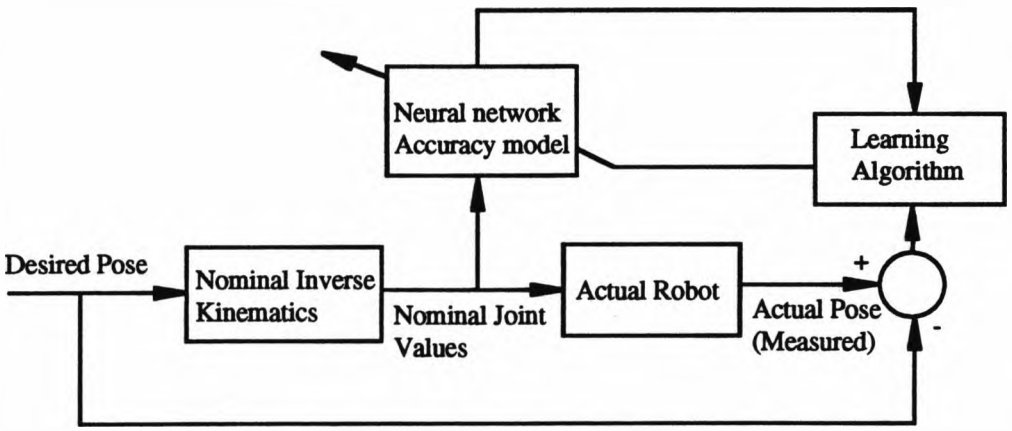
- Instead of identifying the numerous error source items explicitly, the error source information is represented by the distributed network connection weights. The NN learning method provides a natural computation mechanism for the generic accuracy modelling.

- Using the incremental learning algorithm as suggested by Shin (1992) for realising higher-order multi-variate polynomials, the size and the order of the network can grow incrementally by adding a higher-order Pi-sigma network without affecting the established network connections.

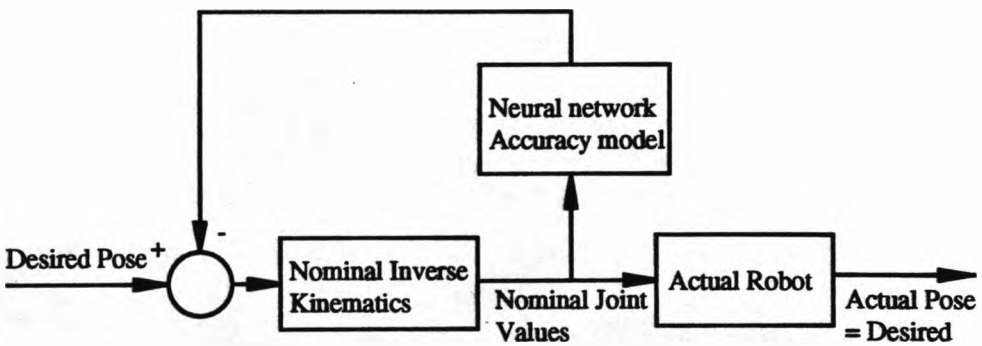
Explicit evaluation realising an exact accuracy model for a multi-DoF robot is difficult due to multiple input variables involved. The training accuracy for multiple input variable network can not be achieved as well as in the example due to the basic learning algorithm (LMS-type) adopted, which stuck to local minima easily for multiple-variate non-linear optimisation problems. Advanced optimisation methods such as simulated annealing, or genetic algorithm (GA) (Masters, 1993) can be incorporated into the learning algorithm to escape from local minima, which however is not the focus of this study.

6.5 Network Training Using Experimental Data

The end-effector inaccuracy data in the local calibration volume was collected for a six DoF PUMA robot using the CMM experimental set-up described in Chapter 4. The neural network was used to construct a robot accuracy model which relates robot joint configurations to end-effector inaccuracy, based on the training samples from the collected data. The trained network was then used to predict the end-effector inaccuracy, given joint configurations determined by the nominal inverse kinematics. The outputs of the trained network were used to modify the desired pose so that the actual pose achieved, by controlling joint values recommended by the nominal inverse kinematics, are close to the desired pose. The schematic of training and implementation of the NN-based accuracy model is shown in Figure 6.5.



a). Training of Neural Network Accuracy Model



b). Implementation of the Trained Network

Figure 6.5. Training and Implementation of NN Accuracy Model

As shown in Figure 6.6, a single Pi-sigma network has been used for each component of the pose inaccuracy vector, with the same inputs of joint variables encoded by sinusoidal functions. Note that only sinusoidal transformations are used in the input layer instead of using both sine and cosine transformations for each input component as in the simulation. This is because the input space is highly-dimensional (six dimensions); if both sine and cosine transformations are used in the input layer, the dimension of the input space will be doubled. Experiments show that network architecture with lower dimensions of input space outperform those with higher dimensions in this case. The product units have no activation functions and are used in the output layer so that the outputs are actually higher-order polynomials of sinusoidal functions of joint variables. The order of the polynomials is equal to the number of units used in the hidden layer of each Pi-sigma network. For the six DoF puma robot, our experiments show that six units in the hidden layer achieves best results. This agrees well with analytic kinematic accuracy models which are generally up to sixth order trigonometric polynomials of joint variables. Figure 6.7 shows the positional accuracy learning curve which exhibits fast and stable learning. The particular learning parameters are chosen for each component of output vector learning. The trained networks are then put together for implementation to obtain multiple outputs simultaneously.

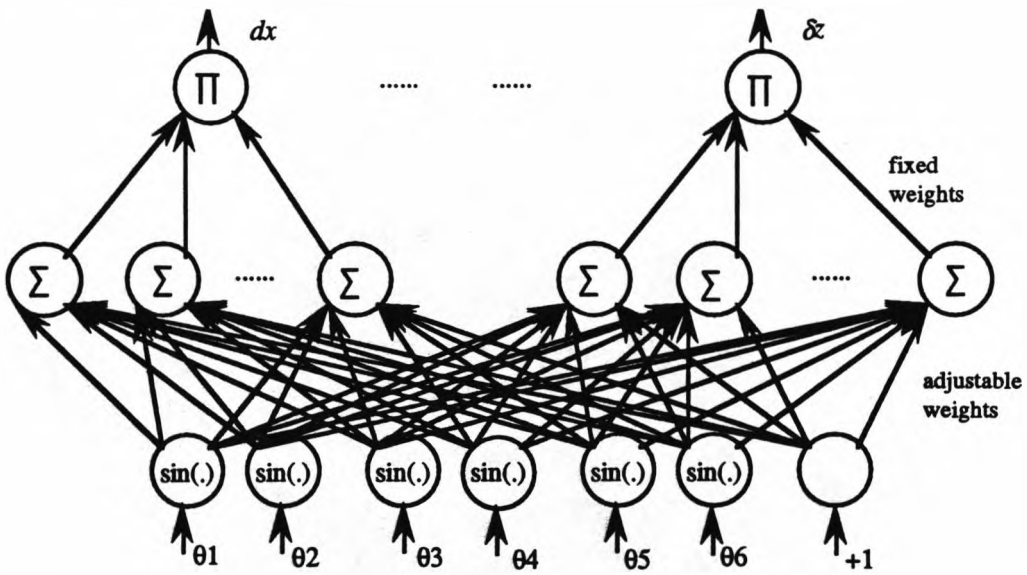
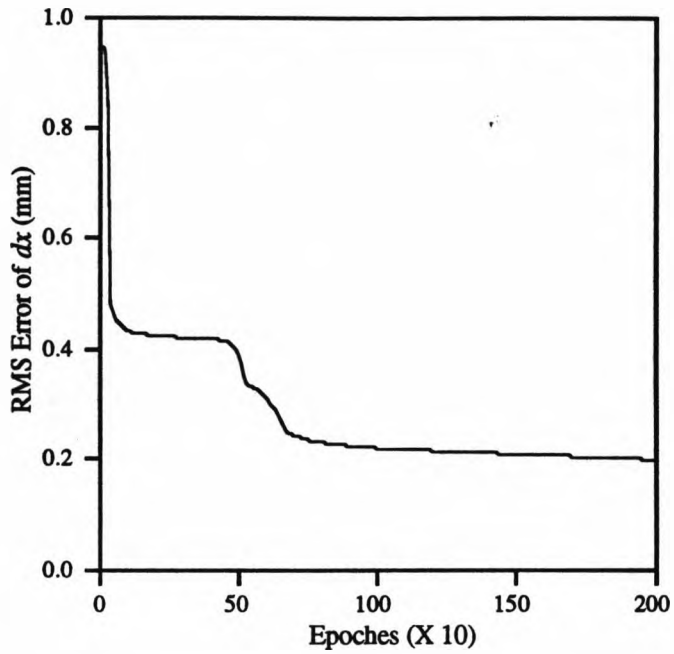
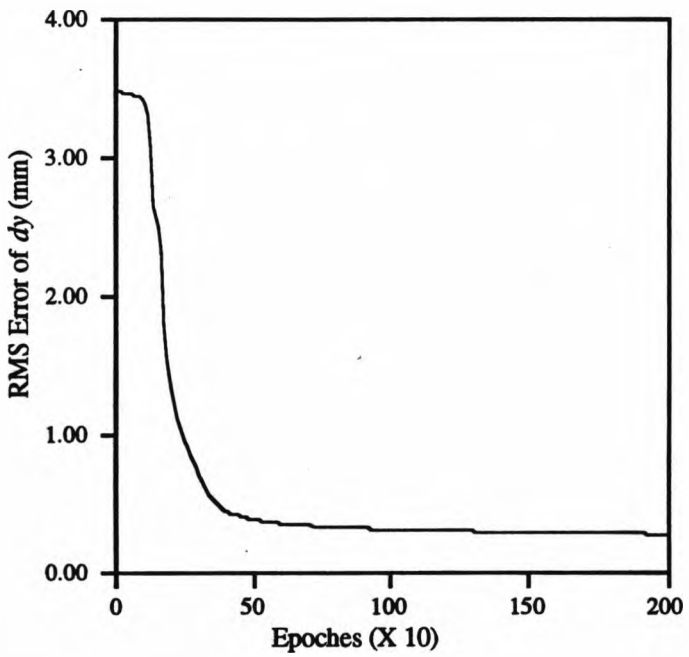


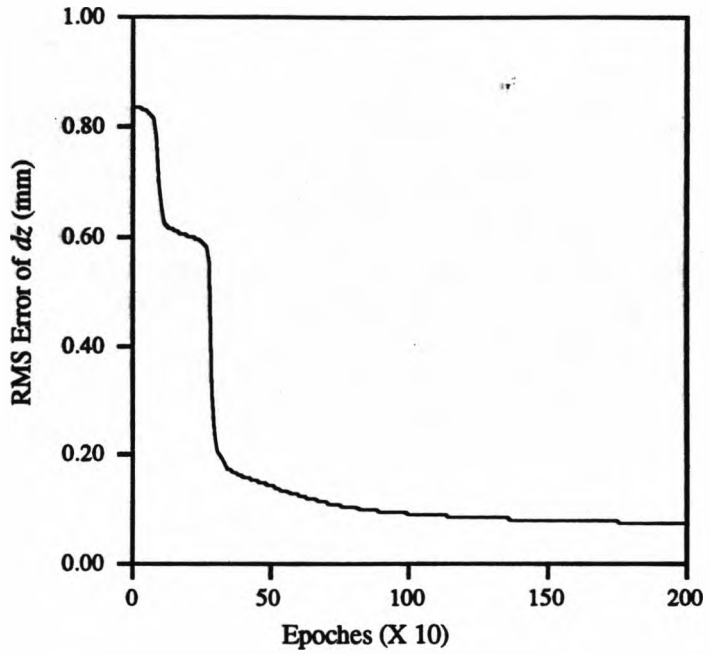
Figure 6.6. Neural Network Architecture for Accuracy Modelling



(a). The *x*-component of accuracy learning curve



(b). The *y*-component of accuracy learning curve



(c). The *z*-component of accuracy learning curve

Figure 6.7. Learning Curves for Positional Accuracy Modelling

Half of the collected 288 data points were used for network training and the remaining 144 points were used for evaluation. Figure 6.8 shows the neural network generalisation test of the positional accuracy modelling for the 144 test points (sorted in ascending order). It is shown that the network can predict robot inaccuracy well in the calibrated area even for the points unseen in the network training patterns. Three statistical measures (average error, standard deviation and maximum error) were used to evaluate the achieved accuracy by the trained network. Table 6.1 lists both position and orientation residual error (learning error) achieved by the trained network based on the 144 test points not included in the training data. Comparing Table 6.1 with the achieved accuracy using kinematic calibration in Chapter 4 (Table 4.6), it can be seen that the NN-based generic accuracy model achieves the same level of positional accuracy as the kinematic calibration, and achieves better orientation accuracy than kinematic calibration. It shows that kinematic model-based calibration can compensate the non-geometric errors in the local calibration volume, since kinematic modelling achieves the same level of positional accuracy as the NN-based generic accuracy

modelling. The better orientation accuracy achieved by the NN model is partially due to the fact of there being independent Pi-sigma networks for each component of pose error vector, and that there are more adjustable parameters (weights) in the NN model than for the kinematic model. The NN-based model uses in total 6*6*(6+1) = 252 adjustable parameters (weights), in comparison with the total 30 adjustable parameters used in the kinematic model.

For comparison, a standard feedforward neural network using a back-propagation learning algorithm (Hecht-Nielsen, 1990; Masters, 1993) was used to approximate the same accuracy model. The input data are also encoded by sinusoidal functions. A three layered network of 6 x 30 x 3 were used for positional and orientation inaccuracy approximation respectively[6]. The hyperbolic tangent activation functions were used in the hidden layer units and linear activation functions are used in the output units. Several network training heuristics such as learning with momentum, Nguyen-Widrow initial conditions, adaptive learning rate (Mathworks, 1993) were used to improve the back-propagation learning. After about 1500 epochs training using 144 training data, the network converged to the desired level of learning accuracy. Accuracy evaluation for the trained back-propagation network based on the remaining 144 data points are shown in Table 6.2. Comparing Table 6.2 with Table 6.1, it shows that the back-propagation networks achieved the same level of accuracy as the Pi-sigma networks. However, the total number of adjustable weights in the back-propagation networks is 2*(30*(6+1)+30*3) = 600, in comparison with the 252 adjustable weights used in the Pi-sigma networks. The computation required in the back-propagation networks is much more intensive than the Pi-sigma networks due to the larger network size and the use of non-linear hyperbolic tangent functions in the hidden units, compared with only one layer of connection weights to be modified and no non-linear activation function used in the Pi-sigma networks.

[6] Since there is no constructive methods for network design, the network architecture is decided based on extensive numerical experiments.

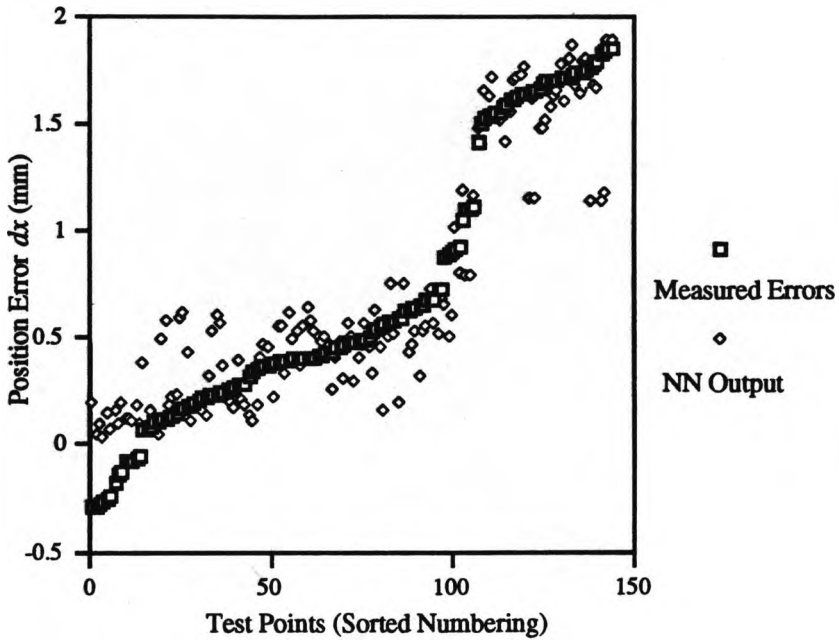


Fig. 6.8 (a) *dx* Generalisation Test

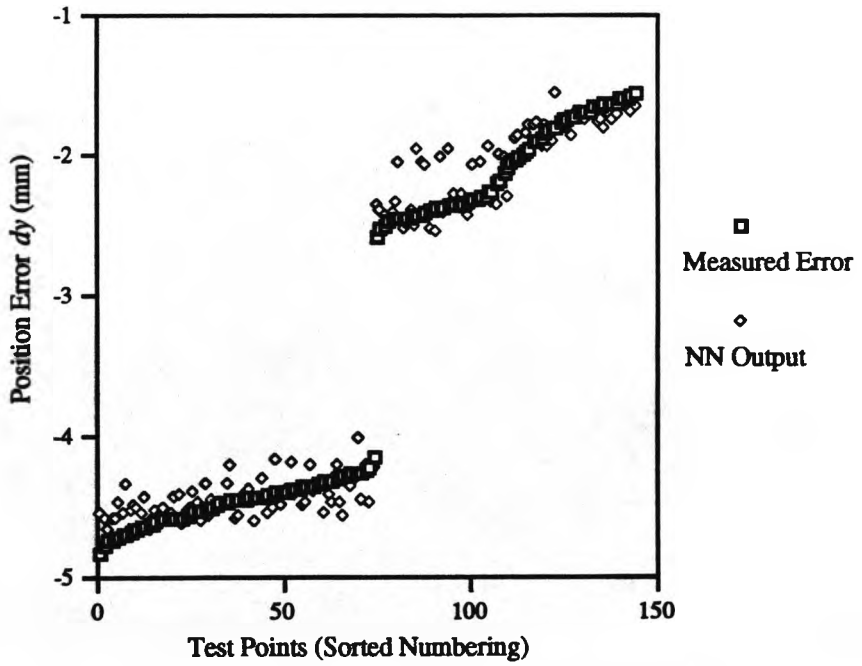


Fig. 6.8 (b) *dy* Generalisation Test

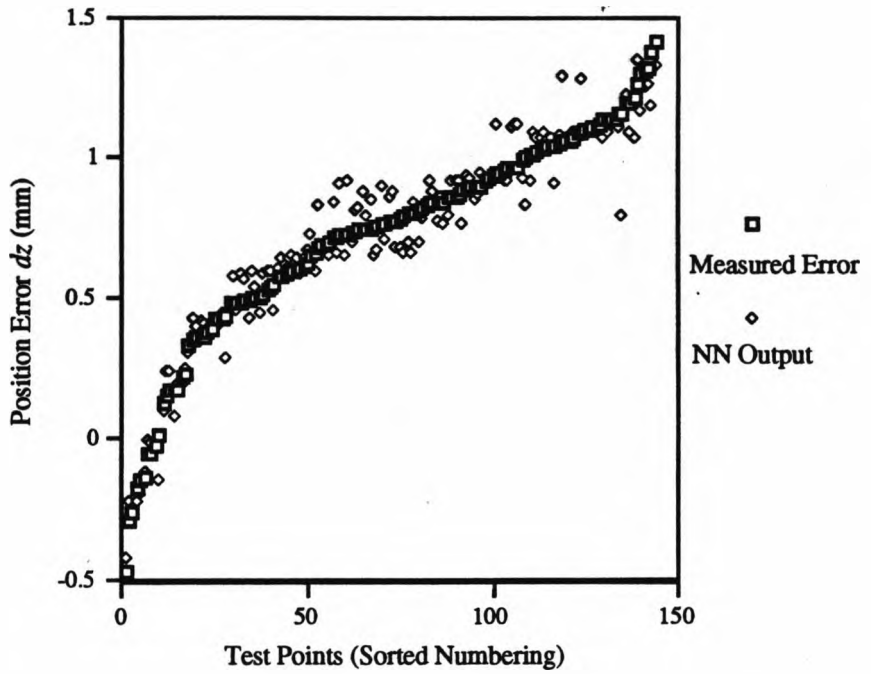


Fig. 6.8 (c) *dz* Generalisation Test

Figure 6.8. NN Generalisation Test for Position Compensation

Table 6.1. Accuracy Evaluation for Pi-sigma Network Based-on Test Pints

	Position Error in Length	Orientation Error in Length
average	0.2411 (mm)	0.4419 (degree)
std. dev.	0.1488 (mm)	0.3744 (degree)
max.	0.7774 (mm)	1.9460 (degree)

