“Novel Artificial Neural Network Application for Prediction of Inverse Kinematics of Manipulator”

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

FOR THE REQUIREMENT FOR THE DEGREE OF

Master of Technology

In

Production Engineering

By

PANCHANAND JHA

Department of Mechanical Engineering

National Institute of Technology

Rourkela – 8

2009

“Novel Artificial Neural Network Application for Prediction of Inverse Kinematics of Manipulator”

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

FOR THE REQUIREMENT FOR THE DEGREE OF

Master of Technology

In

Production Engineering

By

PANCHANAND JHA

Under the guidance of

Dr. B.B. BISWAL

Professor, Department of Mechanical Engineering

Department of Mechanical Engineering

National Institute of Technology

Rourkela – 8

2009

NATIONAL INSTITUTE OF TECHNOLOGY

ROURKELA – 769008

INDIA

CERTIFICATE

This is to certify that the thesis entitled, **"Novel Artificial Neural Network Application for Prediction of Inverse Kinematics of Manipulator** "submitted by **Mr. PANCHANAND JHA in** partial fulfillment of the requirement for the award of **Master of Technology** Degree in **Mechanical Engineering** with specialization in **Production engineering** at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/ Institute for the award of any degree or diploma.

Date:

Prof.B.B Biswal
Mechanical Engineering Department
National Institute of Technology
Rourkela-769008

ACKNOWLEDGEMENTS

No thesis is created entirely by an individual, many people have helped to create this thesis and each of their contribution has been valuable. I express my sincere gratitude and indebtedness to my thesis supervisor, **Dr. B.B.Biswal**, for proposing
this area of research, for his kind and valuable guidance in the completion of the present work. His consistent support and intellectual guidance made me energize and to innovate new ideas.

I am grateful to Dr. R.K Sahu, Professor and Head, Mechanical Engineering Department, NIT, Rourkela for his excellent support during my work. I am also thankful to all the faculty members and staff of Mechanical Engineering Department, for providing me support and advice in preparing my thesis work. Last, but not the least I would like to thank my parents and my friends who are involved directly and indirectly in successful completion of the present work.

Panchanand Jha
Roll no.207me206
M.Tech.(Production Engg.)
Mechanical Engineering Department
NIT Rourkela - 769008

CONTENTS

TITLE	PAGE NO.
LIST OF FIGURES	V
LIST OF TABLES	V
LIST OF ABBREVIATIONS	VI
ABSTRACT	VII
1 INTRODUCTION	1
1.1Background of the Work	7
1.2Objectives	7
1.3Methodology	7
1.4Scope of the Present Work	8
1.5Organization of the Thesis	8
1.6Summary	9
2 LITERATURE REVIEW	10
2.1Introduction	10
2.2Summary	18
3 INTRODUCTIONS TO KINEMATICS OF MANIPULATOR	19
3.1 Introduction	19
3.2 Direct Kinematics	22
3.3 Inverse Kinematics	24
3.4 Summary	27
4 INTRODUCTIONS TO NEURAL NETWORK ARCHITECTURE	28

4.1 Introduction of Artificial Neural Network 29

4.2 Why use Neural Networks? 30

4.3 Neuron Model 31

4.4 Network Layers 31

4.5 Activation Functions 32

4.6 Sigmoidel Function 33

4.7 Feed Forward Networks 34

4.8 Learning 35

4.9 Back Propagation Algorithm 35

4.10 An Overview of Training in Back Propagation 36

4.11 Training Algorithm 37

4.12 Adjusting Weights of Network 37

4.13 Testing 38

4.14 Summary 39

5 NOVEL ANN APPLICATIONS FOR PREDICTING OF INVERSE KINEMATICS OF MANIPULATOR 40

5.1 Introduction 40

5.2 Application of MLP (multi-layer perceptron) 40

5.3 Application of PPN (polynomial preprocessor neural network) 46

5.4 Summary 47

6 SIMULATION RESULTS AND PERFORMANCE ANALYSIS 48

7 CONCLUSIONS AND SCOPE FOR FUTURE WORK 55

8 REFERENCE 56

LIST OF FIGURE & TABLES

1.1 Schematic Representation of Forward and Inverse kinematics.

1.2 Feed-back Neural Network and Feed-Forward Neural Network

3.1 A Schematic of a Planar Manipulator with Three Revolute Joints.

3.2 Representation of 3R Planar Manipulator in Cartesian Space.

3.3 The Two Inverse kinematics Solutions for the 3*R* Manipulator: “elbow-up” Configuration (σ =+1) and the “elbow-down” Configuration (σ = -1)

4.1 Neural Network Model Fig 6: Simple Neuron Model

4.2 A Multi-Layer Neuron Model

4.3 A Multi-Layer Neuron Model

4.4 Sigmoidal Function

4.5 Common Activation Functions

4.6 An example of Feed Forward Network

4.7 Basic Model of Neuron using Back Propagation

4.8 Multilayer Back Propagation Neural Network

4.9 Adjusting Weights of Output Layer

4.10 Adjusting Weights of Hidden Layer

5.1 Architecture for Neuron

5.2 Architecture of MLP

5.3 Architecture for Back-Propagation

5.4 The Influence of the Learning Rate on the Weight Changes.

5.5 Flow Chart for MLP

5.6 Polynomial Perceptron Network.

6.1 Mean Square Error for θ_1

6.2 Graph for Matching of Desired and Predicted Values of θ_1

6.3 Mean Square Error for θ_2

6.4 Graph for Matching of Desired and Predicted Values of θ_2

6.5 Comparison Table for Desired Data and Predicted Data by ANN

6.6 Mean square error for $\theta 1$

6.7 Mean square error for $\theta 2$

6.8 Graph for matching of desired and predicted values of θ_2

6.9 Graph for matching of desired and predicted values of $\theta 1$

LIST OF ABBREVIATIONS

ANN-- -	Artificial Neural Network
AI------	Artificial intellegence
MLP--	Multilayer perceptrons
PPN--	Polynomial Pre-Processor
FLANN--	Functional Link Artificial Neural Network
RBF--	Radial Basis Function
MIMO--	Multi input multi output
DOF--	Degree Of Freedom
NN--	Neural Network
IK--	Inverse Kinematics
FK--	Forward Kinematics
BP--	Back Poropagation
ANFIS--	Artificial Neuro- Fuzzy Inference System
6R--	6 Revolute

ABSTRACT

The robot control problem can be divided into two main areas, kinematics control (the coordination of the links of kinematics chain to produce desire motion of the robot), and dynamic control (driving the actuator of the mechanism to follow the commanded position velocities). In general the control strategies used in robot involves position coordination in Cartesian space by direct or indirect kinematics method. Inverse kinematics comprises the computation need to find the join angles for a given Cartesian position and orientation of the end effectors. This computation is fundamental to control of robot arms but it is very difficult to calculate an inverse kinematics solution of robot manipulator. For this solution most industrial robot arms are designed by using a non-linear algebraic computation to finding the inverse kinematics solution. From the literature it is well described that there is no unique solution for the inverse kinematics. That is why it is significant to apply an artificial neural network models. Here structured artificial neural network (ANN) models an approach has been proposed to control the motion of robot manipulator. In these work two types of ANN models were used. The first kind ANN model is MLP (multi-layer perceptrons) which was famous as back propagation neural network model. In this network gradient descent type of learning rules are applied. The second kind of ANN model is PPN (polynomial poly-processor neural network) where polynomial equation was used. Here, work has been undertaken to find the best ANN configuration for the problem. It was found that between MLP and PPN, MLP gives better result as compared to PPN by considering average percentage error, as the performance index.

CHAPTER

1

INTRODUCTION

Robot manipulator is composed of a serial chain of rigid links connected to each other by revolute or prismatic joints. A revolute joint rotates about a motion axis and a prismatic joint slide along a motion axis. Each robot joint location is usually defined relative to neighboring joint. The relation between successive joints is described by $4X4$ homogeneous transformation matrices that have orientation and position data of robots. The number of those transformation matrices determines the degrees of freedom of robots. The product of these transformation matrices produces final orientation and position data of an *n* degrees of freedom robot manipulator. Robot control actions are executed in the joint coordinates while robot motions are specified in the Cartesian coordinates. Conversion of the position and orientation of a robot manipulator end-effectors from Cartesian space to joint space, called as inverse kinematics problem, which is of fundamental importance in calculating desired joint angles for robot manipulator design and control. In most robotic applications the desired positions and orientations of the end effectors are specified by the user in Cartesian coordinates. The corresponding joint values must be computed at high speed by the inverse kinematics transformation.

For a manipulator with *n* degree of freedom, at any instant of time joint variables is denoted by $\theta_i = \theta(t), i = 1,2,3.........n$ and position variables $x_j = x(t), j = 1,2,3.......m..$ The relations between the end-effectors position *x* (*t*) and joint angle $\theta(t)$ can be represented by forward kinematic equation,

$$x(t) = f(\theta(t)) \tag{1}$$

Where f is a nonlinear, continuous and differentiable function. On the other hand, with the given desired end effectors position, the problem of finding the values of the joint variables is inverse kinematics, which can be solved by,

$$\theta(t) = f(x(t)) \qquad (2)$$

Solution of (2) is not unique due to nonlinear, uncertain and time varying nature of the governing equations. Figure 1 shows the schematic representation of forward and inverse kinematics. The different techniques used for solving inverse kinematics can be classified as algebraic [1], geometric [2] and iterative [3]. The algebraic methods do not guarantee closed form solutions. In case of geometric methods, closed form solutions for the first three joints of the manipulator must exist geometrically. The iterative methods converge to only a single solution depending on the starting point and will not work near singularities.