Table 6.2. Accuracy Evaluation for Back-Prop. Network Based-on Test Points

	Position Error in Length	Orientation Error in Length
average	0.2383 (mm)	0.4249 (degree)
std. dev.	0.1450 (mm)	0.3494 (degree)
max.	1.0140 (mm)	1.4229 (degree)

6.6 Chapter Summary

A generic accuracy function which accounts for various error sources has been introduced. Feedforward mapping neural networks are used to implement the generic accuracy model. The generic accuracy function serves as the basis for the design of the network architecture. Pi-sigma networks, which are capable of representing higher-order non-linear functions using simple network architecture, provide natural computational mechanism for implementation of the generic accuracy function. Instead of identifying various error sources explicitly, the error source information is encoded in the distributed network connection weights. Due to the complex nature of the accuracy problem for multiple DoF robots, the NN representation is appealing because of its learning methodology, robustness and efficiency. However, the NN training accuracy and efficiency will suffer if larger data sets covering larger workspaces for multiple DoF robot are used. Neural network design methodology, which incorporates *a priori* knowledge into network architecture so that the network can capture invariant properties of the problem from high-dimensional data, remains an open research topic.

The generic accuracy modelling problem discussed in this chapter, and kinematic calibration discussed in previous chapters, concentrate on estimating robot end-effector accuracy (exact pose), given robot joint variables readings. This problem is also called robot forward calibration (Shamma and Whitney, 1987). The inverse calibration problem is to determine exact joint variable values, given the desired end-effector pose in robot workspace, which will be investigated in the next chapter.

CHAPTER 7

ROBOT ACCURACY COMPENSATION USING ARTIFICIAL NEURAL NETWORKS

7.1 Introduction

Robot accuracy compensation is a process by which robot pose errors in a workspace are compensated through corrections to the nominal joint variables based on the identified geometric and non-geometric errors. Since robot controllers which accept identified parameter changes are still not widely available, implementation of robot calibration is generally performed through accuracy compensation in robot joint space. Robot accuracy compensation can be regarded as a subset of the inverse kinematics problem, which determines joint variable corrections given robot end-effector pose and the nominal joint values determined by the nominal inverse kinematic model.

There are two approaches to solving the robot accuracy problem; non-parametric and model-based parametric approaches. The non-parametric approach is based on fitting abstract interpolation functions to relate the joint transducer readings in a selected group of robot measurements to the measured pose errors. Such functions can then be used to compute the joint commands correction terms at the application points (a precise inverse kinematic solution is determined from the computed joint correction and nominal joint variables). Shamma and Whitney (1987) used third-order trivariate polynomials as interpolation functions to relate joint variables input and joint corrections output for a three DoF robot. Direct extension of the multi-variate polynomials for a six DoF robot is difficult since the number of polynomial terms

required grows considerably with the number of DoF and the order of polynomials. Approximation functions are valid only in the regions of the workspace where data were taken on which the coefficients of the functions were based. Therefore non-parametric accuracy compensation is a local compensation by nature. Methods of robot local accuracy compensation are discussed in Section 7.2. A local accuracy compensation method based on the Pi-sigma neural network is developed. The Pi-sigma network can generate a multi-variate higher-order polynomial approximation efficiently through NN learning method. The NN-based accuracy compensation has a constant-time solution which is efficient for on-line implementation. Simulation and experimental results of local accuracy compensation are presented in Section 7.2 for a six DoF Puma robot.

Model-based accuracy compensation methods are based on numerical inverse solutions of the calibrated robot. As discussed in Chapter 1, for robots with simple-form kinematics, the nominal inverse kinematics has closed form solutions. However, closed form solutions do not exist for the calibrated robot due to the changes of kinematic structure. Numerical techniques are involved to find the precise inverse solutions for the calibrated non-simple form kinematic model. As to the numerical techniques for solving inverse kinematics problem, the Newton-Raphson (N-R) algorithm is widely used due to its simplicity. Stone (1987) developed numerical inverse kinematics algorithms for the general form kinematic model (calibrated signature model) based on the N-R method and the Jacobi iterative method. A comparative studies of computation complexity of the two algorithms has been performed by Stone (1987). It shows that the N-R algorithm has a quadratic convergence rate while the Jacobi iterative algorithm has a linear convergence rate. The Jacobi iterative method is similar to the differential transformation compensation algorithm as suggested by Veitschegger and Wu (1987), in which an iterative procedure of nominal inverse kinematics is applied until the achieved pose by the identified model is sufficiently close to the desired pose. Model-based compensation belongs to global compensation since it is not limited to specific local workspace. However, numerical compensation algorithms suffer from certain numerical problems such as ill-conditioning and singularities of the Jacobian. More robust compensation algorithms such as Levenberg-Marquardt and linear quadratic regulator algorithm (Zhuang, 1989) can find good solutions in the vicinity of singularities by using regulation terms in the cost functions, but require longer computation time due to the algorithm complexities which make on-line implementation problematic. A recurrent neural network (RNN)

approach to model-based accuracy compensation (Zhong and Lewis, 1994; Zhong, Lewis and N-Nagy, 1995) is developed in Section 7.3 which is both computationally efficient for on-line implementation and robust even at singular configurations. Firstly the N-R compensation algorithm is analysed. The RNN-based accuracy compensation algorithm is then presented. Simulation examples of path compensation and compensation near a singularity are given using the RNN-based compensation algorithm based on the identified kinematic errors for the Puma robot.

7.2 Non-parametric Accuracy Compensation

7.2.1 Accuracy Compensation Using Polynomial Functions

This Section intends to explain the method of non-parametric accuracy compensation using polynomial approximation based on the works by Shamma (1985), Shamma and Whitney (1987) and the review chapter by Mooring, Roth and Driels (1991).

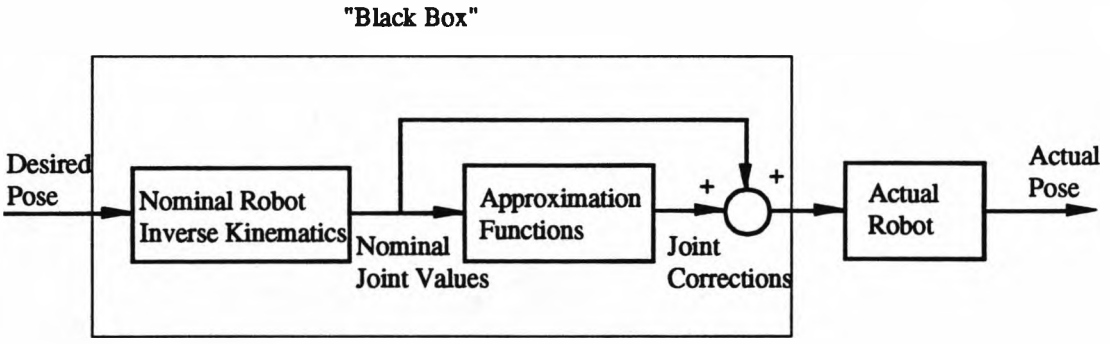


Figure 7.1. Non-parametric Accuracy Compensation (Shamma and Whitney, 1987; Mooring, Roth and Driels, 1991)

The basic idea of non-parametric accuracy compensation is to approach the accuracy problem as a "black box". Figure 7.1 illustrates the non-parametric accuracy compensation scheme. The criteria for the approximation function inside the "black box" were that it be continuous, be able to represent high order functions, and still be implementable in a noisy environment while remaining numerically well behaved. The

first approximation function attempted by Shamma (1985) was the CMAC (Albus, 1975a,b). It was discovered that the CMAC is a discrete (not continuous) linear (unable to represent higher order functions) interpolator therefore is not suitable as an approximation device. Instead, multi-variable polynomials were used as approximation functions. The third-order trivariate polynomials were chosen for a Puma robot comprising of the first three major DoF. Each of the three joint correction δq_i (i = 1, 2, 3) is then represented as the trivariate polynomial function of the three joint variables (q_1, q_2, q_3):

$$\delta q_i = \sum_{r=0}^{3}\sum_{s=0}^{3}\sum_{t=0}^{3} c_{rst}^{i} q_1^r q_2^s q_3^t \tag{7.1}$$

where r, s, t are non-negative integer exponents that satisfy the inequality:

$$0 \le r+s+t \le 3 \tag{7.2}$$

and c_{rst}^{i} is the polynomial coefficient to be determined from the collected training data pairs ($\mathbf{q}$, $\delta\mathbf{q}$). Taking all possible combinations into account, there are in total 20 terms in (7.1) for the third-order trivariate polynomial. The unknowns of the polynomial coefficients are determined through resolving the over-determined linear system formulated by aggregating linear equation (7.1) at different measurement configurations. The selection of measurement configurations is based on the Tchebychev spacing and the polynomials are created to be mutually orthonomal so that the linear least square solutions of the coefficients are numerically efficient and robust (Shamma and Whitney, 1987).

The accuracy compensation procedure for the three DoF Puma robot can be summarised as follows:

Step 1: Define a calibration volume of robot workspace and generate a set of training points via Tchebechev spacing.

Step 2: Construct a set of orthonormal polynomials with the joint encoder angles q_1, q_2, and q_3 as the independent variables.

Step 3: At the above training points find the required joint corrections. To find the joint encoder corrections necessary to drive the manipulator to the desired workspace position involves the following experimental procedure:

3a) Send the robot to the desired position $\mathbf{x}_d$, the corresponding joint encoder reading is $\mathbf{q}_n$ The actual position achieved by controlling joint angles $\mathbf{q}_n$, $\mathbf{x}_a$ is measured using a measuring device.

3b) Manually perturb the joints until the manipulator end point is in the desired training position $\mathbf{x}_d$. Record the joint readings $\mathbf{q}_a$. Then the joint correction is $\Delta\mathbf{q} = \mathbf{q}_a - \mathbf{q}_n$.

Step 4: Solve for the coefficients that give the polynomial the best fit to the training data samples. There will be three such sets of coefficients, one set for each joint.

The procedure was applied to a simulated PUMA robot. Simulations show that the maximum position error improved from 2.5 (mm) before compensation to 0.31 (mm) after compensation within the calibration volume of about one quadrant of robot workspace. Robot positioning errors were simulated using both geometric and non-geometric error models. No experimental results were given in the work by Shamma and Whitney.

Several observations are in order:

1) Direct extension of the method presented for calibrating a 3 DoF manipulator to the general manipulator may be very cumbersome computationally. For instance, the use of third-order, six-variate polynomials requires 84 terms (compared to 20 terms for the third-order trivariate polynomial as shown in the example), thus requiring a very large number of data points[7] and making the computation very complicated. If higher-order polynomials are required, the number of terms grow considerably and the problem may become intractable.

2) The data collection method (*Step* 3) involves manually measuring and teaching the robot, which is time-consuming, tedious and error prone. It is impractical to do so if large number of data points are required.

3) Although the non-parametric accuracy compensation is based on the "black-box" approach, one may still want to benefit from the robot analytic accuracy model as a prime source of useful information. Due to the fact that most of the functions in the analytic accuracy model are trigonometric functions of joint variables (as discussed in

[7] For a six DOF robot using the third-order polynomials, more than 6*84 = 504 data points are needed to satisfy an over determining condition

Chapter 6), it may be more appropriate to use the trigonometric terms of joint variables, rather than directly joint variables as above, as the polynomial variables. The order of the polynomial should match the order of analytic accuracy model.

Based on the above observations, the Pi-sigma neural network, as introduced in the previous chapter, is used as an approximation function for accuracy compensation problem for a six DoF Puma robot.

7.2.2 Accuracy Compensation Using Feedforward Neural Network

For the robot local calibration problem in which robot accuracy is only critical in a small portion of its workspace, a simple feedforward network with higher-order approximation capability is designed to learn the non-linear mapping between robot configurations and joint corrections. As discussed in Chapter 4, 288 data points of a six DoF Puma robot end-effector pose were collected using a CMM. These data points were uniformly distributed in the calibration volume. However, because of the large number of data points required and the contact type measurement method utilised by the CMM, it is impractical to obtain corresponding joint corrections (network training data) to compensate end-effector errors using this manual data collection method (*Step* 3). Alternatively, the actual joint values which drive the robot to minimise the end-effector deviations can be found using non-linear least square optimisation using the six controllable joint angles as optimisation variables. The initial values of joint angles are the nominal joint values from the robot controller. Joint corrections are then the differences between the computed joint values and the nominal ones. The optimisation procedure is as follows.

$$\min_{\mathbf{q}_l}[\mathbf{x}_l - \mathbf{f}(\mathbf{k},\mathbf{q}_l)]^T \mathbf{Q}[\mathbf{x}_l - \mathbf{f}(\mathbf{k},\mathbf{q}_l)] \tag{7.3}$$

where $\mathbf{f}(.)$ is robot forward kinematic model, $\mathbf{k}$ is a constant kinematic parameter vector, $\mathbf{x}_l$is a directly measured end-effector pose vector corresponding the l-th joint configuration vector $\mathbf{q}_l$. $\mathbf{Q}$ is the weight coefficient matrix as defined before. Since there are six adjustment variables to compensate six dimension end-effector pose errors at each configuration, the residual errors after compensation can be very close to zero. The optimisation procedure is equivalent to manually perturbing joint variables so that the end-effector pose is the desired one, but the optimisation procedure is automatic and

time efficient compared with the manual data collection procedure. As we have seen from the kinematic identification (Chapter 4), non-linear optimisation uses a more robust search strategy and can converge to good solutions given sufficient time. However, its convergence is too slow for on-line implementation of accuracy compensation. Therefore, a feedforward neural network is used to store and interpolate the joint corrections obtained from the off-line non-linear optimisation routine. Figure 7.2 illustrates the neural network training and implementation schemes. The trained neural networks are used to augment the robot controller to perform constant-time inverse compensation which is suitable for on-line implementation.

The Pi-sigma network has been employed to approximate robot inverse compensation in this work. As shown in Figure 7.3, a single Pi-sigma network has been used for each compensation vector component, with the same inputs of joint angles encoded by sinusoidal functions. The product units without sigmoid functions are used in the output layer so that the outputs are actually higher order polynomials of sinusoidal functions of joint angles. Six hidden units are used, the Pi-sigma network approximation is equivalent to a sixth order polynomial of six variates. It will be very difficult to determine the sixth order polynomial of six variates numerically if direct polynomial approximation functions are used. This, however, is straightforward if the Pi-sigma network learning method is used.

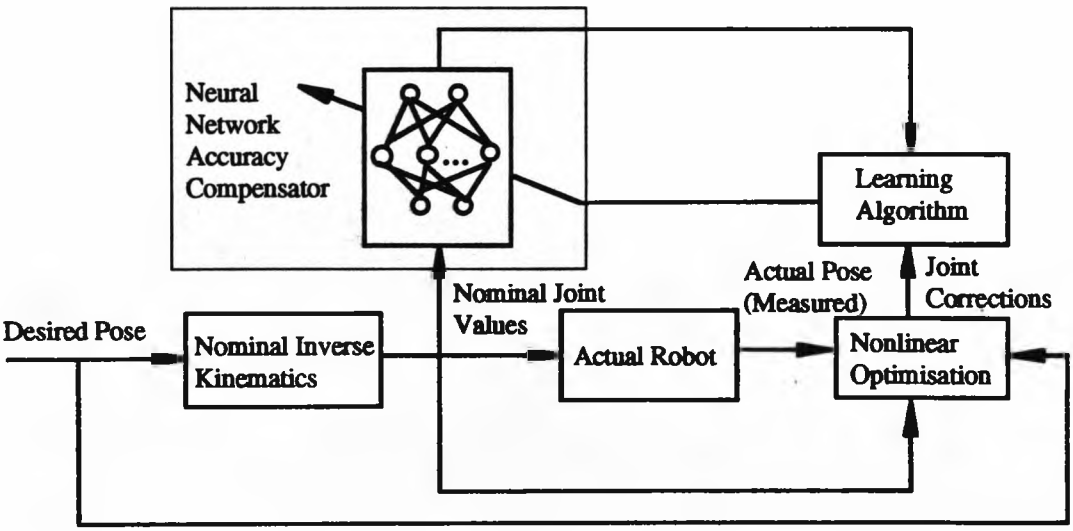
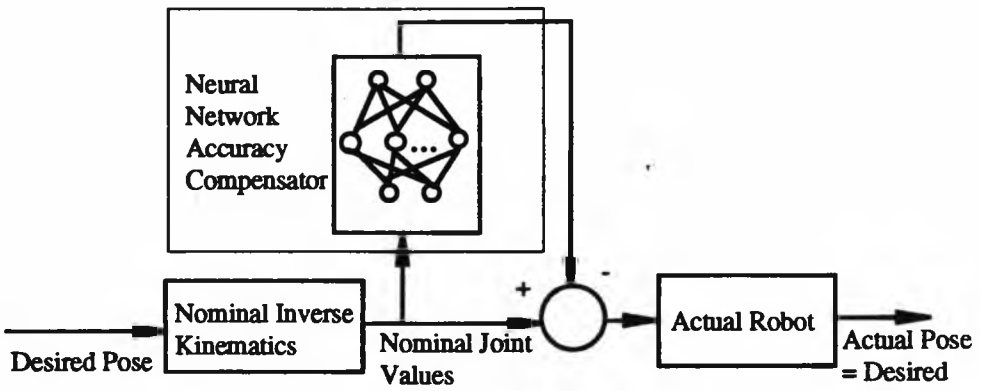


Figure 7.2. a). Training of Neural Network Accuracy Compensator



b). Implementation of the Trained Network

Figure 7.2. Training and Implementation of NN Accuracy Model

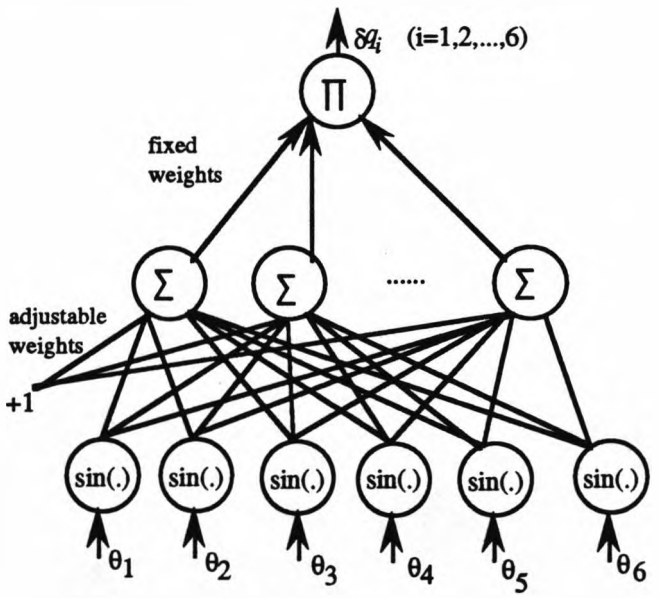
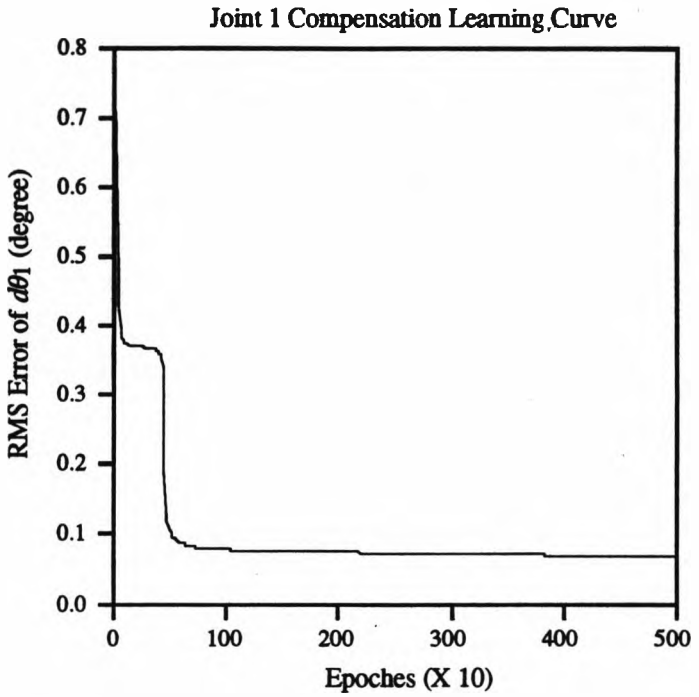
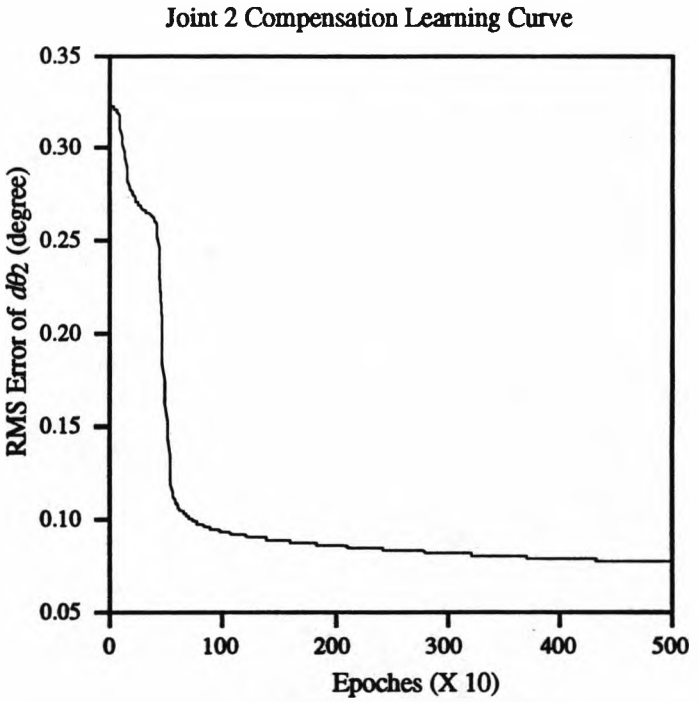


Figure 7.3. Neural Network Architecture for Accuracy Compensation

The network shown in Figure 7.3 has been trained separately for each component of output vector and put together for implementation after training. Therefore, the network training comprises of six Pi-sigma network training processes. Half of the collected data (144 points) are used as training exemplars for network training using the training algorithm as described in the previous chapter. The remaining data are used as a test data set. Figure 7.4 shows the first three of six joint Puma robot compensation learning

curves which exhibit fast and stable learning. The final RMS (root-mean-square) error of each joint compensation is below 0.1 degree which is the resolution of robot joint transducers.





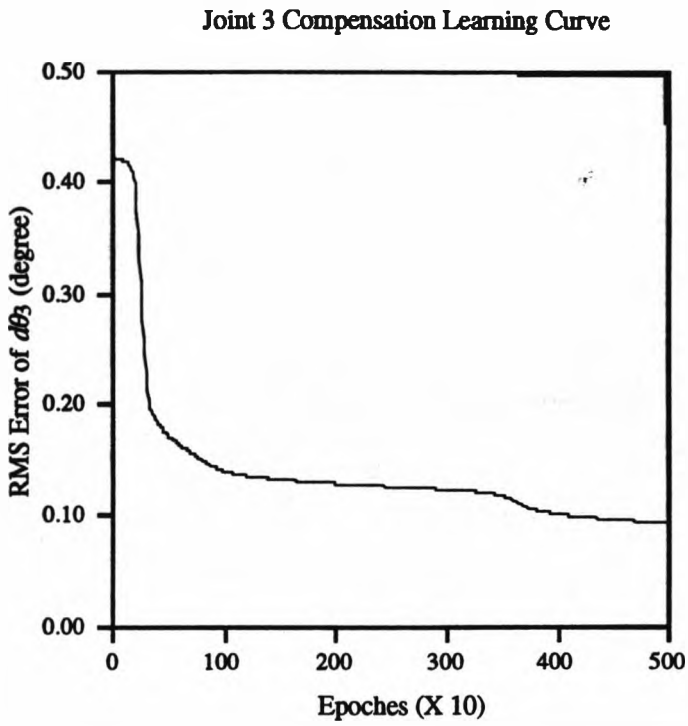


Figure 7.4. Learning Curves for Inverse Compensation

The trained network can generalise well in the calibrated volume. Three statistical measures (average error, standard deviation and maximum deviation), were used to evaluate robot accuracy compensation results. The results of using optimisation inverse compensation and neural network-based inverse compensation are listed in Table 7.1, which are based on 100 randomly-chosen test data points. The compensated positions and orientations are calculated using the compensated joint variables in the forward kinematic model and then compared with the actual data collected. From Table 7.1 we can see that the NN accuracy compensation achieves an average accuracy improvement factor of about 6. Comparing the NN-based forward accuracy modelling in the previous Chapter (Table 6.1), the NN-based inverse compensation has larger residual errors. This is due to the inverse mapping being a more complex relationship than the forward mapping. The optimisation compensation uses six controllable joint variables to compensate for Cartesian error at each configuration, therefore it can achieve a compensated accuracy error close to zero if the robot has sufficient DoF to move in each direction (non-singular configurations). The inverse compensation network stores and interpolates the joint corrections from off-line optimisation procedure and then can be used for on-line implementation of inverse compensation. Figure 7.5 shows the

position and orientation accuracy improvement after using the NN inverse compensation (sorted in ascending order for 100 randomly chosen test points), compared with the results of the optimisation approach. Not surprisingly, the NN inverse compensation is less accurate than optimisation compensation due to the residual errors of the NN learning.

Table 7.1 Inverse Accuracy Compensation Results of Puma Robot

	Before Compensation		Optim. Compensation		NN Compensation	
	Position	Orientat.	Position	Orientat.	Position	Orientat.
average	4.3707	2.5767	0.0021	0.1191	0.6474	0.4974
std dev.	0.8768	0.4075	0.0022	0.1033	0.3127	0.2828
maximum	5.3814	3.2331	0.0119	0.4327	1.7311	1.1477

(length in mm and angles in degrees)

Table 7.2 Experimental Evaluation of Inverse Compensation Results

	Before Compensation	Optim. Compensation	NN Compensation
	Position Error (mm)	Position Error (mm)	Position Error (mm)
average	4.1994	1.2504	1.5776
std dev.	1.0527	0.1416	0.2972
maximum	5.1482	1.4631	1.9678

Table 7.2 lists the experimental evaluation results of joint compensations based-on 12 test points across the calibrated area. The positioning errors before compensation are obtained by measuring robot end-effector positions achieved by controlling joint angles

recommended by the robot controller. The positioning errors after compensation are obtained by measuring end-effector positions achieved by controlling joint angles updated by compensation algorithms (only position data are collected for simplicity). The average position error (in length) decreased from 4.20 (mm) before compensation, to 1.25 (mm) after optimisation compensation and to 1.57 (mm) after the NN compensation. The accuracy improvements indicated by experimental results are less significant compared with the improvement as shown in Table 7.1. This can be partially explained by the fact that the measurements for calibration and the measurements for evaluation were made at a different time and therefore system error may have occurred in the measuring set-up[8]. Note that the standard deviation has been improved from 1.05 before compensation, to 0.14 after optimisation compensation and to 0.29 after the NN compensation, implying that the error changes after compensation are small and the relatively large average errors are due to constant system errors which existed in the measuring set-up. The final residual errors for evaluation points are expected to be less if joint compensations are obtained using on-line pose measurements during evaluation. Robot repeatability, which is limited by the robot controller, also attributed to the final residual error. However, experimental results show that the NN approach can achieve the same level of accuracy improvement as that achieved by numerical optimisation approaches which are computationally more expensive.

[8] The robot has been moved for other commitment therefore systematic error may occur in the base in which the robot has been installed.

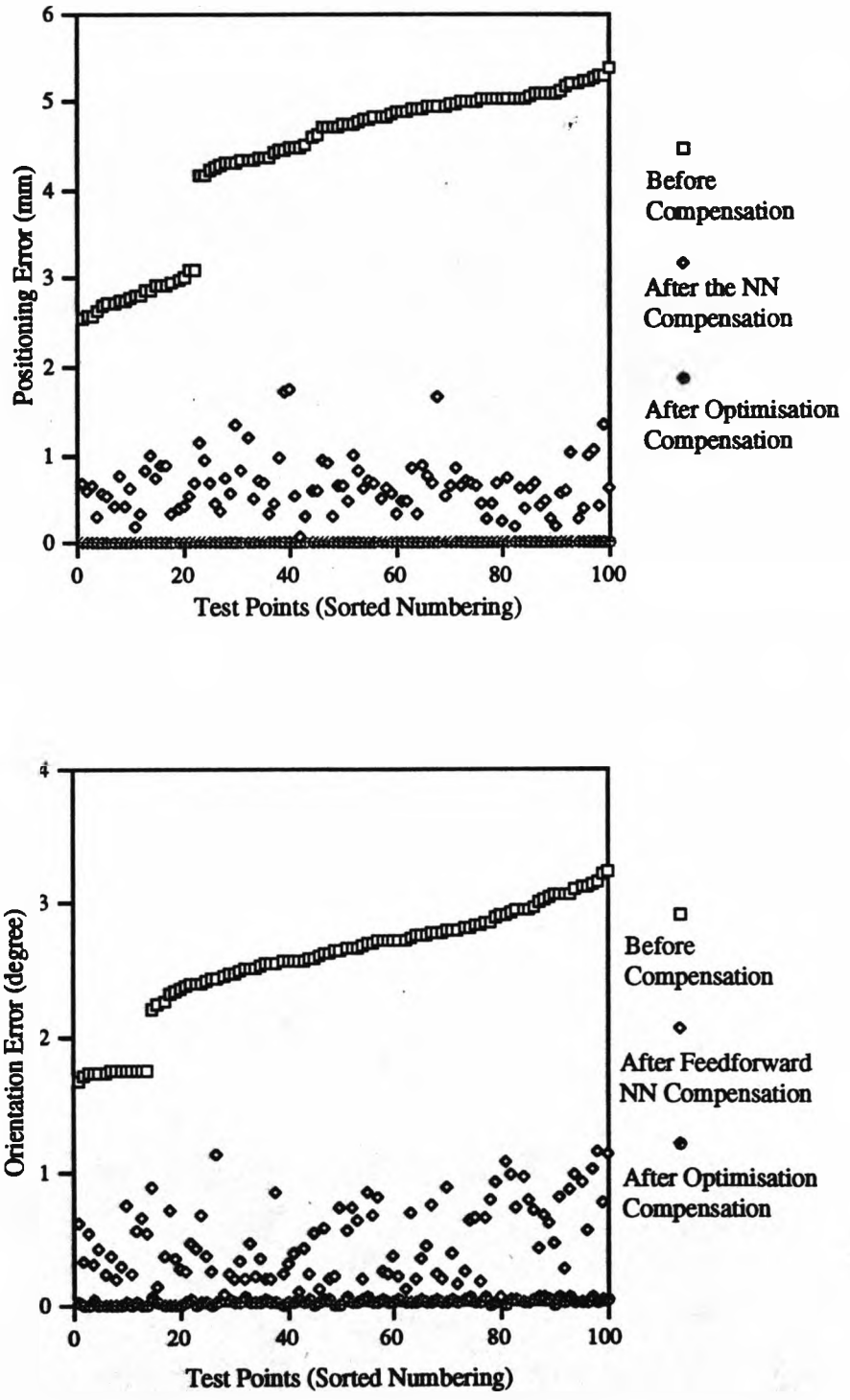


Figure 7.5 Accuracy Improvement of Inverse Compensation

The implementation of a Pi-sigma net is economical and efficient. The total adjustable weights used in the inverse compensation net are 6 * (6 * (6+1)) = 252. For parallel computation, only 1 trigonometric function call, 48 multiplication, and 42 additions are needed, which is equivalent to about 100 floating-point additions according to the conversions in Stone (1992). Even simulating on serial computers, the NN inverse compensation only requires about 6 * 100 = 600 floating-point addition-equivalent computations.

As a rule of thumb, the larger number of data points to be approximated, the larger the number of adjustable connection weights required in the network. A large number of network connection weights means a large network size. It is well-known that training of large feedforward networks is exceedingly slow and the residual training error is unacceptably high. Therefore, the feedforward neural network-based calibration is only suitable for local calibration compensation which has a relatively small number of training data points covering a small portion of robot workspace. Where robotic applications involve a large number of work points across the workspace (such as path or trajectory control), a Hopfield continuous-valued neural network architecture is appropriate to resolve the inverse compensation problem (Zhong and Lewis, 1994; Zhong, Lewis and N-Nagy, 1995).

7.3. Model-based Accuracy Compensation

7.3.1. Problem Formulation and Numerical Solutions

Model-based accuracy compensation is a subset of the robot inverse kinematics problem which involves a process to find the solution of a group of coupled non-linear functions, given the initial conditions determined by the nominal inverse kinematic model. The accuracy compensation problem can be stated as follows:

Given:

1) The robot nominal kinematic model relating the end-effector homogenous transformation matrix **T** to the vector of joint configuration **q**:

$$\mathbf{T} = \mathbf{F}_n(\mathbf{q}) \tag{7.4}$$

2) The robot actual pose homogenous transformation $\mathbf{T}_a$ predicted by the calibrated model or directly measured by measuring device

$$\mathbf{T}_a = \mathbf{F}_c(\mathbf{q}) \tag{7.5}$$

3) Desired pose transformation $\mathbf{T}_d$ and a corresponding nominal inverse kinematic solution $\mathbf{q}_n$ at this pose

$$\mathbf{q}_n = \mathbf{F}_n^{-1}(\mathbf{T}_d) \tag{7.6}$$

Find:

The necessary joint change $d\mathbf{q}$ of the joint values such that

$$\mathbf{F}_c(\mathbf{q}_n + d\mathbf{q}) = \mathbf{T}_d \tag{7.7}$$

Note in the above problem formulation, the actual pose transformation $\mathbf{T}_a$ can be determined by the calibrated model or by direct end-effector pose measurement. The calibrated model is not necessarily restricted to the kinematic model, the non-geometric model can also be added to the model to predict the actual end-effector pose. If an on-line measurement device is used to determine the end-effector pose, then no calibrated model need be involved and the problem can be regarded as the correction phase of the robot re-programming problem (Zhuang 1989; Mooring, Roth and Driels, 1991). Therefore, model-based accuracy compensation methods developed below are not necessarily limited to kinematic calibration compensation. An outline of the commonly used Newton-Raphson (N-R) algorithm is given below.