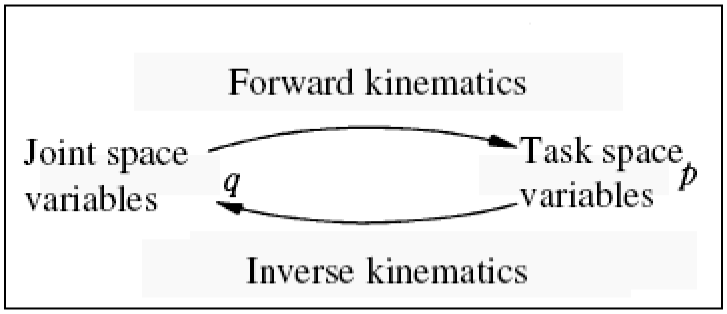


Fig. (1.1) schematic representation of forward and inverse kinematics**.**

If the joints of the manipulator are more complex, the inverse kinematics solution by using these traditional methods is a time consuming. In other words, for a more generalized *m* degrees of freedom manipulator, traditional methods will become prohibitive due to the high complexity of mathematical structure of the formulation. To compound the problem further, robots have to work in the real world that cannot be modeled concisely using mathematical expressions. In recent years, there have been increasing research interest of artificial neural networks and many efforts have been made on applications of neural networks to various control problems. The most significant features of neural networks are the extreme flexibility due to the learning ability and the capability of nonlinear functions approximations. This fact leads us to expect neural networks to be a excellent tool for solving the inverse kinematics problem in robot manipulators with overcoming the difficulties of algebraic, geometric and iterative methods.

The robot motion problem involves in bringing the end-effectors of the manipulator from the present to the desired position and orientation in the global coordinates while following a prescribed trajectory in either the joint coordinates or global coordinates. Since the desired position is usually specified in the global coordinates, whereas the actuators used to drive the system are to be commanded with desired joint values, the inverse kinematics must be solved. The solution of the inverse kinematic problem maps the six-degree of freedom world coordinates of the robot manipulator’s end-effectors into the robot’s joint space. The number of solutions will depend on the manipulator configuration (including the number, type, and relative location of the joints), on the range of motion of each of the joints, and on the location of the selected end-effectors position in world coordinates. There may be no solutions, a unique solution, or multiple solutions. Kinematicians have for some time worked on this problem and have, as of yet, not developed a methodology that can solve the inverse kinematics problem for a generalized N-degree-of-motion freedom manipulator. Solutions have been found, however, for certain manipulator configurations. Industrial robot manipulators have conformed to these configurations in order to facilitate the specification of tasks in a Cartesian-defined Workspace (often called the task space).

An alternative solution to that of developing and solving a set of equations would be useful for those cases where:

A. The equations cannot be derived even though the manipulator can be designed And built,

B. The equations are so computationally intensive that solutions take too long to Compute for practical robot control implementation, and

C. The equations involve coefficients which cannot readily be determined.

Humans and animals control their extremities without recourse to solving a set of equations. There ought to be a methodology that more closely mimics the mechanisms whereby we move our arms without consciously determining (i.e., calculating) the necessary joint angles, and velocities that enable that motion. An acceptable solution must recognize the existence and location of singularities and find a valid solution such that the continuity of the mapping between world and joint space is preserved. Possible solutions to this problem include both numerical procedures and neural network based methods. The success of numerical solution procedures depends to a great extent on the formulation of a mathematical expression that accurately describes the functional relationship between the input parameters (specified end-point position of the manipulator in world coordinates, in our case) and the output solution parameters (the joint angles in our case). There are similarities between traditional numerical solution procedures and neural net methods. These include the existence of an iterative adaptation procedure and a performance measure. However, we wish to limit ourselves to neural net procedures in which the solution is not determined based on a mathematical expression defining the input/output relationship, but is captured in some form of an associative memory relati
Several neural network approaches have been proposed in the literature. Guez and Ahmad [3], applying the back-error propagation algorithm based on a three layer perceptron, solved the problem as a learning process. Their first approach "yielded good results but were not accurate enough to be practically utilized." A second attempt by these authors combined back-error propagation with a conventional numerical procedure. They used the neural network simply as a "lookup table in providing a good initial guess to an iterative procedure." In general, it should be noted that back propagation requires
the development of "hidden units," which will slow down the learning process. Guo and Cherkassky [7] proposed a solution using a Hopfield net. Their solution

did not directly develop the inverse kinematic relationship, but instead coupled the neural net with a Jacobian based control technique.

We used MLP (multiple layer perceptrons) and PPN (polynomial poly-processor neural network) method and comparison with MIMO system which uses a Widrow- Hoff type error correction rule. This unsupervised method learns the functional relationship between input (Cartesian) space and output (joint) space based on a localized adaptation of the mapping, by using the manipulator itself under joint control and adapting the solution based on a comparison between the resulting locations of the manipulator's end effectors in Cartesian space with the desired location. Even when a manipulator is not available; the approach is still valid if the forward kinematic equations are used as a model of the manipulator. The forward kinematic equations always have a unique solution, and the resulting Neural net can be used as a starting point for further refinement when the manipulator does become available. Artificial neural network especially MLP and PPN are used to learn the forward and the inverse kinematic equations of two degrees freedom (DOF) robot arm. The technique is independent of arm configuration, including the number of degrees of freedom and the link geometry. In this paper two types of artificial neural networks were used and finalized which one is giving better result. The comparative study and results presented in this paper indicate the feasibility of using these ANN for learning complex input/output relations of robot kinematic control (based on computation of forward and inverse mapping between joint space and Cartesian space). The simulation shows that the MLP and PPN with MIMO system algorithms assure faster convergence compared to other algebraic and analytical algorithm.

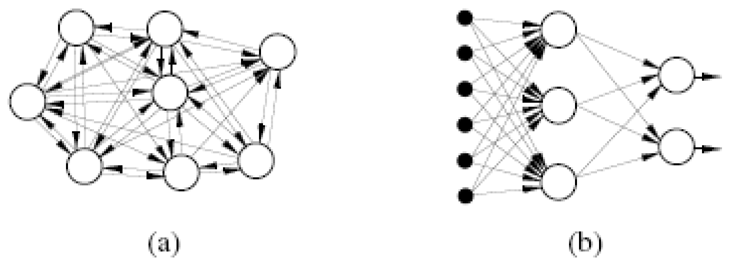


Fig.(1.2). (a) feed-back neural network and, (b) feed-forward neural network

These are some basic neural networks which applied here in this paper as you can see above fig.(2),there are feed-back and fee-forward neural network is shown. Typical network structures include feed-back and feed-forward NNs. Learning algorithms are categorized into supervised learning and unsupervised learning. This section provides an overview of these models and algorithms. In a class of neural networks (NN) called Feed-forward Networks the processing elements, termed as nodes indicated by circles in Fig. 2, are connected in layers through links, termed as weights indicated by arrows in Fig. 2. The output of the node is a function of the inputs, which are weighted outputs of the nodes of the previous layer, and the threshold of the node. The learning takes place through the modification of the weights and the thresholds as specified by the training algorithm that acts on the supplied input and output data pairs as the training set. The training algorithm used in our simulations is the Back Error Propagation (BEP) Algorithm. The nodes to which the input is applied are called as the input nodes and the nodes from which the output is taken are called as the output nodes. The remaining nodes are termed as hidden nodes.

A simple multilayer perceptron neural network (MLP) with back propagation learning was used in the first step. The input layer has as many nodes as the number of inputs to the map, namely four actuator lengths. Similarly the output layer will have two nodes which represent the orientation of the moving plate (θ_1, θ_2). The number of neurons in the hidden layer was used as a design parameter. Sigmoid and linear transfer functions were selected for all hidden and output layer nodes respectively.

1.1 **Background of the work**

In this paper, some methods of artificial neural network applied for the solution of inverse kinematics of 2-link serial chain manipulator. The methods are multilayer perceptrons and polynomial preprocessor neural network has applied. The main objective of this thesis is to predict the values of joint angles (inverse kinematics), as we know that there is no unique solution for the inverse kinematics even mathematical formulae are complex and time taking so it is better to find out solution through neural network. There are so many methods in soft-computing, but in this paper two methods has been taken. After validation of these methods, we multilayer perceptrons giving better result.

1.2 **Objective of the thesis**

The main objective of the thesis is to find out the solution for inverse kinematics of manipulator as well as comparison of neural network methods. Validation of the NN methods ensures future selection of the correct method of NN. From the literature it is well described that there is no unique solution for inverse kinematics. This is why it is significant to apply artificial neural networks models. Here work has been undertaken to find the best ANN configuration for the problem.

1.3 **Methodology**

In this paper the researchers has proposed two methods for the solution of inverse kinematics of manipulator, the proposed methods are multilayer perceptrons and polynomial preprocessor in order to validate the performance of MLP and PPN for inverse kinematics problem, simulation studies are carried out by using MATLAB. Many researchers have followed MLP, PPN, RBF and FLANN with MISO (multi input single output) system. Here in this paper we have applied MLP and PPN with MIMO (multi input multi output) system. A set of 130 data sets were first generated as per the formula equation (10) for this the input parameter X and Y coordinates in inches. Using these data sets was basis for the training and evaluation or testing the MLP and PPN models. Out of the sets of 130 data points, 100 were used as training data and 30 were used for testing for MLP. Back-propagation algorithm was used for training the network and for updating the desired weights. In this work epoch based training method was applied.