Newton-Raphson (N-R) Algorithm:

Step 1: Compute an estimated robot pose $\mathbf{T}_a$ that corresponds to the available nominal inverse kinematics solution $\mathbf{q}_n$ (In the case that on-line measurement is used, $\mathbf{T}_a$ is obtained directly from sensor measurement of robot joint configuration $\mathbf{q}_n$).

$$\mathbf{T}_a = \mathbf{F}_c(\mathbf{q}_n) \tag{7.8}$$

Step 2: Calculate the pose error matrix between the desired pose $\mathbf{T}_d$ and the estimated actual pose $\mathbf{T}_a$.

$$d\mathbf{T} = \mathbf{T}_a - \mathbf{T}_d \tag{7.9}$$

Step 3: From $d\mathbf{T}$ form the equivalent differential error vector $d\mathbf{x}$

$$d\mathbf{x} = \begin{bmatrix} d\mathbf{p} \\ d\mathbf{w} \end{bmatrix} = \begin{bmatrix} (dx, dy, dz)^T \\ (\delta x, \delta y, \delta z)^T \end{bmatrix} \tag{7.10}$$

This is done through the steps as described in Chapter 4 (Equations 4.27-4.35).

Step 4: Compute joint changes $\delta\mathbf{q}$ using

$$\delta\mathbf{q} = \mathbf{J}^{-1} d\mathbf{x} \tag{7.11}$$

where $\mathbf{J}$ is the robot Jacobian formulated using the nominal model. The Jacobian matrix $\mathbf{J}$ is an ordinary Jacobian which is obtained by linearizing the robot inaccuracy model with respect to joint variables only ($\mathbf{J}_\theta$), compared with the special Jacobian matrix which is linearized with respect to all kinematic parameters. Here we denote the $\mathbf{J}_\theta$ as $\mathbf{J}$ for convenience without confusing with the special Jacobian matrix as used in kinematic identification phase.

Step 5: Update joint commands by setting

$$\mathbf{q}_n = \mathbf{q}_n + \delta\mathbf{q} \tag{7.12}$$

Steps 1-5 are repeated until an appropriate termination condition is satisfied. One of the termination conditions can be that the joint changes $\delta\mathbf{q}$ become smaller than the joint encoder resolution. Or if the error vector $d\mathbf{x}$ reaches to the pre-specified threshold, then stop the algorithm.

The computation efficiency of the above algorithm depends critically on Equation 7.6 and *Step* 4. For robots with general geometry, finding the compensated joint commands with the above algorithm may not be any more effective than directly solving the inverse kinematics of calibrated robot using numerical methods. For industrial robots with simple geometry, closed form analytic inverse solutions are available and the analytic solutions are very time efficient. The initial solutions provided by the nominal inverse kinematics can speed up the convergence of the N-R algorithm. Stone (1992) showed that if the initial nominal solutions are close to the actual solutions (the initial end-effector errors are typically within 5 millimetres), then

the N-R algorithm can converge in two iterations, compared with four iterations required to solve directly the inverse kinematics of the calibrated robot.

However, the N-R method breaks down when the desired pose (task points) fall at or near a robot singular configuration. When the robot is at one of its singular configurations, the inversion of the Jacobian (Step 4) does not exist. If the task points are near singular configuration, the Jacobian matrix will be badly-conditioned and the joint compensations determined by Equation (7.11) will be relatively large. The physical interpretation of this is that large joint adjustments are needed to compensate small errors in workspace. Large joint compensation is not desirable for robot accuracy compensation since the joint limits might be exceeded and the large joint adjustment movements of the robot may cause collision with the objects in the robot workspace. To overcome the singularity problem, robust compensation algorithms such as the Singular Value Decomposition (SVD), Levenberg-Marquardt and Linear Quadratic Regulator algorithms were proposed (Zhuang 1989; Zhuang, Hamano and Roth 1989). Existence and uniqueness of the compensation solution are ensured due to the particular structure of the performance index. However, the computation of such robust algorithms is typically rather complex which makes on-line implementation of inverse compensation problematic. In the next section, the inverse compensation problem has been re-formulated such that the recurrent Hopfield continuous-valued neural network is applied. Given the initial conditions of the network, which are determined by the robot nominal kinematic parameters and joint solutions, the network obtains global optimal solutions in a few characteristic time constants of the neural circuit, even in the robot configurations near singularity where the N-R algorithm breaks down.

7.3.2 The RNN-based Algorithm for Accuracy Compensation

Recall that the quadratic form of the linear residual error model was used to construct the network energy for the RNN-based identification algorithm during the identification phase (Equation 4.10-4.11). The same procedure applies for the accuracy compensation problem with a difference in the construction of the linear residual error model and the associated Jacobian. Instead of linearizing the robot inaccuracy model with respect to all kinematic parameters as in the identification phase, the robot inaccuracy model is only linearized with respect to the controllable joint variables in the case of accuracy compensation. Let $\Delta \mathbf{x} = \mathbf{f}(\mathbf{k}, \mathbf{q}) - \mathbf{f}(\mathbf{k}^0, \mathbf{q}^0)$ be the inaccuracy vector

of the robot end-effector location predicated by the actual control model $\mathbf{f}(\mathbf{k}, \mathbf{q})$ and the nominal kinematic model $\mathbf{f}(\mathbf{k}^0, \mathbf{q}^0)$, then the linearized residual errors obtained by perturbing the controllable joint variables $\delta\mathbf{q}$ around the nominal values is:

$$\mathbf{e}(\delta\mathbf{q}) = \Delta\mathbf{x} - \mathbf{J}\delta\mathbf{q} \tag{7.13}$$

where $\mathbf{J}$ is the ordinary Jacobian of robot manipulator evaluated at robot nominal kinematic parameters and joint values $(\mathbf{k}^0, \mathbf{q}^0)$; whilst the aggregated special Jacobian was used in kinematic identification.

The quadratic form of the linearized residual error is then formulated as a network energy function so that a decrease of network energy corresponds to a decrease of robot residual positioning inaccuracy. It is desirable to have only small joint compensation values, therefore a penalty term is added to the energy function to ensure small outputs. The energy function for joint compensation is:

$$E = \frac{1}{2}[\mathbf{e}(\delta\mathbf{q})]^T \mathbf{Q}[\mathbf{e}(\delta\mathbf{q})] + \frac{1}{2}\delta\mathbf{q}^T \Lambda \delta\mathbf{q} \tag{7.13}$$

where Λ is a positive diagonal weight matrix for regulation, and $\mathbf{Q}$ is a weight coefficient matrix defined as in Equation 4.3.

Using the linear residual error model (7.12) in (7.13), and expressing in the standard form of the network energy, we have:

$$E = -\tfrac{1}{2}\sum_{i,j=1}^{n} T_{ij}\delta q_i \delta q_j - \sum_{j=1}^{n} I_j \delta q_j + \tfrac{1}{2}\sum_{i=1}^{m} Q_i (\Delta x_i)^2 \tag{7.14}$$

where

$$T_{ij} = -\sum_{s=1}^{m} (Q_s J_{si} J_{sj} + \lambda_i \delta_{ij}), \tag{7.15}$$

and

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

Q_i is the i-th diagonal element of coefficient matrix $\mathbf{Q}$, λ_i is the i-th diagonal element of the coefficient matrix Λ, and

$$I_j = \sum_{i=1}^{m} J_{ij} \Delta x_i \tag{7.16}$$

T_{ij} and I_j determine the network connection weights and input currents respectively based on the nominal kinematic model and parameters (the ordinary Jacobian). The δq_j is the j-th joint compensation which corresponds to the j-th neuron state. n is the robot DoF while m is the dimension of robot Cartesian space.

Note that the structure of the above formulation is exactly the same as that in the formulation of the RNN-based identification algorithm except for the definitions of the network connection weights and input current (Equation 7.15 and 7.16). Following the same derivation procedures as in the identification algorithm, the neuron circuit dynamics equation of the accuracy compensation network is given as follows:

$$\frac{d(\delta q_i)}{dt} = \mu_i (\sum_{j=1}^{n} T_{ij} \delta q_j + I_i), \qquad i = 1, 2, ..., n \tag{7.17}$$

where μ_i is the i-th diagonal element of the positive diagonal coefficient matrix μ which is chosen to ensure the stability and the convergence speed of the circuit. Given the initial condition of the neuron states ($\delta q_i = 0$, $i = 1, 2, ..., n$), the above differential equation determines the neuron state trajectories, hence the joint compensation amounts (the stable states of the neurons).

Equation (7.17) is a group of coupled first-order ordinary differential equations (ODE). As the implementation of the RNN-based identification algorithm, the RNN-based accuracy compensation algorithm was implemented using the dynamic system simulation software SIMULINK™ (Mathworks, 1992b). There are several options of the ODE solvers provided by the SIMULINK™, the method which subtracts the linear dynamics of system was chosen due to the linear model involved. The ODE solver is called as *linsim* which has the following calling format:

[t, x, y] = linsim('*model*', [tstart, tfinal], $\mathbf{x}_0$, [tol, minstep, maxstep]);

where **[t, x, y]** are returned variables, **t** is a vector of the recorded time sequence of the system evolution, **x** and **y** are the system state variable and output vector respectively. [tstart, tfinal] specifies the simulation start and stop time; while [tol, minstep, maxstep] specifies tolerance, minimum and maximum step size of the

integration. $\mathbf{x}_0$ is the initial conditions of the dynamic system. The *model* is the name of subroutine which defines the system using the state space description:

$$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{7.18}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{7.19}$$

where **x**, **u** and **y** are state, input, and output vectors, respectively. **A**, **B**, **C**, **D** are coefficient matrices which can be specified as follows according to above problem formulation (Equations 7.15-7.17).

$$\mathbf{A} = -\mu*(\mathbf{J}^T*\mathbf{J} + \Lambda) \tag{7.20}$$

$$\mathbf{B} = \mu*\mathbf{J}^T \tag{7.21}$$

Since the desired system output is the final state of the state variable **x**, **C** and **D** are set to be an identity matrix and zero respectively. The input vector **u** is specified by the robot inaccuracy vector $\Delta\mathbf{x}$.

To exemplify the efficiency and robustness of the Hopfield neural net compensation scheme for robot global inverse compensation, we address path compensation and compensation near a robot singularity, comparing with the numerical compensation algorithm (the N-R).

7.3.3 Path Compensation

There are many applications (such as welding) which require a robot to execute a continuous path or trajectory accurately. Robot end-effector inaccuracy at a number of points along the trajectory are calculated by the calibrated model or measured by an on-line measurement sensor. In such cases an accuracy compensation algorithm is required to find joint correction in real-time to compensate the inaccuracy in Cartesian space. Due to the difficulty of actual measurements of robot end-effector across the large volume of workspace using a co-ordinate measuring machine, the end-effector location errors are computed using the calibrated kinematic model based on the identified kinematic parameters (Table 4.2). The end-effector of the PUMA robot was programmed to execute a spiral trajectory specified by the Homogenous transformation in Cartesian space:

$$\mathbf{T} = \left[\begin{array}{ccc|c} \mathbf{n} & \mathbf{s} & \mathbf{o} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{7.22}$$

where

$$\mathbf{n} = (-\sin\varphi,\ \cos\varphi,\ 0)^T, \tag{7.23}$$

$$\mathbf{o} = (\cos(\pi/4)\cos(\varphi), \cos(\pi/4)\sin(\varphi), \sin(\pi/4))^T \tag{7.24}$$

$$\mathbf{s} = \mathbf{o} \times \mathbf{n} \tag{7.25}$$

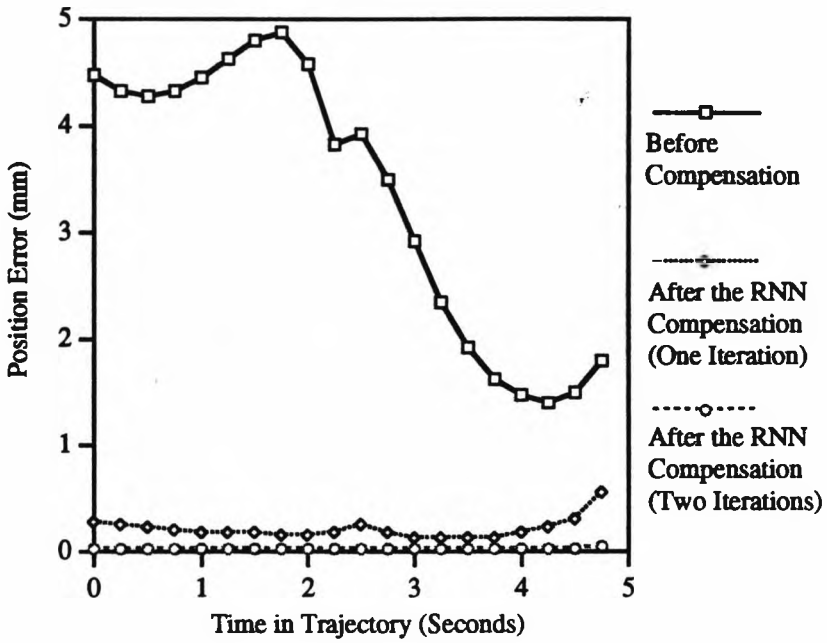
$$\mathbf{p} = (-45 + 20\sin(\varphi), 4.5 + 20\cos(\varphi), -26 + 5\pi t)^T \text{ (cm)} \tag{7.26}$$

and

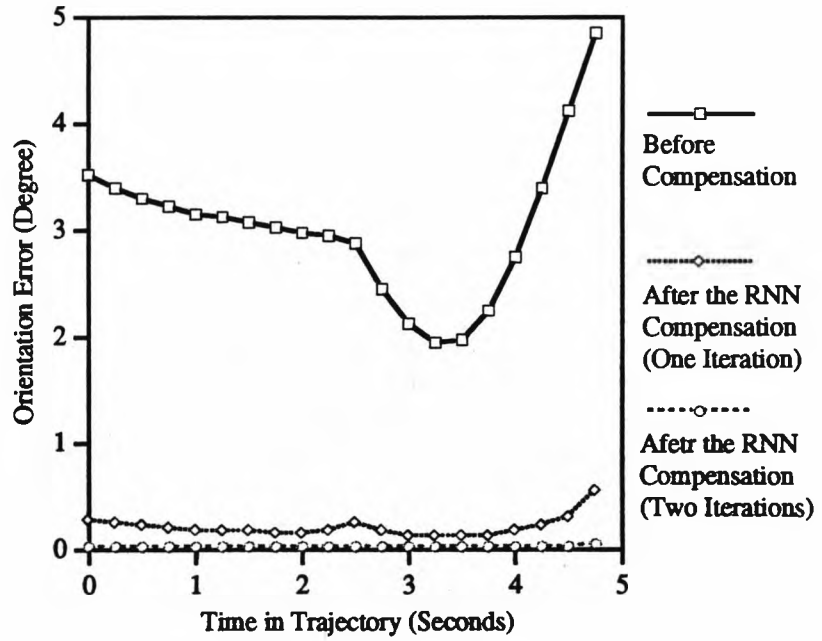
$$\varphi(t) = (\pi/4)t\ ,\ 0 \le t < 5 \tag{7.27}$$

The task points were specified at discrete points along the path with a cycle time of 25 *ms*.[9] The cycle time is demanding to most of the numerical algorithms for on-line compensation. However, this time is sufficient for the neural circuit to converge. The setting time of neural circuit is in the order of μs (simulation time is about 23 *ms* based on the HP workstation), given the coefficients of **Q** as an identity matrix, Λ equal to. zero in Equation 7.13, $\mu_i = 10^6$, and the initial conditions of neuron states $\delta q_i = 0, i = 1,2,...n$ in the neuron dynamics Equation 7.17. Figure 7.6 is the position and orientation errors (in length) caused by kinematic errors before and after one and two iterations of neural network compensation (one of the eight robot joint configurations-RIGHT and ABOVE arm, and UP wrist-is selected), which shows a significant accuracy improvement along the path. After two iterations of the RNN-based compensation, both position and orientation residual errors are decreased to near zero. The regulation coefficient Λ is set to be zero because the Jacobian is well-conditioned in those task configurations.

[9] The servo cycle time of Puma robot is 28 *ms*.



(a). Position Error Along the Path



(b). Orientation Error Along the Path

Figure 7.6. Accuracy Compensation Along the Path

7.3.4 Compensation Near Robot Singularity

Next we examine the compensation near a robot singularity. Let the robot joint configuration be $\theta = [\pi/2, -\pi/2, \pi/4, \pi/4, \theta_5, 0]$ and allow only joint 5 to rotate: $-\pi/4 \leq \theta_5 \leq \pi/4$. The robot will be in its wrist singular configuration when joint 5 is in the neighbourhood of zero. The PUMA wrist singular configurations are commonly used for some typical assembly operations. Figure 7.7 illustrates the PUMA robot wrist singular configuration (when the joint 5 equals to zero, the joint axes of joint 4 and 6 are co-linear). Obviously this configuration is a convenient configuration for many robotic tasks such as assembly, pick-and-place operations. Therefore robot singularity problems for certain type robots can not simply stay away by avoiding the use of singular configurations during the motion planning process.

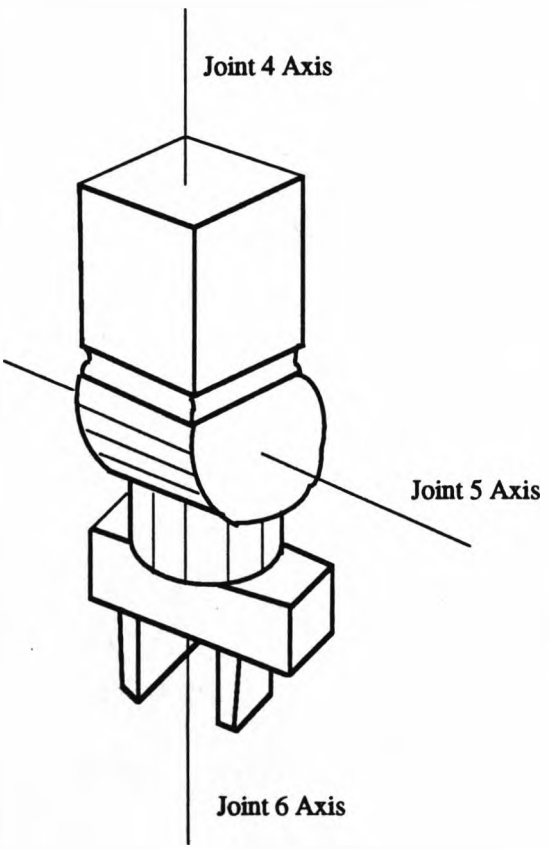


Figure 7.7 Robot End-effector and Wrist Singularity

Figure 7.8 and 7.9 compare the effects of position and orientation compensation respectively near robot singularity after one iteration compensation using the RNN and N-R method. Figure 7.8 shows that the RNN compensation is stable near robot singularity while the widely used numerical algorithm (Newton-Raphson) failed to converge. Figure 7.9 shows that both N-R and RNN method can converge but N-R method has smaller residual orientation error than the RNN method after one iteration compensation. It is not disadvantageous for RNN method since more often than not, robot absolute position accuracy is more critical than orientation accuracy. In the case that more accurate orientation is required, the coefficient matrix **Q** in the network energy function Equation 7.13 can be used to adjust the balance weight between residual position error and orientation error.

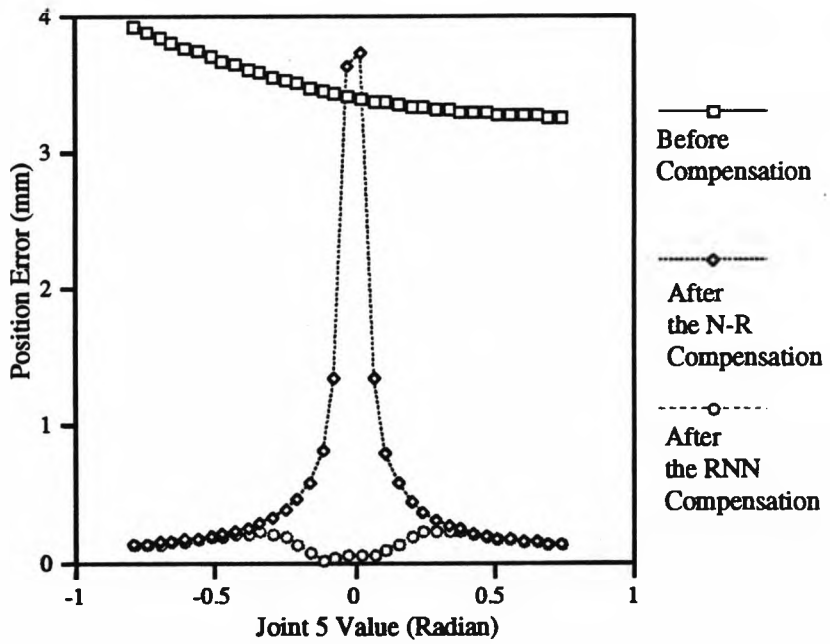


Figure 7.8 Position Compensation Near Robot Singularity

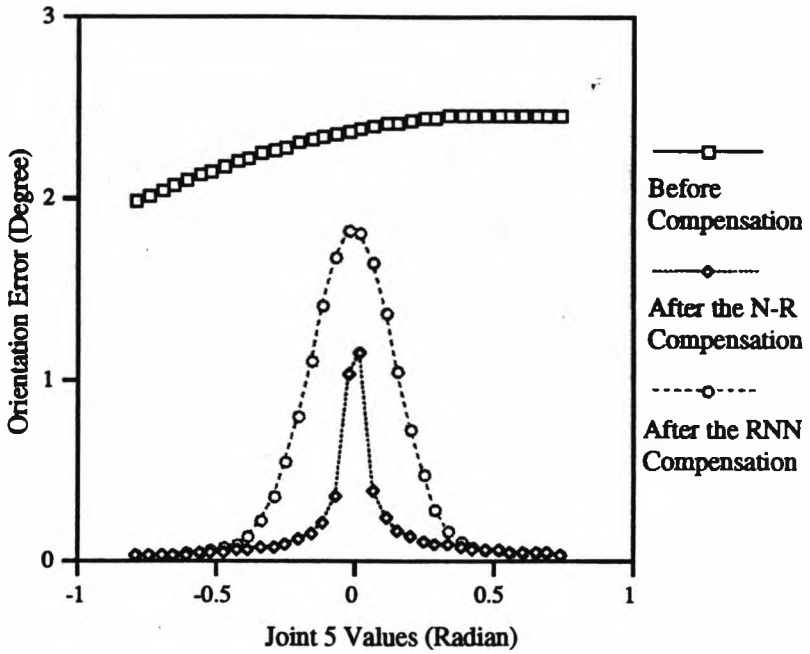


Figure 7.9. Orientation Compensation Near Robot Singularity

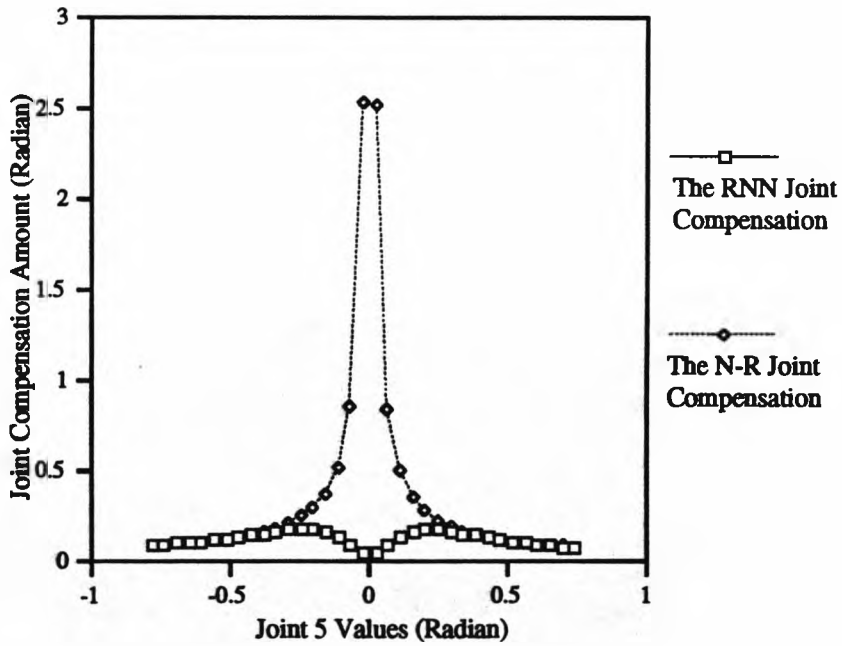


Figure 7.10 Joint Compensation Amount Near Singularity

The joint compensation amount obtained by the RNN and numerical approach are also displayed in Figure 7.10 (length in radian). The joint compensations computed using the Newton-Raphson (N-R) method are too large to be feasible near singularity while the RNN joint compensations using the RNN method are reasonably small for practical implementation. The large joint compensation amount needed to compensate small errors in workspace using the N-R method is highly undesirable since robot joint limits might be exceeded, and the robot might collide with fixtures or other objects in the workspace during the joint adjustment movements. From Figures 7.8-7.10, we can see that the RNN-based algorithm performs similarly with the N-R algorithm for non-singular task points.

In the above simulations, all the network design parameters are set as the same as used in the path compensation for all the task points. The effects of some parameters on the compensation at singular configurations need to be investigated. One singular configuration is chosen by setting joint 5 to -1 degree, i.e. [90, -90, 45, 45, -1, 0].[10] Simulation experiments are performed at this configuration to show the effects of the regulation coefficient Λ and the learning rate μ. The reason for choosing the coefficient Λ and μ as matrices is to consider the case that an individual coefficient value is chosen for each joint variable. If no such discrimination among joint variables is made, the coefficients can be set as scalar constants $\Lambda = \lambda$; $\mu = \eta$ for convenience. Table 7.3 lists simulation results in the singular configuration using the learning rate $\eta = 10^1$ for various cases: * $\lambda = 0.1$; ** $\lambda = 0.01$; *** $\lambda = 0.0$. It shows that the regulation parameter λ has no effect on both the accuracy improvement (upper part of the Table) and the joint correction required (lower part of the Table) in this case. The joint correction required are small for various regulation parameters, and residual errors of both position and orientation are smaller after correction, but the convergence speed is slow due to the small learning rate used.

By increasing the learning rate to 10^6, the same simulations have been performed and the results are listed in Table 7.4. The accuracy improvement in this case is more significant compared with the Table 7.3 due to larger learning rate used. The regulation parameter λ has only a minor effect on accuracy and joint correction required. The joint corrections required are reasonably small even setting λ to zero.

[10] This configuration is selected because it was studied in (Zhuang 1989, Morning, Roth and Driels, 1992). The condition number of the compensation Jacobian at this configuration is up to 1.058×10^4.

Table 7.3 Simulation Results for $\eta = 10^1$ in a Singular Configuration

Residual Error in length	Position Error (mm)	Orientation (degree)
Before Correction	3.4065	2.3643
After Correction*	1.8176	2.3052
After Correction**	1.8176	2.3052
After Correction***	1.8176	2.3052

*$\lambda = 0.1$
**$\lambda = 0.01$
***$\lambda = 0.0$

Joint Correction	$\Delta\theta_1$	$\Delta\theta_2$	$\Delta\theta_3$	$\Delta\theta_4$	$\Delta\theta_5$	$\Delta\theta_6$
For Correction*	0.2428	0.0241	-0.1032	0.0017	-0.0704	0.0010
For Correction**	0.2428	0.0241	-0.1032	0.0017	-0.0704	0.0010
For Correction***	0.2428	0.0241	-0.1032	0.0017	-0.0704	0.0010

(Joint Correction is in Degrees)

Table 7.4 Simulation Results for $\eta = 10^6$ in a Singular Configuration

Residual Error in length	Position Error (mm)	Orientation (degree)
Before Correction	3.4065	2.3643
After Correction*	0.0424	1.8272
After Correction**	0.0470	1.8188
After Correction***	0.0417	1.8155

*$\lambda = 0.1$
**$\lambda = 0.01$
***$\lambda = 0.0$

Joint Correction	$\Delta\theta_1$	$\Delta\theta_2$	$\Delta\theta_3$	$\Delta\theta_4$	$\Delta\theta_5$	$\Delta\theta_6$
For Correction*	0.4468	0.1518	-0.6458	0.4377	0.2013	0.6941
For Correction**	0.4585	0.3558	-0.6559	-0.0600	0.2519	1.2342
For Correction***	0.4606	0.3566	-0.6570	-0.3177	0.2584	1.4903

(Joint Correction is in Degrees)

Table 7.5 shows that the regulation parameter λ plays an important role in the case that learning rate η is increased further to 10^8. The joint correction required is very large at a robot singular configuration if the regulation parameter is set to zero in this case, although both position and orientation accuracy have been improved after correction. Due to the axes of joint 4 and 6 being co-linear in the singular configuration, the joint 4 and joint 6 rotate a large angle in opposite direction which only result in a small movement at the end-effector. Choosing a small value of regulation parameter $\lambda = 0.01$ can suppress the large joint correction required. Increasing λ from 0.01 to 0.1 has relatively little effect on both accuracy and joint correction.

Table 7.5 Simulation Results for $\eta = 10^8$ in a Singular Configuration

Residual Error in length	Position Error (mm)	Orientation (degree)
Before Correction	3.4065	2.3643
After Correction*	0.0424	1.8272
After Correction**	0.0409	1.8108
After Correction***	1.2122	1.0522

*$\lambda = 0.1$
**$\lambda = 0.01$
***$\lambda = 0.0$

Joint Correction	$\Delta\theta_1$	$\Delta\theta_2$	$\Delta\theta_3$	$\Delta\theta_4$	$\Delta\theta_5$	$\Delta\theta_6$
For Correction*	0.4468	0.3518	-0.6458	0.4376	0.2013	0.6942
For Correction**	0.4606	0.3567	-0.6552	-0.6854	0.2519	1.8581
For Correction***	0.6819	0.4413	-0.5864	-64.8128	0.2566	65.8153

(Joint Correction is in Degrees)

From the above simulations we can see that the RNN-based compensation algorithm is not sensitive to regulation parameter and robot singular configuration until the learning rate is very high. When the learning rate is relatively small (the learning rate is normally set below 10^6), the RNN-based algorithm can find desirable joint correction and achieve good accuracy improvement in singular configurations even without using regulation parameter ($\lambda = 0$). This capability of the RNN-based algorithm eliminates the need of special treatments for singular configurations since no regulation is needed when the robot is not in singular configurations. This is a good property of the RNN-based algorithm because the algorithm can handle the singularity problem automatically without explicitly identifying it.

Recent work by Everett, Colson and Mooring (1994) highlighted the importance of automatic singularity avoidance during the joint compensation process. Compared with the approach used by Everett, Colson and Mooring (1994), where the joint compensation problem near singularity was formulated as a constraint optimisation problem in which the joint movement constraints should be decided at a specific task point, the RNN-based approach does not require any task specific constraints and can obtain quality solutions automatically. The RNN-based algorithm is also computationally more efficient than the non-linear constraint optimisation algorithm.

7.4 Chapter Summary

The robot accuracy compensation problem has been treated in this chapter under the framework of artificial neural networks. Both non-parametric and model-based parametric compensation have been studied. For non-parametric compensation, a simple feedforward neural network architecture has been applied successfully to approximate the complex non-linear mapping between robot configurations and robot inverse compensations. Using a NN learning method, the network can generate high-order polynomial approximation efficiently and economically for robots with multiple DoF. A constant-time compensation can be achieved by using the neural network representation of the inverse compensation knowledge from the off-line non-linear optimisation procedure. Results for a six DoF PUMA robot have been presented. However, the feedforward neural network compensation is a local calibration which is effective under the assumption that robot accuracy is critical only for a small volume of workspace. For global compensation, which involves a large number of work points across the whole workspace, a Hopfield continuous-valued recurrent neural network has been applied to achieve efficient and robust inverse compensations. The RNN-based compensation requires no training, only the end-effector inaccuracy and robot nominal joint values should be provided to determine the input and connection weights of the network (the internal representation of the network is based on the model knowledge). Simulation examples of path compensation and compensation near a robot singularity have been presented.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

The concept of autonomous robot calibration has been extended in this thesis to include a fully automated process for a robot to perform self-calibration (using its internal sensor measurements) *on site* whenever and wherever necessary (after a certain period of operation and in the area where high accuracy is required). Autonomous calibration requires an efficient and robust data processing algorithm, and a fully automated measurement method. Artificial neural network techniques have been vigorously investigated for calibration data processing (modelling, identification and compensation), and a novel measurement method has been developed for autonomous robot calibration in this thesis. The contributions of the thesis are summarised as follows:

1) A new kinematic identification algorithm based on the Hopfield type recurrent neural network (RNN) has been developed. The configurations of the network (network connection weights and inputs) are determined by nominal kinematic model and measured robot pose errors. Robot kinematic parameter errors are identified in a few characteristic time constants of the neural circuit (which is determined by the chosen learning parameter) even using a singular kinematic model (standard D-H model). If robot inaccuracy data can be collected on-line, we have shown that robot kinematic identification can be performed in real time by using the RNN-based real time optimisation technique. The RNN-based algorithm also exhibited numerical robustness over conventional least square methods due to the use of ordinary differential equations (ODE) in the simulation. For parallel implementation, the computation time of the RNN-based algorithm was independent of the number of robot DoF and the number of

parameters to be identified. Therefore, the RNN-based identification algorithm is especially attractive for robots with multiple DoF (redundant robots) which are required to maintain calibration in real-time.

2) A generic accuracy function which accounts for various error sources was introduced. Feedforward neural networks were used to implement the generic accuracy model. The generic accuracy function served as the basis for the design of the network architecture. Pi-sigma networks, which are capable of representing higher-order non-linear functions using a simple network architecture, provide a natural computational mechanism for implementation of the generic accuracy function. Instead of identifying error sources explicitly, the error source information was encoded in the distributed network connection weights. Because of the complex nature of the accuracy problem for multiple DoF robots, the NN representation is appealing due to its learning methodology, robustness and efficiency.

3) A new autonomous robot calibration tool was developed using a trigger probe and a reference constraint plane. The probe was manufactured as a standard tool for the robot, enabling the robot to grip it automatically and use it to touch constraint surfaces for consistency checks and data collection when calibration is necessary. Instead of taking partial or complete pose measurements for robot calibration, the tip-point of the probe was constrained to movement within a plane and only robot internal joint measurements were used for kinematic identification. Neither external measurements nor accurate fixture set-up are needed for such a calibration. Both simulation and experimental results for a Puma robot show that robot positioning accuracy can be improved to the level of robot repeatability.

4) Robot accuracy compensation problems were extensively treated under the framework of artificial neural networks. Both non-parametric and model-based parametric compensation were studied. For non-parametric compensation, a simple feedforward neural network architecture has been applied successfully to approximate the complex non-linear mapping between robot configurations and robot inverse compensations. Using NN learning methods, the network generates higher-order polynomial approximation efficiently and economically for robot with multiple DoF. A constant-time compensation was achieved by using the neural network representation of the inverse compensation knowledge (obtained from the off-line non-linear optimisation procedure). The feedforward neural network compensation is a local calibration which is effective under the assumption that robot accuracy is critical only

for a small volume of workspace. For global compensation which involves a large number of work points across the whole workspace, a Hopfield continuous-valued recurrent neural network (RNN) was applied to achieve efficient and robust inverse compensations. The RNN-based compensation requires no training, only the end-effector inaccuracy and robot nominal joint values were provided to determine the input and connection weights of the network (the internal representation of the network is based on the model knowledge). The RNN-based accuracy compensation algorithm is suitable for on-line compensation, and is able to obtain good solutions even in the robot singular configurations.

5) Robot calibration experiments were performed using a PUMA 560 robot. The theories and techniques of modelling, measurement, identification, and compensation developed in this thesis were all verified through experimentation. In the local calibration workspace, it has been shown that positioning errors of the PUMA robot were reduced from 4-5 (mm) to about 0.2 (mm). The model-based kinematic calibration achieved the same level of positional accuracy as the generic model-based (non-parametric) calibration, implying that kinematic models can be used to compensate both geometric and non-geometric errors in local workspace volume. The new autonomous calibration scheme presented in the thesis which uses a simple trigger probe has improved robot positioning accuracy to the level of robot repeatability. These results are consistent with the results obtained using precision external measuring devices such as CMM (Coordinate Measuring Machine).