1.4 **Scope of the Present Work**

In this study the MLP and PPN has been proposed for the solution of inverse kinematics problem of robot manipulator. However, it has some limitations. There are several types of soft computing methods are available which can be used for finding the solution, but this is beyond the scope of this thesis but this technique can be used for the future scope of the thesis. These methods are followed:

- Application of fuzzy inference system (FIS)
- Adaptive network based fuzzy inference system (ANFIS)
- Functional link artificial neural network (FLANN)
- Evaluation computation

1.5 **Organization Of The Thesis**

Robot control actions are executed in the joint coordinates while robot motions are specified in the Cartesian coordinates. Conversion of the position and orientation of a robot manipulator end-effectors from Cartesian space to joint space, called as inverse kinematics problem. In chapter [2] various researchers has proposed neural network models for the prediction of inverse kinematics, methods they applied are feed forward architectures, multilayer perceptrons using back propagation algorithms, polynomial preprocessor networks , functional link artificial neural network and radial basis functional network. In chapter [3] we focused on fundamentals of inverse kinematics and direct kinematics of manipulator. In chapter [4] we have discussed about artificial neural network and various neural models and explained why neural networks are important for inverse kinematics. In chapter [5] the researchers has proposed two methods for the solution of inverse kinematics of manipulator, their proposed methods are multilayer perceptrons and polynomial preprocessor in order to validate the performance of MLP and PPN for inverse kinematics problem. In chapter [6] simulation studies are carried out by using MATLAB. In chapter [7] conclusion and future work were discussed.

1.6 **Summary**

The robot motion problem involves in bringing the end-effectors of the manipulator from the present to the desired position and orientation in the global coordinates while following a prescribed trajectory in either the joint coordinates or global coordinates. Since the desired position is usually specified in the global coordinates, whereas the actuators used to drive the system are to be commanded with desired joint values, the inverse kinematics must be solved. There are several types of soft computing methods available which can be used for finding the solution of the inverse kinematics and further we’ll discuss about them in next chapter.

CHAPTER

2

LITERATURE REVIEW

2.1 Introduction:

Another important consideration is the choice of appropriate criteria for kinematics of robotics and artificial neural networks. Although the ultimate objective of this problem is to find out inverse kinematics, but as we know that there is no unique solution for this problem so we tried to find out IK through neural networks. Some researchers are developing methodologies which can approach to finding this problem.

2.2 Previous Work:

Alavandar and Nigam [1] developed Neuro-Fuzzy based Approach for Inverse Kinematics Solution of Industrial Robot Manipulators. Obtaining the joint variables that result in a desired position of the robot end-effectors called as inverse kinematics is one of the most important problems in robot kinematics and control. In this paper, using the ability of ANFIS (Adaptive Neuro-Fuzzy Inference System) to learn from training data, it is possible to create ANFIS, an implementation of a representative fuzzy inference system using a BP neural network-like structure, with limited mathematical representation of the system. Computer simulations conducted on 2 DOF and 3DOF robot manipulator shows the effectiveness of the approach.

Morris and Mansor [2] developed artificial neural network for finding inverse kinematics of robot manipulator using look up table. The neural networks utilized were multi-layered perceptions with a back-propagation training algorithm. They used 5 hidden layer neurons , the rate of training , ή , was 2.018 , and the momentum factor , α , was 0.54 . The training of the 9 patterns was done 3000 times. The average percentage error, i.e. the percentage of the difference between actual and targeted / desired output, at the final iteration of the final session was 0.02% for the first joint and 0.3% for the second joint.

Guez and Ahmad [3] developed solution to inverse kinematics problem in robots using neural network they employ a neural network model in the solution of the inverse kinematics problem in Robotics. It is found that the neural network can be trained to generate a fairly accurate solution which when augmented with local differential inverse kinematic methods will result in minimal burden on processing load of each control cycle and thus enable real time robot control. The back propagation algorithm simulating a three layer perceptron was employed to tackle this problem. Symmetric sigmoidal nonlinearity was used. The learning rate and the momentum term assumed the values of 0.1 and 0.4 respectively, throughout the different Simulations described below. Also the desired outputs were normalized between -0.9 and +0.9. The average error is less than 0.01 radians while maximum error is 0.25 radians.

Karlik & Aydinb developed [4**]** an improved approach to the solution of inverse kinematics problems for robot manipulator. A structured artificial neural-network (ANN) approach has been proposed here to control the motion of a robot manipulator. Many neural-network models use threshold units with sigmoid transfer functions and gradient descent-type learning rules. The learning equations used are those of the back propagation algorithm. In this work, the solution of the kinematics of a six- degrees-of-freedom robot manipulator is implemented by using ANN. An appropriate computer program has been developed in the Borland C++ language for the ANN architectures considered in this study. Iterations were performed on a PC P-90 computer, and 6000 iterations were used for teaching the ANN.

Npyen., et.al [5] Neural Network Architectures For The Forward Kinematics Problem in Robotics In this paper, various neural network models are considered for solving the robot forward kinematics problem. It is found that certain models with proper training strategies can generate a fairly accurate solution for the robot forward kinematics problem. In this paper, various neural network architectures are used to solve the forward kinematics problem in robotics. Its purpose is to identify the advantages and disadvantages of each architecture in this robotic application. All the results have been obtained by simulation on a SUN 3/60 work station. The networks were trained with a set of sixty four desired inputs/outputs collected from measurements. All the weights of the networks were randomly initialized from -0.5 to *+0.5*. In the output layer of this network, all the weights were initialized to zero. This is to avoid the case in which a local minimum predominates when the training process is started. The training process was stopped when the average error was under 10%.

Jaein, et.al [6] developed Robot Control Using Neural Network. A neural network theory is applied to theoretical robot kinematics to learn accuracy transforms**.** The network is trained on accuracy data that characterize the actual robot kinematics. The network learns the differences in the joint angles to improve the accuracy between the effectors endpoint resulting from the theoretically calculated joint angles and the desired endpoint. It is hoped that the capabilities of modem day neural networks will solve problems that appear to be beyond the bounds of conventional computational devices. The results were virtually identical for both test cases. After the network had be trained on 1 point, the accuracy of positioning the end-effectors to a desired point was improved by an average of 60% and for all test sets presented, the accuracy was greater than the accuracy of the uncompensated or naked controller.

Guo & Cherkassky [7] developed A Solution to the Inverse Kinematic Problem in Robotics Using Neural Network Processing. In this paper, a solution algorithm is presented using the Hopfield and Tank [1985] analog neural computation scheme to implement the Jacobian control technique. 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. At each sampling time, connection weights and neuron states are updated according to current joint positions. During each sampling period, the energy function is minimized and joint velocity command signals are obtained as the states of the Hopfield network. They proposed a solution to the inverse kinematic problem using Hopfield neural network. Our approach is based on the Jacobian control technique. The Hopfield network should be capable of updating its connection weights in real time. The outputs of the network are joint velocity commands which can be used to control joint actuators of a robot manipulator.

Wang and Zilouchian[8] has given solutions of Kinematics of Robot Manipulators Using a kohonen Self- Organizing Neural Network. Kohonen self-organizing neural network is used to solve the forward kinematics problems of robot manipulators. Through competition learning, neurons learn their distribution in the training phase. In sequel, the nonlinear mapping has been obtained by proper calibration of training results. The proposed method is based on the unsupervised learning which does not rely on the knowledge of process model and target information. Simulation results have shown the effectiveness of the proposed method for a two degree planer robot manipulator. The initial weights of neuron are randomly selected to congregate within the estimated range of neuron space. After 1000 end effectors positions have been presented for training, the weight vectors have been moving out of the center range. With 5000 training steps, the neurons begin to approach the shape of its distribution. After 20,000 steps of training, the neurons develop to a proper distribution which is very close to its final form. Fig. 6 shows the average error versus the number of training steps. Obviously, the training result is fare satisfactory after 6.000 iterations.

Xia and Wang [9] developed A Dual Neural Network for Kinematic Control of Redundant Robot Manipulators the inverse kinematics problem in robotics can be formulated as a time-varying quadratic optimization problem. A new recurrent neural network, called the dual network, is presented in this paper. The proposed neural network is composed of a single layer of neurons, and the number of neurons is equal to the dimensionality of the workspace. The proposed dual network is proven to be globally exponentially stable. The proposed dual network is also shown to be capable of asymptotic tracking for the motion control of kinematic ally redundant manipulator.

Yee & Lim [10] developed Forward kinematics solution of Stewart platform using neural networks. The Stewart platform's unique structure presents an interesting problem in its forward kinematics (FK) solution. It involves the solving of a series of simultaneous non-linear equations and, usually, non-unique, multiple sets of solutions are obtained from one set of data. In addition, most effort usually results in having to find the solution of a 16th-order polynomial by means of numerical methods. A simple feed-forward network was trained to recognize the Relationship between the input values and the output values of the FK problem and was able to provide the solution around an average error of 1.0" and 1.0 mm. By performing a few iterations with an innovative offset adjustment, the performance of the trained network was improved tremendously. Two extra iterations with the offset adjustment reduced the average error of the same trained neural network to 0.017" and 0.017 mm.

Gallaf [11] developed Neural Networks for Multi-Finger Robot Hand Control. This paper investigates the employment of Artificial Neural Networks (ANN) for a multi-finger robot hand manipulation in which the object motion is defined in task-space with respect to six Cartesian based coordinates. The approach followed here is to let an ANN learn the nonlinear functional relating the entire hand joints positions and displacements to object displacement. This is done by considering the inverse hand Jacobian, in addition to the interaction between hand fingers and the object being grasped and manipulated. The developed network has been trained for several object training patterns and postures within a Cartesian based palm dimension. The paper demonstrates the proposed algorithm for a four fingered robot hand, where inverse hand Jacobian plays an important role in robot hand dynamic control.

Houvinen and Handroos [12] they used basically ADAMS –Model for training of neural network for inverse kinematics of flexible robot manipulator. If flexibility of system is included than problem becomes more complicated. Neural network can be used to solve inverse kinematic problem. Multiple layer networks are capable of approximating any function with a finite number of discontinuities. For learning the inverse kinematics neural network needs information about join coordinates, joint angles, and actuator position. By creating flexible ADAMS-model of the robot and equipped with virtual instrument it is possible to simulate the data needed for the training of neural network. in this study the number of training vectors was used 1750 and 350 separate vectors were used for the testing the neural network. In this paper they used of simulation data in training neural network for inverse kinematic computation of manipulator. The result shows that the positioning of a flexible robot using an inverse neural network model is possible but the accuracy is not yet good enough. The accuracy is increased by increasing the number of training vector and training the neural network again.

Daniel Patiño, et.al [13] Neural Networks for Advanced Control of Robot Manipulators. This paper presents an approach and a systematic design methodology to adaptive motion control based on neural networks (NNs) for high-performance robot manipulators, for which stability conditions and performance evaluation are given. The neuro-controller includes a linear combination of a set of off-line trained NNs (bank of fixed neural networks), and an update law of the linear combination coefficients to adjust robot dynamics and payload uncertain parameters. This paper deals with a neural network-based controller for motion dynamic control of robot manipulators. The dynamical behavior of a rigid manipulator can be characterized by a system of highly coupled and nonlinear differential equations. The nonlinear effects are emphasized for robots working at high speeds with direct drive motors or low ratio gear transmissions. A simulation study has been carried out for the PUMA-560 robot. They have presented an approach and a systematic design methodology to a motion adaptive control based on NNs for high-performance robot manipulators, for which stability conditions and performance evaluation have been given.

Ted Hesselroth,et. al [14] they proposed Neural Network Control of a Pneumatic Robot Arm. A neural map algorithm has been employed to control a five-joint pneumatic robot arm and gripper through feedback from two video cameras. The pneumatically driven robot arm (Soft Arm) employed in this investigation shares essential mechanical characteristics with skeletal muscle systems. To control the position of the arm, 200 neurons formed a network representing the three-dimensional workspace embedded in a four-dimensional system of coordinates from the two cameras, and learned a three-dimensional set of pressures corresponding to the end effectors positions, as well as a set of 3×4 Jacobian matrices for interpolating between these positions. The gripper orientation was achieved through adaptation of a 1×4 Jacobian matrix for a fourth joint. Because of the properties of the rubber-tube actuators of the Soft Arm, the position as a function of supplied pressure is nonlinear, no separable, and exhibits hysteresis. Nevertheless, through the neural network learning algorithm the position could be controlled to an accuracy of about one pixel (_3 mm) after two hundred learning steps and the orientation could be controlled to two pixels after eight hundred learning steps. This was achieved through employment of a linear correction algorithm using the Jacobian matrices mentioned above. Applications of repeated corrections in each positioning and grasping step leads to a very robust control algorithm since the Jacobians learned by the network have to satisfy the weak requirement that the Jacobian yields a reduction of the distance between gripper and target.

Benhabib,et.al.[15] A solution to the inverse kinematics is a set of joint coordinates which correspond to a given set of task space coordinates (position and orientation of end effectors). For the class of kinematic ally redundant robots the solution is generically no unique such that special methods are required for obtaining a solution. The paper presents a new algorithm for solving the inverse kinematics which is based on a modified Newton-Raphson iterative technique. The new algorithm is efficient, converges rapidly, and completely generalizes the solution of the inverse kinematics problem for redundant robots. The method is illustrated by a numerical example.

Oyama and Tachi [16] Inverse kinematics computation using an artificial neural network that learns the inverse kinematics of a robot arm has been employed by many researchers. However, conventional learning methodologies do not pay enough attention to the discontinuity of the inverse kinematics system of typical robot arms with joint limits. The inverse kinematics system of the robot arms is a multi-valued and discontinuous function. Since it is difficult for a well-known multi-layer neural network to approximate such a function, a correct inverse kinematics model for the end-effectors's overall position and orientation cannot be obtained by using a single neural network. In order to overcome the discontinuity of the inverse kinematics function, we propose a novel modular neural network system for the inverse kinematics model learning. We also propose the on-line learning and control method for trajectory tracking.

Manocha and Canny [17] in this paper, they present an algorithm and implementation for efficient inverse kinematics for a general *6R* manipulator. When stated mathematically, the problem reduces to solving a system of multivariate equations. They make use of the algebraic properties of the system and the symbolic formulation used for reducing the problem to solving a
univariate polynomial. However, the polynomial is expressed as a matrix determinant and its roots are computed by reducing to an eigenvalue problem. The other roots of the multivariate system are obtained by computing eigenvectors and substitution. The algorithm involves symbolic preprocessing, matrix computations and a variety of other numerical techniques. The average running time of the algorithm, for most cases, is 11 milliseconds on an IBM RS/6000 workstation. This approach is applicable to inverse kinematics of all serial manipulators.

Kieffer et.al.[18], presented a methodology where a neural network is used to learn the inverse kinematic relationship for a robot arm. They presented two link, two degree of freedom planar robot arm simulation, and an accompanying neural network which solves the inverse kinematic problem. Their method is based on Kohonen’s self organizing mapping algorithm using a Widrow-Hoff type error correction rule. They have specifically addressed a number of issues associated with the inverse kinematic solution, including the occurrence of singularities and multiple solutions.

Kozalziewicz et.al,[19] developed the solution of inverse kinematics of robot manipulator with the help of Partitioned Neural Network architecture. In this paper they obtained quit good result, and they demand high accuracy. The Partitioned Neural Network is composed of a Pre - Processing layer and Partition Modules containing dedicated neurons. The learning equations used are those of the Back propagation algorithm. The Network has been applied to learning of the Inverse Kinematic solution of a 6 degree of freedom robot manipulator. After training, the Partitioned network was able to predict robot joint angles.

2.3 Summary:

The different types of approaches to the inverse kinematics have been reported. Here these approaches show their various advantages and disadvantages to the development of new design problem. Taking the old approach in to consideration the development of new approaches conceptualized through these literatures.

CHAPTER

3

INTRODUCTION TO KINEMATICS OF MANIPULATOR

3.1 Introduction

The purpose of this chapter is to introduce you to robot kinematics, and the concepts related to both open and closed kinematics chains. Forward kinematics is distinguished from inverse kinematics. Kinematics is the study of motion without regard to the forces that create it. The forward kinematics is about finding an end effectors or tool piece pose given a set of joint variables. The Inverse Kinematics is the opposite problem. We want to and a set of joint variables that give rise to a particular end effectors or tool piece pose. Kinematics is the study of motion. In this subsection, we will explore the relationship between joint movements and end effectors movements. More precisely, we will try to develop equations that will make explicit the dependence of end effectors coordinates on joint coordinates and vice versa.

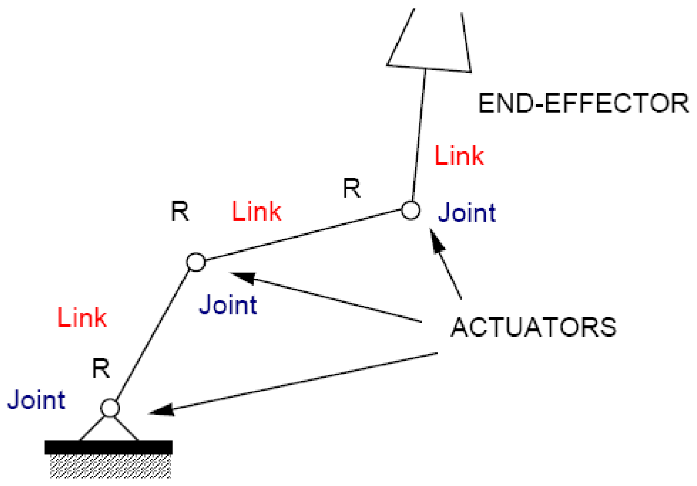


Fig. (3.1) A schematic of a planar manipulator with three revolute joints**.**

We will start with the example of the planar 3*R* manipulator. From basic trigonometry, the position and orientation of the end effectors can be written in terms of the joint coordinates in the following way:

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \qquad \ldots\ldots\ldots (3)$$

$$\phi = \theta_1 + \theta_2 + \theta_3$$

Note that all the angles have been measured counter clockwise and the link lengths are assumed
to be positive going from one joint axis to the immediately distal joint axis. Equation (3) is a set of three nonlinear equations that describe the relationship between end effectors coordinates and joint coordinates. Notice that we have explicit equations for the end effectors coordinates in terms of joint coordinates. However, to find the joint coordinates for a given set of end effectors coordinates (x, y, φ), one needs to solve the nonlinear equations for θ1, θ2, and θ3.
The kinematics of the planar *R-P* manipulator is easier to formulate. The equations are:

$$x = d_2.\cos\theta_1$$

$$y = d_2.\sin\theta_1 \qquad \ldots\ldots\ldots\ldots\ldots (4)$$

$$\phi = \theta_1$$

Again the end-effector coordinates are explicitly given in terms of the joint coordinates. However, since the equations are simpler (than in (3)), you would expect the algebra involved in
Solving for the joint coordinates in terms of the end effector coordinates to be easier. Notice that
in contrast to (3), now there are three equations in only two joint coordinates, θ_1, and d_2. Thus, in general, we cannot solve for the joint coordinates for an arbitrary set of end effector coordinates. Said another way, the robot cannot, by moving its two joints, reach an arbitrary end effector position and orientation.