The following topics are suggested for further research:

1) This thesis primarily concentrates on calibration of robot static errors (e.g. kinematic errors, static deformations, etc.). However, the dynamic characteristics of the robot (e.g. actuator/link mass and inertia, friction in actuators and joints, stiffness, etc.) are also very important in affecting robot positioning accuracy for high speed robots. Therefore, dynamic calibration, which determines dynamics related parameters of robot manipulator, is a natural extension of the static calibration techniques described in this thesis. A dynamic measurement system, which can collect the information dynamically regarding the end-effector's position, speed, and acceleration, is crucial for dynamic calibration. The ANN-based calibration algorithms developed in the thesis are particularly well suited to the problem of dynamic calibration processing.

2) Accuracy evaluation and test experiments were limited to the specific experimental set-up in this research due to the expensive Coordinate Measuring Machine being used. With the availability of some economical and automatic on-line measurement devices, more extensive experiments are needed to test and evaluate robot accuracy performance according to ISO 9000 standard.

3) The optimal placement of the constraint plane in the constrained environments must be investigated so that the robot has optimal identification configurations. Such a study could lead to the construction of a portable mechanical fixture for robot on-site calibration. Constraint surfaces other than planes may also be suitable for the proposed calibration method (such as a spherical ball) as long as the surfaces have known shapes and are suitable for touching with the probe. The effect of measurement noises such as the flatness of constraint planes and the resolution of the probe on identification accuracy need to be studied by simulation in future work.

4) Robot inaccuracy in Cartesian space is minimised in the least square sense by configuring the network energy function as a quadratic form of the inaccuracy vector (L_2-norm). This standard least square criterion is optimal for a Gaussian distribution of measurement noise. If the set of measurements has non-Gaussian error distribution due to different sources of errors such as instrument errors, modelling errors, sampling errors and human errors, other criteria such as the least absolute value (L_1-norm), and maximum likelihood criterion (L_∞-norm), etc. can be used to construct the network energy function. Cichocki and Unbehauen (1992, 1993) discussed the design of an efficient and robust neural network architecture based on various criteria. Future work will investigate more advanced neural network architecture for the robot kinematic identification problem taking various measurement errors (noise) into account.

5) The concept of using a known shape reference object for robot calibration developed in the thesis can also be extended for non-contact type sensors such as CCD camera.

6) The NN training accuracy and efficiency will suffer if larger data sets covering larger workspace for multiple DoF robot are used. Neural network design methodology, which incorporate *a priori* knowledge into network architecture so that the network can capture invariant properties of the problem from high-dimensional data, remains an open research topic.

7) The Hopfield analogue (continuous-valued) model is one of the most popular neural network model and has found many applications. It can easily be implemented using VLSI electronic circuits (Cichocki and Unbehauen, 1993). There are two small variations in the network used for inverse compensation. One is that the weight connections are determined by robot Jacobian matrix which is time-variant. This is different from the normal Hopfield model which has fixed weights. This can be implemented by electronic circuits with programmable resistors. Another variation is that the neuron activation function is simply a linear function of high gain, which simplifies the solution of the differential equations of the neuron dynamics. With the availability of neural circuits, the hardware implementation of the neural network-based identification and compensation algorithm is suggested for future work which can be used to augment the nominal robot controller.

References

Albus, J.S., 1975a, "Data storage in the cerebellar model articulation controller (CMAC)", *Trans. of the ASME Journal of Dynamic Systems, Measurement, and Control,* vol. 97: pp. 228-233

Albus, J.S., 1975b, "A New Approach to Manipulator Control: The Cerebellar model Articulation Controller", *Trans. of the ASME Journal of Dynamic Systems, Measurement, and Control,* vol. 97: pp. 270-277

Ahmad, Z. and Guez, A., 1990, "On the solution of the inverse kinematic problem," *Proc. of IEEE Int. Conf. on Robotics and Automation,* IEEE, pp. 1692-1697

Bassi, D.F. and Bekey, G.A., 1989, "High Precision Control by Cartesian Trajectory Feedback and Connectionist Inverse Dynamics Feedforward", *Proc. International Joint Conf. on Neural Networks,* vol. 2, pp. 325-332

Bekey, G.A., 1992, "Robotics and Neural Networks", Ch 6 in *Neural Networks for Signal Processing,* Edited by Kosko, B., Prentics-Hall international, pp. 161-188

Bennet, D.J. and Hollerbach, J.M, 1990, "Closed-loop kinematic calibration of the Utah-MIT Hand," *Experimental Robotics 1---The First Int. Symp.,* 1990, pp. 539-552

Bennet, D.J. and Hollerbach, J.M, 1991, "Autonomous Calibration of Single-Loop Closed Kinematic Chains Formed by Manipulators with Passive Endpoint Constraints," *IEEE Trans. on Robotics and Automation,* Vol. 7, No. 5, pp. 597-606

Bennett, D. J., Geiger, D. and Hollerbach, J.M., 1991, "Autonomous Robot calibration for hand-eye coordinaton", *Int'l J. of Robotics Research,* Vol: 10, No. 5, pp. 550-559, MIT Press

Bernhardt, R. and Albright, S.L., 1993, "Introduction to Robot Calibration", "Future of Calibration", in *Robot Calibration*, Edited by Bernhardt, R. and Albright, S.L., Chapman & Hall Press

Borderick, P.L. and Cipra, R.J., 1988, "A method for determining and correcting robot position and orientation errors due to manufacturing," *J. of Mechanisms, Transmissions, and Automation in Design,* Vol. 110, March, pp. 3-10

Borm, J H and Meng, C H, 1989, "Experimental study of observability of parameter errors in robot calibration," *Proc. IEEE Intl Conf. on Robotics and Automation,* IEEE, Scottsadale, Arizona, pp. 587-592

Borm, J H and Meng, C H, 1991, "Determination of optimal measurement configurations for robot calibration based on observability measure", *Int'l J. Robotics Research*, vol. 10 no. 1, pp. 51-63, MIT Press

Chen, J., and Chao, L.M., 1986, "Positioning error analysis for robot manipulators with rotary joints," *Proc. of IEEE International Conf. on Robotics and Automation,* IEEE, San Francisco, pp. 1011-1016

Cichocki A. and Unbehauen, R., 1992, "Neural Networks for Solving Systems of Linear Equations and Related Problems," *IEEE Trans. on Circuits and Systems,* Vol. 39, No. 2, pp. 124-138

Cichocki A. and Unbehauen, R., 1993, *Neural Networks for Optimization and Signal Processing*, John Wiley & Sons

Craig, J.J., 1986, *Introduction to Robotics---Mechanics and Control,* Addison Wesley, Reading, MA.

Daunicht, W.J., 1991, "Approximation of the inverse kinematics of an industrail robot by DEFAnet,", *Proc. of IEEE Int. Conf. on Neural Networks,* IEEE, pp.1995-2000

Denavit, J., and Hartenberg, R.S., 1955, "A kinematic notation for lower-pair mechanisms based on matrices," *Trans. of ASME J. Applied Mechanics,* June, pp. 215-221

Driels, M.R., Swayze, LW and Potter L.S. 1993, "Full-pose calibration of a robot manipulator using a coordinate measuring machine", *Int. J. of Adv Manuf Technol* 8: pp. 34-41

Driels, M.R. and Swayze, W., 1994, "Automated partial pose measurement system for manipulator calibration experiments", *IEEE Trans. on Robotics and Automation*, Vol. 10, No. 4, pp. 430-440

Easthope, C.E., 1964, *Three Dimensional Dynamics-A Vectorial Treatment,* Butterworth & Co (Publishers) Ltd., London

Everett, L.J., 1993, "Models for Diagosing Robot Error Sources", *Proc. of IEEE International Conf. on Robotics and Automation,* pp. 155-159

Everett, L.J., Colson, J.C. and Mooring, B.W., 1994, "Automatic Singularity Avoidance Using Joint Variations in Robot Task Modification", *IEEE Robotics & Automation Magazine,* Vol. 1, September 1994, pp 13-19

Everett, L.J., McCarroll, D.R., 1986, "Using finite element methods to approximate kinematic solutions for robot manipulators when closed form solutions are unattainable," *Proc. of IEEE Int. Conf. on Robotics and Automation,* Philadelphia, IEEE, pp. 798-800

Everett, L.J., and Suryohadiprojo, A.H., 1988, "A study of kinematic models for forward calibration of robot manipulators," *Proc. of IEEE International Conf. on Robotics and Automation,* IEEE, Philadelphia, pp. 798-800

Fu, K.S., Gonzalez, R.C. and Lee, C.S.G., 1987, *Robotics---Control, Sensing, Vision, and Intelligence*, McGraw-Hill, Inc.

Ghosh, J. and Shin, Y., 1992, "Efficient Higher-order Neural Networks for Classification and Function Approximation", *Int'l J. of Neural Systems,* Vol. 3, No. 4, pp 323-350

Giles, C. L. and Maxwell, 1987, "Learning, invariance, and generalization in high-order neural networks", *Applied Optics,* Vol. 26, No. 23, pp. 4972-4978

Goswami, A, Quaid, A, and Peshkin, M, 1993, "Complete parameter identification of a robot from partial pose information", *Proc. IEEE Int'l Conf. on Robotics and Automation*, Vol. 1: pp. 168-173

Grossberg, S., 1982, *Studies of Mind and Brain,* Reidel, Boston, BA

Guez, A. and Ahmad, Z., 1988, "Solution to the Inverse Kinematics Problem in Robotics by Neural Networks", *Proc. IEEE Int'l Conf. on Neural networks*, pp II-617-624

Guo, J. and Cherkassky, V., 1989, "A Solution to the Inverse Kinematic Problem in Robotics Using Neural Network Processing," *Proc of International Joint Conf. on Neural Networks,* Vol. II, pp. 299-304

Hayati, S.A., 1983, "Robot arm geometric link parameter estimation," *Proc. of the 22nd IEEE Conference on Decision and Control,* pp. 1477-1483

Hayati, S.A. and Roston G.P., 1986, "Inverse Kinematic Solution for Near-simple Robots and Its Application to Robot Calibration", *Recent Trends in Robotics: Modelling, Control, and Education*, Elservier Science Publishing Co., Inc. pp 41- 49

Hecht-Nielsen, R., 1990, *Neurocomputing,* Addison-Wesley Publishing Company

Ho, J. and Wu, C.H., 1987, "Robot Accuracy Compensator,", *Proc. IEEE International Conference on Robotics and Automation,* IEEE Philadephia, pp. 214-219

Hollerbach, J.J, 1989, "A review of kinematic calibration," in *The Robotics Review 1,* Khatib, O., Craig, J.J., and LozanolPerez, T., Eds Cambridge, MA: MIT Press, pp. 207-242

Hopfield, J.J and Tank, D.W., 1986, "Computing with Neural Circuits: A Model," *Sciences,* Vol. 233, pp. 625-633

Hopfield, J.J. and Tank, D.W., 1985, "Neural computation of decision in optimisation problems", *Biological Cybernetics*, 52, pp 141-152.

Hornik, K., Stinchocombe, M. , and White, H., 1990, "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks," *Neural Networks,* Vol. 3, pp. 551-560, Pergamon Press

Hornik, K., 1991, "Approximation Capabilities of Multilayer Feedforward Networks," *Neural Networks,* Vol. 4, pp. 251-257

Ibarra, R. and Perreira, N.D., 1986, "Determination of linkage parameter and pair variable errors in open chain kinematic linkages using a minimal set of pose measurement data," *J. of Mechanisms, Transmissions, and Automation in Design,* pp. 159-166, June 1986

Josin, G., 1988, "Neural-Space Generalization of a Topological Transformation", *Biological Cybernetics,* Vol. 59, pp. 283-290, Springer-Verlag Press

Judd, R.P. and Knasinski, A.B., 1991, "A technique to calibrate industrial robots with experimental verification", *IEEE Trans. on Robotics & Automation*, 6(1): pp. 20-30

Khalil, W., Gautier, M. and Enguehard, Ch., 1991, "Identifiable parameters and optimum configurations for robots calibration," *Robotica,* vol. 9, pp. 63-70, Cambridge University Press

Kirchner, H.O., Gurumoorthy, B. and Printz, F.B., 1987, "A Perturbation Approach to Robot Calibration", *Int. J. of Robotics Research* 6(4), pp 47-59

Kohonen, T., 1984, *Self-Organization and Associative Memory,* Springer-Verlag, New York, NY

Kumar, A., and Waldron, K.J., 1981, "Numerical Plotting of Surfaces of Positioning Accuracy of Manipulators," *Mechanism and Machine Theory,* vol. 16, no. 4, pp. 361-368

Kung, S.-Y., and Hwang, J.-N., 1989, "Neural Network Architecture for Robotics Applications," *IEEE Trans. on Robotics and Automation,* vol. 5, pp. 641-657

Kuperstein, M., 1988, "Neural network model for adaptive hand-eye coordination for single postures", *Sciences*, Vol. 239, pp. 1308-1311

Kuperstein, M., and Rubinstein, J., 1989, "Implementation of an adaptive neural controller for sensory motor coordination," *IEEE Control Syst. Mag.,* vol. 9, no. 3, pp. 25-30

Kozakiewicz, C., Ogiso, T., and Miyake, N., 1990, "Calibration Analysis of a Direct Drive Robot", *IEEE Int. Workshop on Intelligent Robot and Systems*, pp 213-220

Kozakiewicz, C., Ogiso, T., Miyake, N., 1991, "Partitioned Neural Network Architecture for Inverse Kinematic Calculation of a 6 DOF robot Manipulator", *Proc. of IEEE International Conf. on Neural Networks,* pp. 2001-2006

Landesman, E. and Hestenes, M., 1992, *Linear Algebra for Mathematics, Science, and Engineering,* Prentics-Hall International

Lau, K., Hocken, R., and Haynes, L., 1988, "Testing," *International Encyclopedia of Robotics, Application and Automation,* Dorf, R.C. and Nof, S.Y. Eds, John Wiley & Sons, Inc., pp. 1753-1769

Li, Y.-T. and Jiang Y.-S., 1993, "A Neural Network Based Resolved Motion Rate Control", *Proc. of the Third Int'l Sympos. on Measurement and Control in Robotics,* Session Cm. II-13-18, Turino, Italy

Martinetz, T.M., Ritter, H.J. and Schulten, K.J., 1990, "Three-Dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm", *IEEE Trans. on Neural Networks,* Vol. 1 , No. 1, pp. 131-136

Mirman, C.R. and Gupta, K.C., 1992, "Compensation of Robot Joint Variables using Special Jacobian Matrices", *J. of Robotic System* 9(1), pp. 113-137, John Wiley & Sons

Masters, T., 1993, *Practical Neural Networks Recipes in C++,* Academic Press

MathWorks Inc., 1992a, *Matlab User's Guide for UNIX Workstation*

MathWorks Inc., 1992b, *SIMULINK---Dynamic System Simulation Software User's Guide for the X Window System*

MathWorks Inc., 1992c, *Optimization TOOLBOX for Use with MATLAB User's Guide*

MathWorks Inc., 1993, *Neural Network TOOLBOX for Use with MATLAB User's Guide*

Miyamoto, H., Kawato, M., Setoyama, T., and Suzuki, R., 1988, "Feedback Error Learning Neural Networks for Trajectory Control of a Robotic Manipulator", *Neural Networks,* Vol. 1, pp. 251-265

Miyazaki, T., Maekawa, K. and Bamba, T., 1992, "Compensation of Positioning Errors of Industrial Robot Using Neural Network", *Proc. of the 23rd International Symposium on Industrial Robots,* Barcerona, Spain, pp. 377-381

Mooring, B.W., Roth Z.S., and Driels, M.R., 1991, *Fundamentals of Manipulator Calibration*, John Wiley & Sons, Inc., pp. 221-225

Mooring, B.W., 1983, "The effect of joint axis misalignment on robot positioning accuracy," *Proc. of the 1983 ASME Conputers in Engineering Conference, pp. 151-155*

Mooring, B.W. and Padavala, S.S., 1989, "The effects of kinematic model complexity on manipulator accuracy." *Proc. of IEEE Int. Conf. on Robotics and Automation,* Scottsdale, Arizona, IEEE, pp. 593-598

Mooring, B.W. and Tang, G.-R., 1984, "An improved method for identifying the kinematic parameters in a six axis robot," *Proc. of the 1983 ASME Conputers in Engineering Conference,* Las Vegas, Nevada, pp. 79-84

Newman, W.S. and Osborn, D.W., 1993, " A new method for kinematic parameter calibration via laser line tracking", *Proc. IEEE Int'l Conf. on Robotics and Automation*, Vol. 2: pp. 160-165

N-Nagy, F L and Siegler, T, 1987, *Engineering Foundation of Robotics*, Prentice-Hall International, pp 26-28.