Let us instead consider only the position of the end effector described by (*x*, *y*), the coordinates of the end effector tool point or reference point. We have only two equations:

$$x = d_2.\cos\theta_1$$

$$y = d_2.\sin\theta_1 \qquad \text{...........} \ (5)$$

Given the end effectors coordinates (*x*, *y*), the joint variables can be computed to be:

$$d_2 = +\sqrt{x^2 + y^2}$$

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) \qquad \text{.........} \ (6)$$

Notice that we restricted *d*2 to positive values. A negative *d*2 may be physically achieved by allowing the end effector reference point to pass through the origin of the *x*-*y* coordinate system
over to another quadrant. In this case, we obtain another solution:

$$d_2 = -\sqrt{x^2 + y^2}$$

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) \qquad \text{.......} \ (7)$$

In both cases (6-7), the inverse tangent function is multivalued10. In particular,

$$\tan(x) = \tan(x + k\pi), k = ...-2,-1,0,1,2.... \qquad \text{.......}(8)$$

However, if we limit θ1 to the range 0<θ1<2π, there is a unique value of θ1 that is consistent with the given (*x*, *y*) and the computed *d*2 (for which there are two choices). The existence of multiple solutions is typical when we solve nonlinear equations. As we will see later, this poses some interesting questions when we consider the control of robot manipulators. The planar Cartesian manipulator is trivial to analyze. The equations for kinematic analysis are:

$$x = d_2, y = d_1 \qquad \text{.... (9)}$$

The simplicity of the kinematic equations makes the conversion from joint to end effector coordinates and back trivial. This is the reason why *P-P* chains are so popular in such automation equipment as robots, overhead cranes, and milling machines.

3.2 Direct kinematics

As seen earlier, there are two types of coordinates that are useful for describing the configuration of the system. If we focus our attention on the task and the end effector, we would prefer to use Cartesian coordinates or end effector coordinates. The set of all such coordinates is generally referred to as the Cartesian space or end effector *space*. The other set of coordinates is the so called joint coordinates that is useful for describing the configuration of the mechanical lnkage. The set of all such coordinates is generally called the *joint space*.

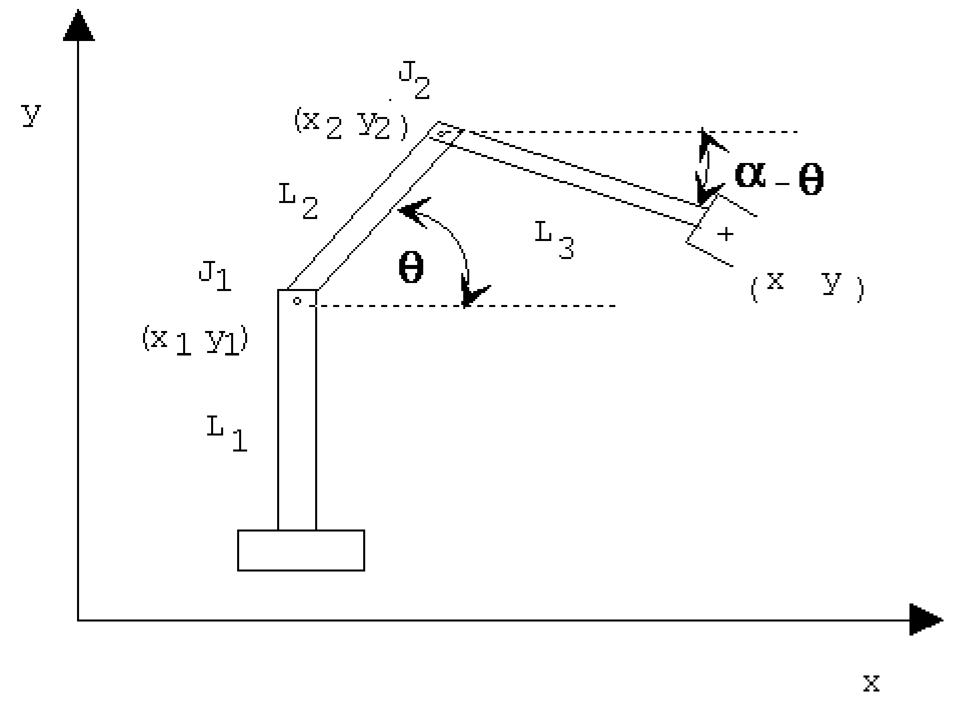


Fig. (3.2) Representation of 3R planar manipulator in Cartesian space.

In robotics, it is often necessary to be able to "map" joint coordinates to end effector coordinates. This map or the procedure used to obtain end effector coordinates from joint coordinates is called direct kinematics. For example, for the 3-*R* manipulator, the procedure reduces to simply substituting the values for the joint angles in the equation

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \qquad \ldots\ldots (10)$$

$$\phi = \theta_1 + \theta_2 + \theta_3$$

And determining the Cartesian coordinates, *x*, *y*, and φ. For the other examples of open chains Discussed so far (*R*-*P*, *P*-*P*) the process is even simpler (since the equations are simpler). In fact, for all serial chains (spatial chains included), the direct kinematics procedure is fairly straight Forward. On the other hand, the same procedure becomes more complicated if the mechanism contains one or more closed loops. In addition, the direct kinematics may yield more than one solution or no solution in such cases. For example, in the planar parallel manipulator in Figure 3, the joint positions or coordinates are the lengths of the three telescoping links ($q1$, $q2$, $q3$) and the end effectors coordinates (x, y, φ) are the position and orientation of the floating triangle. It can be shown that depending on the value of ($q1$, $q2$, $q3$), the number of (real) solutions for (x, y, φ) can be anywhere from zero to six. For the Stewart Platform in Figure 4, this number has been shown to be anywhere from zero to forty.

3.3 Inverse kinematics

The analysis or procedure that is used to compute the joint coordinates for a given set of end effector coordinates is called inverse kinematics. Basically, this procedure involves solving a set of equations. However the equations are, in general, nonlinear and complex, and therefore, the inverse kinematics analysis can become quite involved. Also, as mentioned earlier, even if it is possible to solve the nonlinear equations, uniqueness is not guaranteed. There may not (and in General, will not) be a unique12 set of joint coordinates for the given end effector coordinates.

We saw that for the *R-P* manipulator, the direct kinematics equations are:

$$\begin{aligned} x &= d_2.\cos\theta_1 \\ y &= d_2.\sin\theta_1 \end{aligned} \qquad \text{.....(11)}$$

If we restrict the revolute joint to have a joint angle in the interval [0, 2π), there are two solutions for the inverse kinematics:

$$\begin{aligned} d_2 &= \sigma\sqrt{x^2+y^2} \\ \theta_1 &= a\tan 2\left(\frac{y}{d_2},\frac{x}{d_2}\right) \qquad \text{....(12)} \\ \sigma &= \pm 1 \end{aligned}$$

The inverse kinematics analysis for a planar 3-*R* manipulator appears to be complicated but we can derive analytical solutions. Recall that the direct kinematics equations (10) are:

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \qquad 10(a)$$

$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \qquad 10(b)$$

$$\phi = \theta_1 + \theta_2 + \theta_3 \qquad 10(c)$$

We assume that we are given the Cartesian coordinates, *x*, *y*, and φ and we want to find analytical expressions for the joint angles θ_1, θ_2 and θ_3 in terms of the Cartesian coordinates.

Substituting 10(c) into 10(a) and 10(b) we can eliminate θ_3 so that we have two equations in θ_1

And θ_2 :

$$x - l_3 \ \cos\phi = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \qquad 10(d)$$

$$y - l_3 \sin\phi = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \qquad 10(e)$$

Where the unknowns have been grouped on the right hand side; the left hand side depends only on the end effector or Cartesian coordinates and are therefore known.

Rename the left hand sides, $x' = x - l_3 \cos\phi, y' = y - l_3 \sin\phi$ for convenience. We regroup

Terms in (d) and (e), square both sides in each equation and add them:

$$(x' - l_1 \cos\theta_1)^2 + (y' - l_1 \sin\theta_1)^2 = (l_2 \cos(\theta_1 + \theta_2))^2 + (l_2 \sin(\theta_1 + \theta_2))^2$$

After rearranging the terms we get a single nonlinear equation in $\theta 1$:

$$(-2l_1 x')\cos\theta_1 + (-2l_1 y')\sin\theta_1 + (x'^2 + y'^2 + l_1{}^2 - l^2{}_2) = 0 \qquad 10(f)$$

Notice that we started with three nonlinear equations in three unknowns in (a-c). We reduced the problem to solving two nonlinear equations in two unknowns (d-e). And now we have simplified it further to solving a single nonlinear equation in one unknown (f). Equation (f) is of the type

$$P\cos\alpha + Q\sin\alpha + R = 0 \qquad 10(g)$$

There are two solutions for $\theta 1$ given by:

$$\theta_1 = \gamma + \sigma \cos^{-1}\left[\frac{-(x'^2 + y'^2 + l^2{}_1 - l_2{}^2)}{2l_1\sqrt{x'^2 + y'^2}}\right] \qquad 10(h)$$

Where,

$$\gamma = a\tan 2\left[\frac{-y'}{\sqrt{x'^2 + y'^2}}, \frac{-x'}{\sqrt{x'^2 + y'^2}}\right]$$

And $\sigma = \pm 1$

Note that there are two solutions for $\theta 1$, one corresponding to $\sigma = +1$, the other corresponding to

$\sigma = -1$. Substituting any one of these solutions back into Equations (d) and (e) gives us:

$$\cos(\theta_1 + \theta_2) = \frac{x' - l_1 \cos\theta_1}{l_2}$$

$$\sin(\theta + \theta_2) = \frac{y' - l_1 \sin\theta_1}{l_2}$$

And,

$$\theta_2 = a\tan 2\left[\frac{y' - l_1 \sin\theta_1}{l_2}, \frac{x' - l_1 \sin\theta_1}{l_2}\right] - \theta_1 \qquad 10(i)$$

Thus, for each solution for $\theta 1$, there is one (unique) solution for $\theta 2$.

Finally, $\theta 3$ can be easily determined from (c):

$$\theta 3 = \Phi - \theta 1 - \theta 2 \qquad 10(j)$$

Equations (h-j) are the inverse kinematics solution for the 3-*R* manipulator. For a given end effectors position and orientation, there are two different ways of reaching it, each corresponding to a different value of σ. These different configurations are shown in Figure

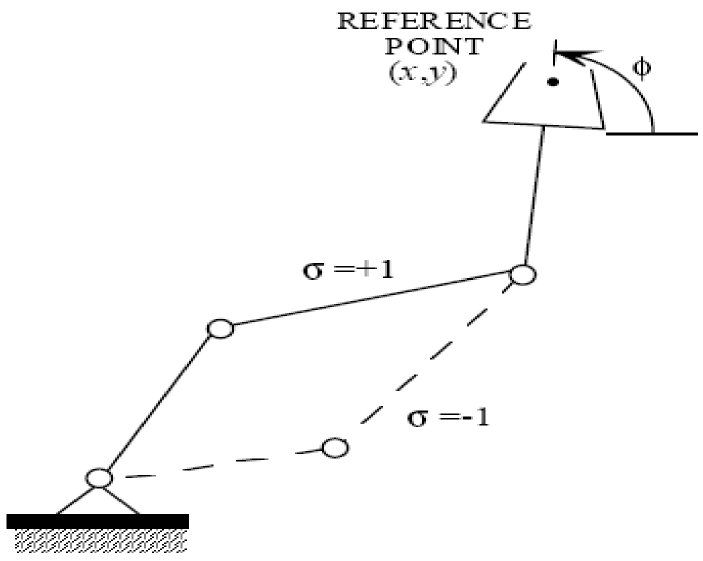


Fig. (3.3) The two inverse kinematics solutions for the 3R manipulator: "elbow-up" configuration (σ=+1) and the "elbow-down" configuration (σ= -1).

3.4 Summary

Robot manipulator is composed of a serial chain of rigid links connected to each other by revolute or prismatic joints. A revolute joint rotates about a motion axis and a prismatic joint slide along a motion axis. Each robot joint location is usually defined relative to neighboring joint. The relation between successive joints is described by 4X4 homogeneous transformation matrices that have orientation and position data of robots. The number of those transformation matrices determines the degrees of freedom of robots. The purpose of this chapter is to introduce you to robot kinematics, and the concepts related to both open and closed kinematics chains. Forward kinematics is distinguished from inverse kinematics. Kinematics is the study of motion without regard to the forces that create it. The forward kinematics is about finding an end effectors or tool piece pose given a set of joint variables. The Inverse Kinematics is the opposite problem. We want to and a set of joint variables that give rise to a particular end effectors or tool piece pose. Kinematics is the study of motion.

CHAPTER 4

INTRODUCTION TO NEURAL NETWORK ARCHITECTURE

Artificial neural network (ANN) takes their name from the network of nerve cells in the brain. Recently, ANN has been found to be an important technique for classification and optimization problem. Artificial Neural Networks (ANN) has emerged as a powerful learning technique to perform complex tasks in highly nonlinear dynamic environments. Some of the prime advantages of using ANN models are their ability to learn based on optimization of an appropriate error function and their excellent performance for approximation of nonlinear function. The ANN is capable of performing nonlinear mapping between the input and output space due to its large parallel interconnection between different layers and the nonlinear processing characteristics. An artificial neuron basically consists of a computing element that performs the weighted sum of the input signal and the connecting weight. The sum is added with the bias or threshold and the resultant signal is then passed through a nonlinear function of sigmoid or hyperbolic tangent type. Each neuron is associated with three parameters whose learning can be adjusted; these are the connecting weights, the bias and the slope of the nonlinear function. For the structural point of view a NN may be single layer or it may be multilayer. In multilayer structure, there is one or many artificial neurons in each layer and for a practical case there may be a number of layers. Each neuron of the one layer is connected to each and every neuron of the next layer. The functional-link ANN is another type of single layer NN. In this type of network the input data is allowed to pass through a functional expansion block where the input data are nonlinearly mapped to more number of points. This is achieved by using trigonometric functions, tensor products or power terms of the input. The output of the functional expansion is then passed through a single neuron. The learning of the NN may be supervised in the presence of the desired signal or it may be unsupervised when the desired signal is not accessible. Here in this paper ANN is supervised learning. Rumelhart developed the Back-propagation (BP) algorithm, which is central to much work on supervised learning in MLP. A feed-forward structure with input, output, hidden layers and nonlinear sigmoid functions are used in this type of network. In recent years many different types of learning algorithm using the incremental back-propagation algorithm, evolutionary learning using the nearest neighbor MLP and a fast learning algorithm based on the layer-by-layer optimization procedure.

4.1 Introduction of Artificial Neural Network:

A neural network is a machine that is designed to model the way in which the brain performs a particular task or function of interest. To achieve good performance, they employ a massive interconnection of simple computing cells referred to as ‘Neurons’ or ‘processing units’. Hence a neural network viewed as an adaptive machine can be defined as A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are us to store the acquired knowledge.

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements. Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. Such a situation is shown below.

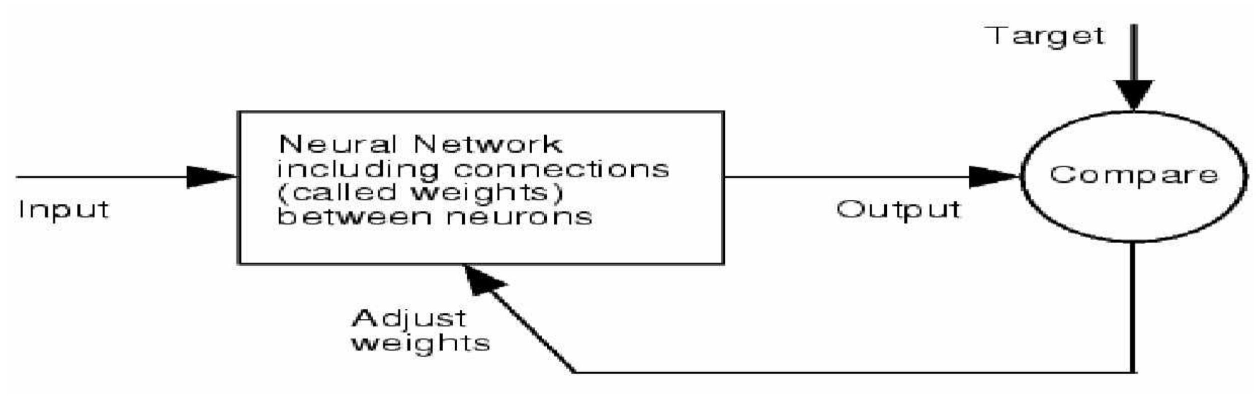

Target
Neural Network including connections (called weights) between neurons
Input
Output
Compare
Adjust weights

Fig 4.1: Neural Network Model

The true power and advantage of neural networks lies in their ability to represent both linear and non-linear relationships and in their ability to learn these relationships directly from the data being modeled. Traditional linear models are simply inadequate when it comes to modeling data that contains non-linear characteristics. Neural networks are designed to work with patterns - they can be classified as pattern classifiers or pattern associates.

4.2 WHY USE NEURAL NETWORKS?

It is apparent that a neural network derives its computing power through, first, its massively parallel distributed structure and, second, its ability to learn and therefore generalize. The use of neural networks offers the following useful properties and capabilities:

- Massive parallelism
- Distributed representation and computation
- Learning ability
- Generalization ability
- Input-output mapping
- Adaptivity
- Uniformity of Analysis and Design
- Fault tolerance
- Inherent contextual information processing

- VLSI implements ability.

4.3 NEURON MODEL

An artificial neuron is a device with many inputs and many outputs. Each input is multiplied by a corresponding weight, analogous to a synaptic strength, and all the weighted inputs are then summed to determine the activation level of the neuron. These weighted inputs are then added together to produce ***'net'*** output and if they exceed a pre-set threshold value, the neuron fires. The ***'net'*** output produced is further processed by an activation function (***f***) to produce the neuron's output signal. A simple neuron model can be represented as below:

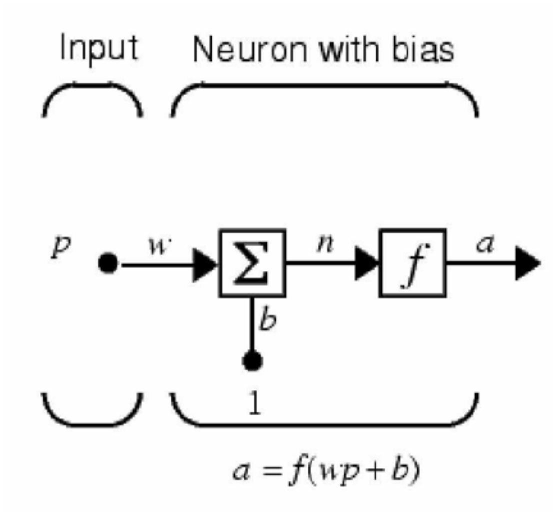


Fig 4.2: Simple Neuron Model

In the above figure p is input of signal w is the weighted input and b is bias input. The block **'å'** produces the **'net'** output by summing the weighted inputs. The block **'f'** represents the activation function.

4.4 NETWORK LAYERS

Although a single neuron can perform certain simple pattern detection functions, the power of neural computation comes from connecting neurons into network layers. These multilayer networks have been proven to have capabilities beyond those of a single layer. These networks are formed by cascading group of single layers; the output of one layer provides the input to the subsequent layer.