Pao, Y., 1989, *Adaptive Pattern Recongnition and Neural Networks,* Addison-Wesley

Pathre, U.S., and Driels, M.R., 1990, "Simulation experiments in parameter identification for robot calibration," *Int J Adv Manuf Technol,* Vol. 5, No. 2, pp. 13-33, Springer-Verlag, London

Paul, R., 1981, *Robot Manipulators: mathematics, Programming, and Control,* MIT Press, Cambridge, MA

Prenninger, J.R., Vincze, M. and Gander, H, 1993, "Contactless position and orientation measurement of robot end-effector", *Proc. of IEEE Int. Conf. Robotics and Automation*, Atlanta, Vol.1: pp. 180-185

Rea, H.J., 1992, "POSE Correction - Robot System Simulation/Real Environment", *PhD Transfer Report,* Department of Mechanical, Manufacturing and Software Engineering, Napier University, Edinburgh, Scotland

Renders, J.M., Rossignol, E., Becquet, M., and Hanus, R., 1991, "Kinematic Calibration and Geometrical Parameter Identification for Robots," *IEEE Trans. on Robotics and Automation,* Vol. 7, No. 6, pp. 721-732

Renishaw Metrology Ltd., 1983, *User's Guide: LP2 and LP2H Probes with hard wired or induction transmission*

Roth, Z.S., Mooring, B.W., and Ravani, B., 1987, "An Overview of Robot Calibration," *IEEE J of Robotics and Automation,* Vol. RA-3, No. 5, pp. 377-385

Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1986, "Learning Internal Representation by Error Propagation," Ch 8 in *Parallel Distributed Processing,* Vol. 1, Rumelhart, D.E. and McClelland, J.L., MIT Press

Roth, Z.S., Mooring, B.W. and Ravani, B.,1987, "An Overview of Robot Calibration," *IEEE J. of Robotics and Automation*, Vol.Ra-3, No.5, 337-385

Shamma, J.S., 1985, "A method for Inverse Robot Calibration," *Master's Thesis,* Massachusetts Institute of Technology

Shamma, J.S. and Whitney, D.E., 1987, "A Method for Inverse Robot Calibration", *AMSE J. of Dynamic Systems, Measurement, and Control*, Vol.109, pp. 36-43

Shin, Y., 1992, "Efficient higher-order feedforward networks for function approximation and classification," *Ph.D Disseration*, Department of Electrical and Computer Engineering, The University of Texas at Austin, USA

Silma Inc., 1992, "Robot Calibration Package", *CimStation 4.3 User's Manual*

Stone, H W, 1986, "Kinematic modelling, identification and control of robotic manipulators," *PhD Dissertation,* Robotics Institute, Carnegie Mellon University

Stone, H W, 1992, *Kinematic modelling, identification and control of robotic manipulators,* Kluwer Academic Publishers, Chapter 5, pp 78-95

Stone, H.W., Sanderson, A.C. and Neuman, C.P., 1986, "Arm Signature Identification", *Proc. IEEE Int'l Conf. on Robotics and Automation*, San Francisco, CA, pp 41-48

Suh, C. and Radcliffe, C.W., 1978, *Kinematics and Mechanisms Design,* Wiley, New York

Tang G.-R and Mooring B W, 1992, "Plane-Motion Approach to Manipulator Calibration", *Int J. of Adv Manuf Technol,* Vol. 7, pp. 21-28

Tank, D and Hopfield, D, 1986, "Simple 'Neural' Optimisation Networks: An A/D Converter, Signal Decision Circuit, and Linear Programming Circuit", *IEEE Trans. on Circuits and System*, Vol. CAS-33, pp. 533-544

Tsai, R.-Y. and Lenz, R.K., 1989, "A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration," *IEEE Trans. on Robotics and Automation,* Vol. 5 No. 3, pp. 345-358

Veitschegger, W. K. and Wu, C.-H., 1986, "Robot Accuracy Analysis Based on Kinematics," *IEEE J. Robotics and Automation,* Vol. 2, No. 3, pp. 171-179

Veitschegger, W. K. and Wu, C.-H., 1988, "Robot Calibration and Compensation", *IEEE Int'l J of Robotics and Automation,* Vol. 4, No 6, pp. 643-656

Vira, N. and Tunstel, E., 1989, "Computer generation of geometrical error equations applicable for improvement of robots' positioning accuracy", *Intelligent Autonomous systems 2,* Amsterdam, pp.650-660

Vuskovic, M.I., 1989, "Compensation of Kinematic Errors Using Kinematic Sensitivities,", *Proc. IEEE International Conf. on Robotics and Automation*, Scottsdale, Arizona, pp. 745-750

Werbos, P.J., 1974, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," *PhD Dissertation in Statistics,* Harvard University

Whitney, D., Lozinsky and Rourke, J., 1986, "Industrial robot forward calibration method and results", *Trans. of the ASME J. of Dynamic Systems, Measur. and Control,* 108, pp. 1-8

Wu, C.H., 1983, "The kinematic error model for the design of robot manipulator," *Proc. of American Control Conference,* San Francisco, pp. 407-502

Wu, C.H., 1984, "A Kinematic CAD Tool for the Design and Control of a Robot Manipulator,' *Int'l J. of Robotics Research*, MIT Press, Vol. 3, No. 1, pp. 58-67

Wu, C.H., Ho, J. and Young, K.,1988, "Design of Robot Accuracy Compensator after Calibration", *IEEE Int'l Conf. on Robotics and Automation*, pp 780-785

Wu, C.M., Jiang, B.C. and Shiau, Y.R., 1993, "Controlling a Robot's Position Using Neural Networks," International J. of Advanced Manufacturing Technology, No. 8: pp. 216-226

Wu, G. and Wang, J., 1994, "A recurrent neural network for manipulator inverse kinematics computation," *Proc. of IEEE Int. Conf. on Neural Networks,* IEEE, Orlando, vol. 5, pp. 2715-2720,

Yeung, D.-T., and Bekey, G.A., 1989, "Using a Context-Sensitive Learning Network for Robot Arm Control," *Proc. IEEE International Conf. on Robotics and Automation*, pp. 1441-1447

Yeung, D.-T., 1989, "Handling Dimensionality and Nonlinearity in Connectionist Learning," *Ph.D Dissertation*, Computer Science Department, University of Southern California

Zhen, H., 1985, "Error analysis of robot manipulators and error transmission functions," *Proc. of the 15th International Conference on Industrial Robots,* pp. 873-878, Tokyo, 1985

Zhong, X.L., Lewis, J.M. and Rea, H., 1994, "Neuro-accuracy Compensator for Industrial Robots", *Proc. IEEE Int'l Conf. on Neural networks*, *IEEE World Congress on Computational Intelligence*, Orlando, Florida, Vol. 5 pp 2797-2802

Zhong, X.L. and Lewis, JM, 1994, "Kinematic Identification and Compensation of Robot Manipulators Using Neural Optimisation Networks", *Proc. Third Int'l Conf. on Automation, Robotics and Computer Vision*, Vol. III, pp 1472-1476, Singapore

Zhong, X. L. and Lewis, L.M., 1995, "A New Method for Autonomous Robot Calibration", *Proc. of IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 1790-1795, Nagoya, Japan

Zhong, X.L.and Lewis, JM, 1993, "A New Kinematic Measure of Robot Manipulators", *Proc. of the Third International Symp. on Measurement and Control in Robotics,* Session Cm. II-25-30, Torino, Italy

Zhong, X.L., Lewis, J.M., and N-Nagy, F, 1995, "Autonomous Robot Calibration Using a Trigger Probe ", Accepted for publication in the *Int'l J. of Robotics and Autonomous Systems,* Elsevier Science Press, Amsterdam, the Netherland

Zhong, X.L., Lewis, J.M., and N-Nagy, F, 1995, "Robot Inverse Calibration Using Artificial Neural Networks", Submitted to *Int. J. of Engineering Applications of Artificial Intelligence,* Pergamon Press

Zhuang, H., Hamano, F. and Roth, Z.S., 1989, "Optimal Design of Robot Accuracy Compensators", *Proc. IEEE Int. Conf. Robotics and Automation*, Arizona. pp 751-756

Zhuang, H., 1989, "Kinematic modeling, Identification, and Compensation of Robot manipulators", *Ph.D Dissertation,* Florida Atlantic University

Zhuang, H., Wang, L. and Roth, Z.S., 1993a, "Simultaneous calibration of a robot and a hand-mounted camera", *Proc. IEEE Int'l Conf. on Robotics and Automation*, Vol. 2: pp. 149-154

Zhuang, H., Wang, L. and Roth, Z.S., 1993b, "Error-Model-Based Robot Calibration Using a Modified CPC Model", *Robotics & Computer-Integrated Manufacturing,* Vol. 10, No. 4, pp. 287-299, Pergamon Press

Zhuang, H., Wang, K., and Roth, Z.S., 1994, "Optimal Selection of Measurement Configurations for Robot Calibration using Simulated Anealing," *Proc. of IEEE Int. Conf. on Robotics and Automation,* San Diago, CA., 1994

Ziegert, J. and Datseris, P., 1990, "Robot Calibration Using Local Pose Measurements," *International J. of Robotics and Automation,* Vol. 5, No. 2, pp. 68-76

Appendix 1. Forward Kinematics of Puma 560 Robot and Orientation Representations

```
function f = kinfwd(x)
% This function calculates the pose matrix T using forward kinematics based on  modified D-H model
%=================================================================

T = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
a = x(1:6);
d = x(7:12);
alfa = x(13:18);
theta = x(19:24);
belta = x(25:30);
NUM = 6;

for j = 1:NUM
        ct = cos(theta(j));
        st = sin(theta(j));
        sa = sin(alfa(j));
        ca = cos(alfa(j));
        sb = sin(belta(j));
        cb = cos(belta(j));
        TM = [ct*cb-sa*st*sb -st*ca ct*sb+sa*st*cb a(j)*ct;...
            st*cb+ct*sa*sb ca*ct st*sb-sa*cb*ct a(j)*st;...
            -ca*sb      sa  ca*cb     d(j);...
            0       0       0           1];
        T = T * TM;
end

% Return the pose transformation matrix
f = T;
```

Conversion of Euler angles to [n, s, o] orientation representation

```
function f = conv(e1,e2,e3)
% This function converts Euler angles (system II) (e1, e2, e3) into [n, s, o] orientation transformation
%=================================================================
se1 = sin(e1);
ce1 = cos(e1);
se2 = sin(e2);
ce2 = cos(e2);
se3 = sin(e3);
ce3 = cos(e3);
nx = ce1*ce2*ce3 - se1*se3;
ny = se1*ce2*ce3 + ce1*se3;
nz = -se2*ce3;
sx = -ce1*ce2*se3 - se1*ce3;
sy = -se1*ce2*se3 + ce1*ce3;
sz = se2*se3;
ox = ce1*se2;
oy = se1*se2;
oz = ce2;
% Return orientation transformation matrix
f = [nx sx ox; ny sy oy; nz sz oz];
```

Appendix 2. Closed-form Inverse Kinematic Solution of Puma Robot

```
function f = kininv(x)
% This function is used to find joint vector for a given Cartesian Location (x) using nominal model
%==========================================================================
PI = 3.141592;
%nominal model
a = [0 431.800 -20.320 0 0 0]./10;
d = [0 149.090 0 433.070 0 0]./10;
alfa = [-PI/2 0 PI/2 -PI/2 PI/2 0];
nx = x(1); ny = x(2); nz = x(3);
sx = x(4); sy = x(5); sz = x(6);
ax = x(7); ay = x(8); az = x(9);
av = [ax;ay;az]; sv = [sx;sy;sz]; nv = [nx;ny;nz];
% Tool offset x, y, z
offset_tx = 0.5;
offset_ty = -0.15;
offset_tz = 35.859;
px = x(10) - offset_tz*ax - offset_tx*nx - offset_ty*sx;
py = x(11) - offset_tz*ay - offset_tx*ny - offset_ty*sy;
pz = x(12) - offset_tz*az - offset_tx*nz - offset_ty*sz;
% Cofigurations indicators
ARM = +1;                       % = -1 LEFT arm, = +1 RIGHT arm
ELBOW = +1;                     % = +1 ABOVE arm, = -1 BELOW arm
WRIST = +1;                     % = +1 WRIST DOWN, = -1 WRIST UP
sinth1 = (-ARM*py*sqrt(abs(px*px+py*py-d(2)*d(2)))-...
                px*d(2))/(px*px+py*py);
costh1 = (-ARM*px*sqrt(abs(px*px+py*py-d(2)*d(2)))+...
                py*d(2))/(px*px+py*py);
th1 = atan2(sinth1,costh1);
s1 =sinth1; c1 = costh1;
if (-px*c1-py*s1) < 0
        ARM = -1;
        sinth1 = (-ARM*py*sqrt(abs(px*px+py*py-d(2)*d(2)))-...
                px*d(2))/(px*px+py*py);
        costh1 = (-ARM*px*sqrt(abs(px*px+py*py-d(2)*d(2)))+...
                py*d(2))/(px*px+py*py);
        th1 = atan2(sinth1,costh1);
end
K = ARM*ELBOW;
R1 = sqrt(abs(px*px+py*py+pz*pz-d(2)*d(2)));
R2 = sqrt(abs(px*px+py*py-d(2)*d(2)));
sina = -pz/R1;
cosa = -ARM*R2/R1;
cosb = (a(2)*a(2)+R1*R1-(d(4)*d(4)+a(3)*a(3)))/(2*a(2)*R1);
sinb = sqrt(1-cosb*cosb);

sinth2 = sina*cosb+K*cosa*sinb;
costh2 = cosa*cosb-K*sina*sinb;
th2 = atan2(sinth2,costh2);

cosf = (a(2)*a(2)+d(4)*d(4)+a(3)*a(3)-R1*R1)/...
                (2*a(2)*sqrt(d(4)*d(4)+a(3)*a(3)));
sinf = K*sqrt(1-cosf*cosf);
sinb1 = d(4)/sqrt(d(4)*d(4)+a(3)*a(3));
```

```
cosb1 = abs(a(3))/sqrt(d(4)*d(4)+a(3)*a(3));

sinth3 = sinf*cosb1-cosf*sinb1;
costh3 = cosf*cosb1+sinf*sinb1;
th3 = atan2(sinth3,costh3);

if ARM*(d(4)*costh3-a(3)*sinth3) < 0
        ELBOW = -1; K = ARM*ELBOW;
        sinth2 = sina*cosb+K*cosa*sinb;
        costh2 = cosa*cosb-K*sina*sinb;
        th2 = atan2(sinth2,costh2);
        sinf = K*sqrt(1-cosf*cosf);
        sinth3 = sinf*cosb1-cosf*sinb1;
        costh3 = cosf*cosb1+sinf*sinb1;
        th3 = atan2(sinth3,costh3);
end
s1= sin(th1); c1 = cos(th1);
s23 =sin(th2+th3); c23 = cos(th2 +th3);
% Decision equation, to decide symbol of M
R3 = [c1*c23 -s1 c1*s23;. s1*c23 c1  s1*s23;  -s23    0   c23];
z3 = R3(:,3);
xv = cross(z3,av);
ip = sx*xv(1)+sy*xv(2)+sz*xv(3);
if ip == 0
        EM = nx*xv(1)+ny*xv(2)+nz*xv(3); else
        EM = ip;
end
if EM >= 0
        M = WRIST*(+1);
else
        M = WRIST*(-1);
end
sinth4 = M*(cos(th1)*ay-sin(th1)*ax);
costh4 = M*(cos(th1)*cos(th2+th3)*ax+...
        sin(th1)*cos(th2+th3)*ay-sin(th2+th3)*az);
th4 = atan(sinth4/costh4);
R4 = R3*[costh4 0 -sinth4;sinth4 0 costh4; 0 -1 0];
z4 = R4(:, 3);
ip4 = sx*z4(1)+sy*z4(2)+sz*z4(3);
if ip4 < 0
        WRIST = -1;M = -1*M;
        sinth4 = M*(cos(th1)*ay-sin(th1)*ax);
        costh4 = M*(cos(th1)*cos(th2+th3)*ax+...
        sin(th1)*cos(th2+th3)*ay-sin(th2+th3)*az);
        th4 = atan(sinth4/costh4);
end
sinth5 = (cos(th1)*cos(th2+th3)*cos(th4)-sin(th1)*sin(th4))*...
        ax+(sin(th1)*cos(th2+th3)*cos(th4)+cos(th1)*sin(th4))*...
        ay-cos(th4)*sin(th2+th3)*az;
costh5 = cos(th1)*sin(th2+th3)*ax+sin(th1)*sin(th2+th3)*ay+...
        cos(th2+th3)*az;
th5 = atan2(sinth5,costh5);
sinth6 = (-sin(th1)*cos(th4)-cos(th1)*cos(th2+th3)*sin(th4))*...
        nx+(cos(th1)*cos(th4)-sin(th1)*cos(th2+th3)*sin(th4))*...
        ny+(sin(th4)*sin(th2+th3))*nz;
costh6 = (-sin(th1)*cos(th4)-cos(th1)*cos(th2+th3)*sin(th4))*...
        sx+(cos(th1)*cos(th4)-sin(th1)*cos(th2+th3)*sin(th4))*...
        sy+(sin(th4)*sin(th2+th3))*sz;
```

```
        th6 = atan2(sinth6,costh6);
% Return joint variable vector
f = [th1 th2 th3 th4 th5 th6];
```

Appendix 3. Ordinary Jacobian Matrix for Puma 560 Robot

```
function f = jcbrec(x)
% The ordinary Jacob matrix of 6 DOF Puma-560 robot using recursive procedure
% ========================================================

PI = 3.141592;
%nominal model
a = [0 431.800 -20.320 0 0 0]./10;
d = [0 149.090 0 433.070 0 113.25]./10;
alfa = [-1.5708 0 1.5708 -1.5708 PI/2 0];
theta = x;
[1, NUM] = size(x);
JP = []; PP = []; JO = []; RR = [];
TT = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
T = [];

for j = 1:NUM
        ct = cos(theta(j)); st = sin(theta(j));
        sa = sin(alfa(j)); ca = cos(alfa(j));
        % Transformation matrix j wrt j-1
        TM = [ct -ca*st sa*st a(j)*c(j);...
                st ca*c(j) -sa*ct a(j)*st;...
                0    sa  ca        d(j);...
                0       0       0        1];
        % Record of rotation j wrt j-1
        RR = [RR TM];
        % Totaion matrix j wrt 0
        TT = TT*TM;
        % Rrocord of transformation matrix
        T = [T TT];
        P = [a(j)*ct; a(j)*st; d(j)];          % Position vector j wrt j-1
        PP = [PP P];                           % Record of position vectors
        OM = TT(1:3,1:3)*[0;0;1];              % jth revolute joint z-axis vector
        JO  = [JO OM];                         % Record of joint axis vectors
end

TINV = TT;
TINM = [];
j = NUM;

while j > 0
        RM = RR(:, (j-1)*4+1:j*4);
        TINV = RM*TINV;
        TINM = [TINM TINV(1:3,4)];     % Record of the 4th Column of T matrix wrt TCP
        j = j-1;                       % Reverse order j = NUM to 1
end

for j = 1:NUM
        % Position vector j wrt 0
        JPJ = T(1:3,(j-1)*4+1:j*4-1)*[-TINM(2, NUM-j+1);...
                -TINM(1, NUM-j+1); 0];
        JP = [JP JPJ];
```

```
end

% Return the ordinary Jacobian matrix
f = [JP; JO];
```