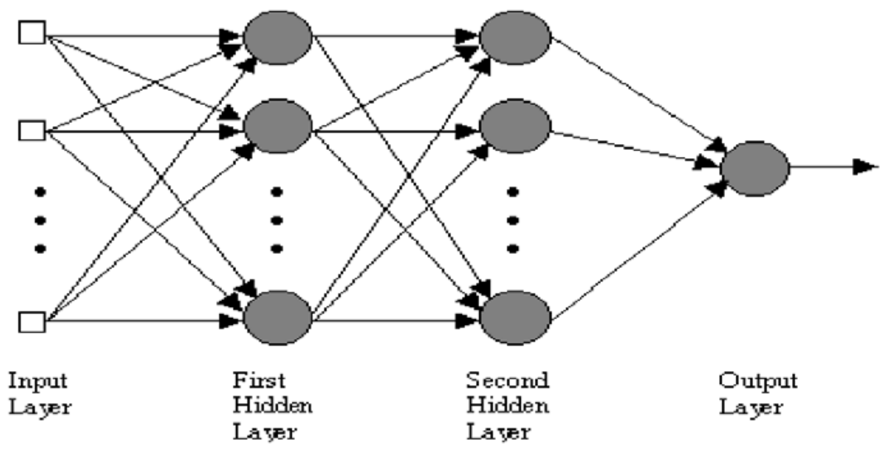


Figure 4.3: A multi-layer neuron model

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units as in the figure:

- The activity of the input units represents the raw information that is fed into the network.

- The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.

- The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

The hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

4.5 ACTIVATION FUNCTIONS

Activation functions for the hidden units are needed to introduce nonlinearity into the network. Without nonlinearity, hidden units would not make nets more powerful as it is the nonlinearity (i.e, the capability to represent nonlinear functions) that makes multilayer networks so powerful.

There are three types of Activation functions:

- Binary Function - PERCEPTRON
- Sigmoidal Function
- Hyperbolic Tangent Function

4.6 SIGMOIDAL FUNCTION

The block **'f'** accepts the NET output and produces the signal labeled OUT. If the 'f' processing block compresses the range of NET, so that OUT never exceeds some limits regardless of the value of NET, 'f' is called a squashing function. The squashing function is often chosen to be the logistic function or "sigmoid". This function is expressed in mathematically as

$$OUT = \frac{1}{1+e^{-net}}$$

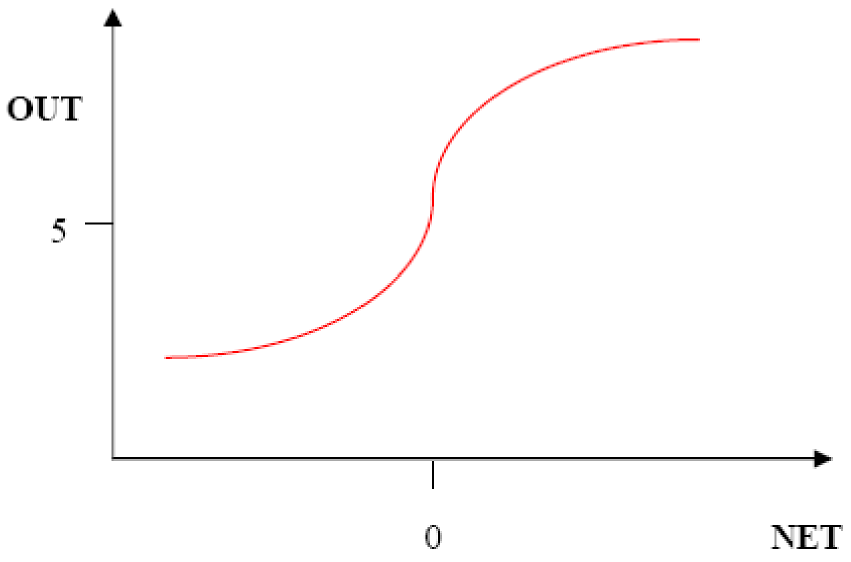


Figure 4.4: Sigmoidal Function

The non-linear gain is calculated by finding the ratio of the change in OUT to a small change in NET. Thus, gain is the slope of the curve (shown in figure) at a specific excitation level. It varies from a low value at large negative excitations, to a high value at zero excitation, and it drops back as excitation becomes very large and positive. Small signals, while its regions of decreasing gain at positive and negative extremes are appropriate for large excitations. In this way, a neuron performs with appropriate gain over a wide range of input levels.

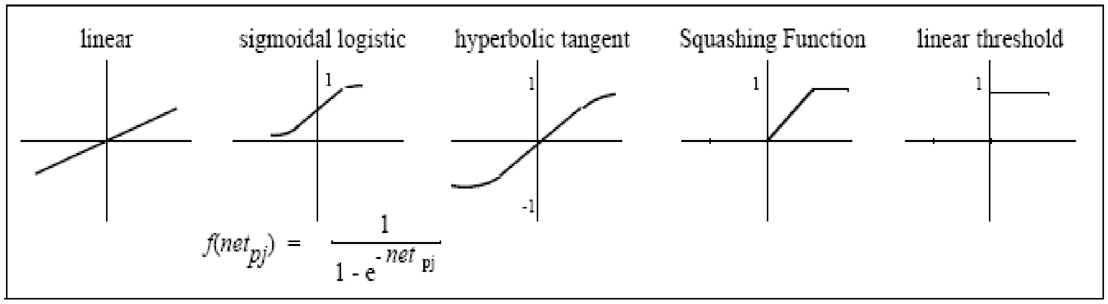


FIG.*(4.5)* Common activation functions

4.7 FEED-FORWARD NEURAL NETWORKS:

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. These networks are called non-recurrent networks and they do not require any memory as outputs are directly related to inputs and weights. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down. The figure (9) below shows a simple feed forward network:

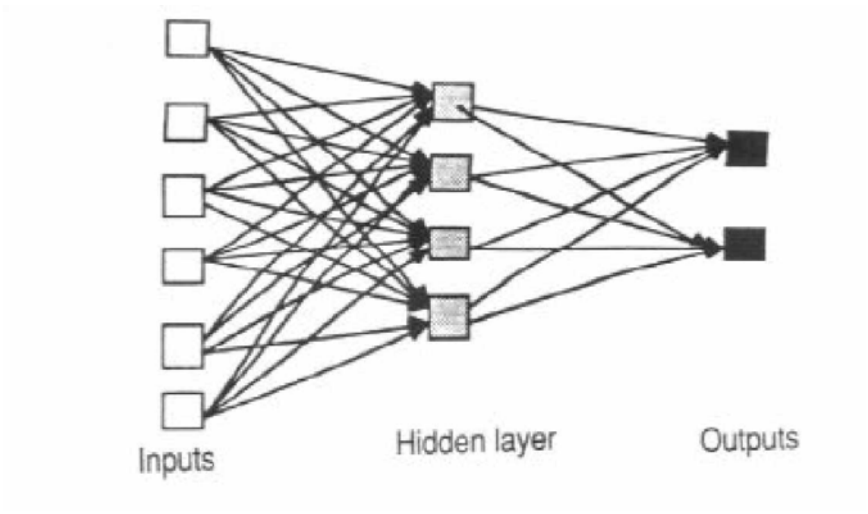


Figure (4.6): An example of Feed Forward Network

4.8 LEARNING

Learning is essential to most of these neural network architectures and hence the choice of a learning algorithm is a central issue in network development. Learning implies that a processing unit is capable of changing its input/output behavior as a result of changes in the environment. Since the activation rule is usually fixed when the network is constructed and since the input/output vector cannot be changed, to change the input/output behavior the weights corresponding to that input vector need to be adjusted. In a neural network, learning can be supervised or unsupervised.

4.9 BACK PROPOGATION ALGORITHM

For many years, there was no theoretically sound algorithm for training multilayer artificial neural networks. The invention of the back propagation algorithm has played a large part in the resurgence of interest in artificial neural networks. Back propagation is a systematic method for training multilayer artificial neural networks (Perceptrons). The following figure shows the basic model of the neuron used in Back propagation networks.

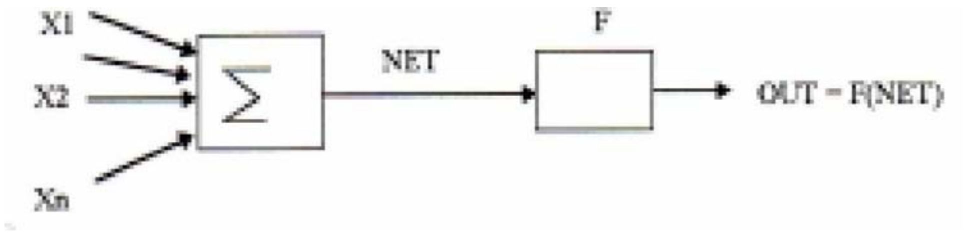


Figure (4.7): Basic model of neuron using back propagation

Each input is multiplied by corresponding weights, analogous to a synaptic strength, and all the weighted inputs are then summed to determine the activation level of the neuron. These summed (NET) signals are further processed by an activation function (F) to produce the neuron's output signal (OUT). In back propagation, the function used for the activation is the logistic function or Sigmoid. This function is expressed mathematically as:

$$F(x) = \frac{1}{1+e^{-x}}\text{, thus } OUT = \frac{1}{1+e^{-net}}$$

The Sigmoid compresses the range of NET so that OUT lies between zero and one. Since the back-propagation uses the derivative of the squashing function, it has to be everywhere differentiable. The Sigmoid has this property and the additional advantage of providing a form of automatic gain control (i.e. if the value of NET is large, the gain is small and if it is small the gain is large).