Appendix 4. Special Jacobian Matrix of Puma 560 Robot

```
function f = sjcbrec(x)
% The special Jacob matrix of Puma-560 robot using recursive procedure based on the modified D-H
%=============================================================================
PI = 3.141592;
%nominal model
a = [0 431.800 -20.320 0 0 0]./10;
d = [0 149.090 0 433.070 0 0]./10;
alfa = [-1.5708 0 1.5708 -1.5708 PI/2 0];
% Tool offset
%offset_tx = 2; offset_ty = 0; offset_tz = 11.125;       %CMM calibration
offset_tx = 0.5; offset_ty = -0.15; offset_tz = 35.859; %probe calibration
theta = x;
% Initialization of variables
z_x_p = [];
p_vecs = [];
zaxis = [];
RR = [];
xaxis = [];
x_x_p = [];
yaxis = [];
y_x_p = [];
TT = [1 0 0; 0 1 0; 0 0 1];
T = [];
NUM = 6;

for j = 1:NUM
        c(j) = cos(theta(j));
        s(j) = sin(theta(j));
        sa(j) = sin(alfa(j));
        ca(j) = cos(alfa(j));

        % Rotation matrix j wrt j-1
        RM = [c(j) -ca(j)*s(j) sa(j)*s(j);...
              s(j) ca(j)*c(j) -sa(j)*c(j);...
              0     sa(j)  ca(j)];

        % Record of rotation j wrt j-1
        RR = [RR RM];
        % Record of rotation matrix
        T = [T TT];
        % (j-1)th revolute joint z-axe vector
        zaxis_1 = TT*[0;0;1];
        % Rotaion matrix j wrt 0
        TT = TT*RM;
        % jth joint x-axe vector
        xaxis_0 = TT*[1;0;0];
        % jth joint y-axe vector
        yaxis_0 = TT*[0;1;0];
```

```
        % Position vector j wrt j-1
        p_vec = [a(j)*c(j); a(j)*s(j); d(j)];
        % Record of position vectors
        p_vecs = [p_vecs p_vec];
        % Record of z-axis vector
        zaxis = [zaxis zaxis_1];
        % Record of x-axis vector
        xaxis = [xaxis xaxis_0];
        % Record of y-axis vector
        yaxis = [yaxis yaxis_0];
end

T = [T TT];
% Position vector connecting hand center to the origin of (j-1)th joint frame
phtj_1 = [offset_tx; offset_ty; offset_tz];
phtjs_1 = [];
phtjs_0 = [];
phtj_0 = [0; 0; 0];
j = NUM;

while j > 0
        RM = RR(:, (j-1)*3+1:j*3);
        % Position vector wrt j-1
        phtj_1 = RM*phtj_1 + p_vecs(:,j);
        % Record of position vector wrt j-1
        phtjs_1 = [phtjs_1 phtj_1];
        % Record of position vector wrt j
        phtjs_0 = [phtjs_0 phtj_0];
        % Reverse order j = 6 to 1
        j = j-1;
        phtj_0 = phtj_1;
end

for j = 1:NUM
        %  jth vector (z X p) wrt base(0)
        z_x_p_1 = T(:,(j-1)*3+1:j*3)*[-phtjs_1(2,(NUM-j+1));...
                phtjs_1(1,(NUM-j+1)); 0];
        % (j+1)th vector (x X p) wrt base(0)
        x_x_p_0 = T(:,j*3+1:(j+1)*3)*[0; -phtjs_0(3,(NUM-j+1));...
                phtjs_0(2,(6-j+1))];
        % (j+1)th vector (y X p) wrt base(0)
        y_x_p_0 = T(:,j*3+1:(j+1)*3)*[phtjs_0(3,(NUM-j+1)); 0;...
                -phtjs_0(1,(NUM-j+1))];
        % Record of vector which is corss porduct between z and position vector p
        z_x_p = [z_x_p z_x_p_1];
        % Record of vector which is cross porduct between x and position vector (p-d)
        x_x_p = [x_x_p x_x_p_0];
        % Record of vector which is cross product between y and (p-d)
        y_x_p = [y_x_p y_x_p_0];
end

% Jacobian matrix w.r.t. theta
jth = [z_x_p; zaxis];
% Jacobian matrix w.r.t. alpha
jal = [x_x_p; xaxis];
% Jacobian matrix w.r.t. belta
jbt = [y_x_p; yaxis];
% Jacobian matrix w.r.t. link length a
```

```
ja = [xaxis; zeros(3,6)];
% Jacobian matrix w.r.t. link offest d
jd = [zaxis; zeros(3,6)];

% Modified D-H representation
jd(:,2) = jbt(:,2);

% Return the special Jacobian matrix
f = [ja jal jd jth];
```

Appenix 5. Listed program for data collection using a trigger Probe

```
; This program is used to collect data using a trigger probe to probe the z-axis constraint plane
listp
.PROGRAM lineup
  1       SPEED 50  ALWAYS
  2       FINE ALWAYS
  3       i = 1
  4       MOVES left
  5       CALL probe
  6       i = i+1
  7       MOVES right
  8       CALL probe
  9       MOVES left
 10       RETURN
.END
.PROGRAM main
  1       TOOL probe
  2       SPEED 50  ALWAYS
  3       FINE ALWAYS
  4       TYPE /C25
  5       MOVE #start
  6       DELAY 0.5
  7       HERE start
  8       DELAY 0.5
  9 ;     PROMPT "Enter Number of Poses : ", poses
 10 ;set global variables
 11       xmax = 200
 12       xmin = 0
 13       ymax = 200
 14       ymin = -200
 15       zmax = 0
 16       zmin = 0
 17       omax = 45
 18       omin = -25
 19       amax = 45
 20       amin = -25
 21       tmax = 45
 22       tmin = -30
 23       FOR i = init TO poses
 24          CALL random
 25          TYPE /C25, "Moving to Pose : ", i
 26          MOVE random.loc
 27          CALL probe
 28          MOVE #start
```

```
 29      END
 30      TOOL
 31      RETURN
.END
.PROGRAM probe
  1      DELAY 0.5
  2      SPEED 50  ALWAYS
  3      TYPE "Probing for Position ..."
  4      COARSE ALWAYS
  5      WHILE SIG(-1001) DO
  6        SET d = SHIFT(HERE BY xdist, ydist, zdist)
  7        MOVE d
  8      END
  9
 10      FINE ALWAYS
 11      DISABLE CP
 12      IF SIG(1001) THEN
 13        SET d = SHIFT(HERE BY (-xdist/10), (-ydist/10), (-zdist/10))
 14        MOVE d
 15      END
 16      DELAY 0.2
 17      CALL record.loc
 18      ENABLE CP
 19      TYPE "Position ", i, " Stored"
 20      SET temp = SHIFT(HERE BY 0, 0, 50)
 21      MOVE temp
 22      SPEED 100  ALWAYS
 23      RETURN
.END
.PROGRAM random
  1 ;program to compose a transformation using a random value
  2 ;for x,y,z,o,a,t
  3
  4      x = (RANDOM*(xmax-xmin))+xmin
  5      x = x*(-1)
  6      y = (RANDOM*(ymax-ymin))+ymin
  7
  8      o = (RANDOM*(omax-omin))+omin
  9      a = (RANDOM*(amax-amin))+amin
 10      t = (RANDOM*(tmax-tmin))+tmin
 11      DECOMPOSE temp[] = start
 12      temp[0] = temp[0]+x
 13      temp[1] = temp[1]+y
 14      temp[2] = temp[2]+z
 15      temp[3] = temp[3]+o
 16      temp[4] = temp[4]+a
 17      temp[5] = temp[5]+t
 18      SET random.loc = TRANS(temp[0], temp[1], temp[2], temp[3], temp[4], temp[5])
 19      DELAY 0.5
 20      RETURN
END
.PROGRAM record.loc
  1      DELAY 0.5
  2      HERE loc[i]
  3      DELAY 0.5
  4      HERE #loc[i]
  5      DELAY 0.5
  6      RETURN
```

```
.END

.listl
; Recorded robot location data: X/#Jt1   Y/#Jt2   Z/#Jt3   O/#Jt4   A/#Jt5   T/#Jt6
 datum    -573.09   21.66 -470.84  89.714  89.846   0.000
#datum     12.936 -142.169  -5.625  -0.099 -32.349  13.30
 loc[1]   -539.13  269.69 -470.41  73.663  70.153   9.058
 loc[2]   -574.41   -2.56 -470.97  65.550  88.006  30.031
 loc[3]   -631.03  137.03 -470.75  63.875  78.300   8.641
 loc[4]   -624.38  381.59 -469.16 100.272  69.016  -1.719
 loc[5]   -662.09  393.94 -469.81 -133.044  87.682 -141.801
 loc[6]   -658.41  169.34 -470.22  87.792  78.975 -19.995
 loc[7]   -600.50  384.78 -470.13  70.807  88.830  21.736
 loc[8]   -536.78  116.22 -470.47 -63.622  64.742 -156.022
 loc[9]   -642.03  313.88 -471.00 -88.116  69.000 -137.955
 loc[10]  -677.09  240.19 -470.75  54.470  71.082  10.184
 loc[11]  -538.81  148.47 -470.97  58.255  84.441 -27.114
 loc[12]  -562.06  141.69 -470.38 113.044  77.866  23.895
 loc[13]  -610.59  185.16 -470.38 -70.626  55.931 -180.000
 loc[14]  -562.06  214.75 -470.72  55.195  75.591  38.655
 loc[15]  -696.06   20.56 -471.41  55.723  71.290 -20.017
 loc[16]  -652.19  327.72 -469.72 111.204  83.491  -7.295
 loc[17]  -556.47  250.75 -469.97 108.490  78.184   5.422
 loc[18]  -677.13  223.72 -468.59 -128.815  57.480  176.408
 loc[19]  -600.72  285.00 -469.47 114.038  75.591  29.888
 loc[20]  -607.69  242.78 -469.19 -90.802  44.731 -152.979
 loc[21]  -515.66   51.47 -470.84 106.705  78.140  11.772
 loc[22]  -590.66   93.84 -471.03 -84.716  70.444  165.998
 loc[23]  -598.19  158.50 -470.38  85.605  70.175 -25.406
 loc[24]  -668.78  353.56 -471.34 -77.553  66.396 -170.118
 loc[25]  -570.84   33.81 -470.66 -104.255  75.531  178.616
 loc[26]  -533.63  115.13 -470.19 -61.029  51.680  160.395
 loc[27]  -509.63  109.03 -469.41 -127.793  65.028  166.361
 loc[28]  -666.63  331.00 -470.06 -112.868  86.556 -135.989
 loc[29]  -577.81  349.66 -470.47  112.538  76.970  29.553
 loc[30]  -543.28  349.59 -470.34 -105.815  88.742  166.311
 loc[31]  -554.34    6.16 -471.28 -87.407  69.977  155.814
 loc[32]  -621.91   57.66 -470.41 -69.802  65.347 -150.029
 loc[33]  -526.00  371.16 -470.94 -75.701  57.437 -164.680
 loc[34]  -504.13   60.91 -470.84  110.550  79.970   6.152
 loc[35]  -695.88  197.22 -469.72  91.879  69.099   8.915
 loc[36]  -615.38  311.72 -470.78 -90.000  82.326  147.101
 loc[37]  -507.91  155.81 -470.88  94.570  83.452  23.961
 loc[38]  -664.09   49.44 -470.88 -77.217  66.610 -178.308
 loc[39]  -578.84  138.44 -470.56  112.368  85.660  38.419
 loc[40]  -524.31  205.88 -470.44 -94.614  71.933  165.317
 loc[41]  -528.97  210.31 -470.75  91.126  89.835   0.000
 loc[42]  -512.31   79.94 -471.16  29.932  89.764   0.000
 loc[43]  -526.03  373.88 -470.38  64.869  88.314 -21.780
 loc[44]  -659.91  159.09 -470.28  103.387  86.193  -9.355
 loc[45]  -654.31  319.41 -470.28  56.646  79.579 -14.178
 loc[46]  -596.88   17.53 -471.09 -84.408  44.621 -146.942
 loc[47]  -523.69  357.50 -470.84 -94.219  86.342  164.185
 loc[48]  -607.34  301.28 -470.34 -74.438  46.176  167.305
 loc[49]  -594.91  385.13 -469.16  106.452  77.970  40.891
 loc[50]  -521.56  243.13 -470.34 -88.308  66.830 -172.227
 loc[51]  -606.00   24.63 -470.97  78.519  65.330  36.178
 loc[52]  -576.94   36.19 -470.91  81.447  86.710  11.140
```

loc[53]	-511.53	393.97	-470.03	-85.441	51.894	-162.614
loc[54]	-522.72	2.56	-470.28	-63.078	66.187	178.934
loc[55]	-670.50	208.09	-470.13	-116.038	54.981	-163.603
loc[56]	-647.59	94.94	-470.31	93.038	79.124	26.730
loc[57]	-601.78	36.66	-470.44	106.430	78.799	16.408
loc[58]	-506.25	288.13	-471.53	-72.971	74.185	164.669
loc[59]	-625.94	291.19	-469.56	-86.721	48.488	168.887
loc[60]	-546.13	91.88	-471.31	-89.594	55.756	-154.204
#loc[1]	-12.876	-156.209	24.049	1.269	-67.659	11.83
#loc[2]	14.963	-142.784	-4.092	2.230	-34.684	67.65
#loc[3]	-1.417	-154.298	22.083	5.702	-58.563	29.93
#loc[4]	-15.370	-171.755	57.761	-8.948	-85.139	-25.09
#loc[5]	-18.809	-158.176	31.943	-1.214	-51.658	63.18
#loc[6]	-1.934	-157.506	29.493	0.060	-63.018	-19.75
#loc[7]	-20.429	-153.946	22.599	-0.033	-49.829	20.58
#loc[8]	-4.993	-139.927	-26.840	123.607	-15.469	-129.48
#loc[9]	-17.287	-146.821	-0.110	27.532	-14.755	-2.69
#loc[10]	-11.212	-171.101	56.783	7.723	-83.090	32.39
#loc[11]	-1.752	-143.240	-3.219	4.466	-38.452	-0.75
#loc[12]	3.400	-147.508	6.147	-5.279	-50.208	8.04
#loc[13]	-13.458	-145.668	-17.183	124.096	-21.517	-154.16
#loc[14]	-9.816	-152.122	16.353	7.158	-57.541	59.10
#loc[15]	4.752	-165.009	43.748	12.134	-73.834	14.07
#loc[16]	-13.310	-158.500	32.448	-4.285	-59.387	-39.46
#loc[17]	-7.185	-149.551	11.228	-6.422	-52.531	-15.85
#loc[18]	5.367	-146.662	-10.250	-93.675	-22.055	129.70
#loc[19]	-8.190	-155.517	24.379	-8.690	-61.408	2.67
#loc[20]	-13.442	-150.452	-24.225	166.036	-40.089	-151.17
#loc[21]	11.816	-143.481	-3.988	-1.439	-44.357	8.01
#loc[22]	5.290	-139.598	-20.786	0.000	-0.060	-14.00
#loc[23]	-1.642	-158.000	28.098	0.994	-69.906	-23.16
#loc[24]	-23.022	-151.397	9.899	36.398	-23.049	-57.40
#loc[25]	16.837	-138.246	-21.116	-42.429	-11.025	70.81
#loc[26]	-12.299	-146.041	-28.740	133.841	-34.541	166.46
#loc[27]	20.544	-138.565	-34.442	-106.957	-22.066	150.35
#loc[28]	-14.293	-154.413	23.494	-0.720	-45.676	53.06
#loc[29]	-14.233	-155.440	24.379	-8.981	-59.749	-1.92
#loc[30]	-19.605	-146.591	5.477	0.137	-37.628	-17.53
#loc[31]	18.528	-137.840	-28.713	42.731	7.965	-51.62
#loc[32]	4.955	-141.641	-18.583	123.030	-7.515	-107.19
#loc[33]	-35.189	-147.112	-9.635	88.396	-24.164	-117.52
#loc[34]	11.332	-141.713	-8.108	-2.483	-40.111	-1.04
#loc[35]	-3.016	-174.825	64.127	-1.747	-90.132	4.33
#loc[36]	-15.809	-147.914	7.767	3.873	-32.520	-51.84
#loc[37]	-0.192	-141.724	-7.295	-0.890	-37.513	19.95
#loc[38]	7.498	-143.146	-12.475	62.787	-2.357	-65.92
#loc[39]	2.148	-144.965	1.329	-2.313	-40.463	20.03
#loc[40]	-6.658	-138.038	-24.851	146.058	-1.137	-162.70
#loc[41]	-6.465	-141.509	-7.333	-0.308	-31.207	-7.33
#loc[42]	8.218	-138.494	-15.754	0.505	-25.653	67.83
#loc[43]	-22.242	-148.008	8.882	0.126	-42.561	-18.97
#loc[44]	-0.544	-152.078	18.259	-1.198	-49.889	-22.41
#loc[45]	-15.754	-163.213	42.028	3.362	-68.796	1.92
#loc[46]	19.116	-151.622	-35.228	12.189	52.059	35.08
#loc[47]	-21.610	-144.926	1.099	2.027	-32.695	-34.88
#loc[48]	-28.702	-152.040	-9.673	116.840	-32.800	-168.85
#loc[49]	-17.265	-158.923	32.563	-7.405	-63.886	11.01
#loc[50]	-13.914	-139.570	-24.983	138.697	-9.229	-145.69

```
#loc[51]    7.570 -161.071   32.899    8.091  -75.438    51.56
#loc[52]   10.909 -143.586   -2.098    1.802  -37.436    29.16
#loc[53]  -37.700 -148.123  -14.271  115.428  -27.356  -136.34
#loc[54]   11.898 -138.373  -33.096  157.286  -15.727  -172.91
#loc[55]    4.098 -146.393  -15.699   52.581   21.270    -8.79
#loc[56]    4.845 -154.616   23.038    0.396  -59.299    28.29
#loc[57]   11.514 -149.689   11.602   -1.198  -53.075    12.28
#loc[58]  -21.297 -140.658  -14.573   38.985  -15.590   -90.54
#loc[59]  -20.819 -149.634  -14.843  144.723  -27.944  -177.63
#loc[60]   10.025 -143.800  -37.310    9.360   35.338    26.11
 probe       5.00    0.00   302.34   90.000  -90.000    0.000
 random.loc -539.72   98.03  -368.66  -90.324   56.184 -153.787
left     -718.97   354.47    58.38 -180.000   76.003   59.238
 right    -358.00   354.50    58.38 -179.984   75.992   59.233
#start     -5.345 -127.238  -14.821    0.000  -37.947    10.69
 start    -500.22   197.41  -368.66   73.982   89.995    0.000
```

PUBLISHED PAPER(S)
NOT INCLUDED WITH THESIS