4.10 AN OVERVIEW OF TRAINING IN BACK PROPOGATION

TRAINING ALGORITHM:

The objective of training the network is to adjust the weights so that the application of a set of inputs (input vectors) produces the desired outputs (output vectors). Training a back propagation network involves each input vector being paired with a target vector representing the desired output; together they are called a training pair. The following figure shows the architecture of the multilayer back propagation neural network.

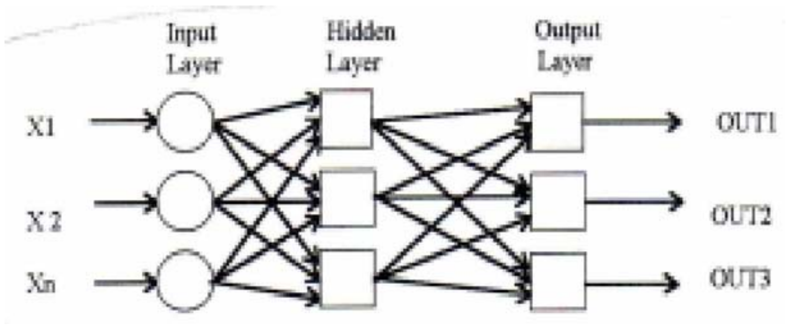


Figure (4.8): Multilayer Back propagation neural network

Before starting the training process, all of the weights are initialized to small random numbers. Training the back propagation network requires the following steps:

1. Select a training pair (next pair) from the training data set and apply the input vector to the network input.

2. Calculate the output of the network, i.e. to each neuron NET=ΣXiWi must be calculated and then the activation function must be applied on the result F (NET).

3. Calculate the error between the network output and the desired output (TARGET – OUT).

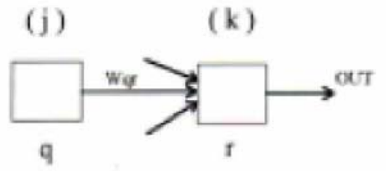
4. Adjust the weights of the network in a way that minimizes the ERROR (described below).

5. Repeat step 1 through 4 for each vector in the training set until no training pair produces an error larger than a pre-decided acceptance level.

4.11 ADJUSTING WEIGHTS OF THE NETWORK

Adjusting the weights of the output layer is easier, as a target value is available for each neuron. The following shows the training process for a single weight from neuron “q” in the hidden layer “j” to neuron “r” in the output layer “k”. The output of a neuron in layer “k” is subtracted from its target values to produce an ERROR signal. This is multiplied by the derivative of a squashing function OUT (1-OUT) calculated for that neuron (“r”) thereby producing a “δ” value.

$$\delta = OUT * (1-OUT) * (TARGET - OUT)$$



$$\Delta_{W_{qr,k}} = \eta \delta_{r,k}\, OUT_{qj} \text{ ------------------------(1)}$$

$$W_{qr,k}(n+1) = W_{qr,k}(n) + \Delta W_{qr,k} \text{ ------------(2)}$$

Figure (4.9): adjusting weights of output layer

Where,

Wqr,j(n) = the value of weight from neuron in the hidden layer “j” to neuron “r” in the output layer “k” at step “n” (before adjustment).

Wr,j(n+1) = the value of weight from neuron in the hidden layer j to neuron “r” in the output layer “k” at step n+1 (after adjustment).

OUTq,j = the value of OUT of neuron in the hidden layer “j”.

ΔWqr,k = amount that Wqr,j to be adjusted.
Adjusting the weights of the hidden layer:

Back propagation trains the hidden layer by propagating the output ERROR back through the network layer by layer, adjusting weights at each layer. The same 2 equations (1) and (2) above are used for all layers, both output & hidden except that, for hidden layers the ή , δs values must be generated without the benefit of targets. The following figure explains how this is accomplished. δs for hidden layer neurons are calculated according to equation (3) by using the δs calculated for output layer (δy,k s) and propagating them backward through the corresponding weights.

$$\delta_{q,j} = OUT_{q,j}(1 - OUT_{q,j})(\Delta\delta_r, kW_{q,r}, k) \text{ ----------- (3)}$$

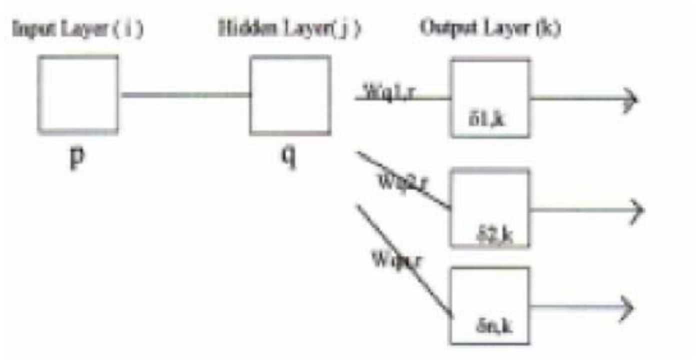


Figure (4.10): adjusting weights of hidden layer

Then, with δs in hand, hidden layer weights can be adjusted similar to the output layer weights as given below:

$$\Delta W_{qr,j} = \eta\delta_{q,j} OUT_{q,j} \text{ --------------- (4)}$$

$$W_{qr,j(n+1)} = W_{qr,j(n)} + \Delta W_{qr,j} \text{ -------------- (5)}$$

4.12 TESTING

In this paper testing data has taken from training data. We have just selected 100 random numbers of data from the training data and tested. Since Inverse Kinematics is **a** nonlinear mapping from (X, Y, Z, $\phi_x, \phi_y, and \phi_z$) space to **(**$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$**)** space, it can be regarded as an input - output process with an unknown transfer function. Approximation of such a relationship is an example of general approximation problem and **as** such it is well suited for learning by **a** Neural Network. Two problems encountered when teaching **a** NN robot Inverse Kinematics are low accuracy of Approximation and the need to create **a** teach data set with **a** relatively large number of data points. In the case where the kinematic parameters of each robot link are known, **a** simple kinematic model of the robot can be made based on equations 10(a) to 10(j) and **a** required number of data points can be readily generated. In the case where the joint parameters are unknown, the required data has to be obtained by measuring the position and orientation of the robot gripper for **a** number of arm configurations such measurement is very difficult in practice. The accuracy of NN approximation is a current research topic.

4.13 Summary

Artificial Neural Networks (ANN) has emerged as a powerful learning technique to perform complex tasks in highly nonlinear dynamic environments. Some of the prime advantages of using ANN models are their ability to learn based on optimization of an appropriate error function and their excellent performance for approximation of nonlinear function. So it is required to have basic knowledge of artificial neural networks that’s why we have discussed here.

CHAPTER

5

NOVEL ANN APPLICATION FOR PREDICTING OF INVERSE KINEMATICS OF MANIPULATOR

5.1 Introduction

The prime advantages of using ANN models are their ability to learn based on optimization of an appropriate error function and their excellent performance for approximation of nonlinear functions. Here in this paper two ANN architectures MLP and PPN are discussed. Both methods are widely used in present research scenario. In most of the field ANN models are preferred for predicting the values and optimizing the problems. ANN models especially MLP with back-propagation model can solve complex problem.

5.2 MULTI-LAYER PERCEPTRONS:

A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward. Multi-layer networks can represent arbitrary functions, but an effective learning algorithm for such networks was thought to be difficult. The weights determine the function computed. Given an arbitrary number of hidden units, any boolean function can be computed with a single hidden layer.

The Neuron

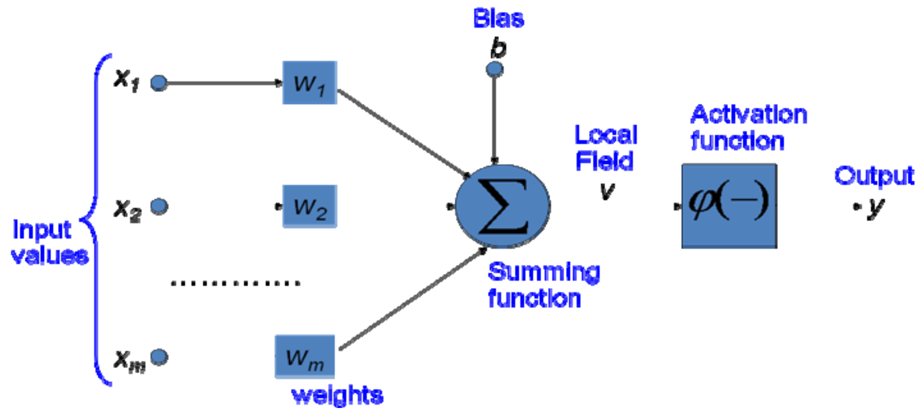


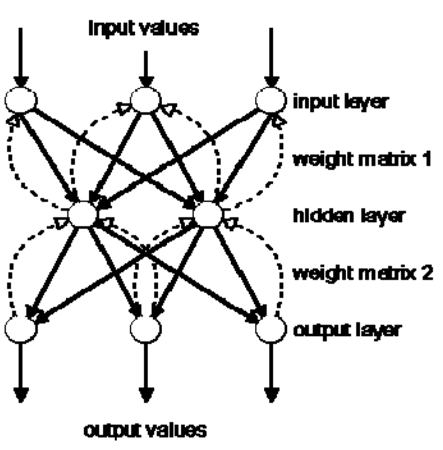
Fig (5.1) architecture for neuron

- The neuron is the basic information processing unit of a NN. It consists of:
- ➢ A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$
- ➢ An adder function (linear combiner) for computing the weighted sum of the inputs (real numbers): $u = \sum_{j=1}^{m} W_j X_j$
- ➢ Activation function (squashing function) φ for limiting the amplitude of the neuron output. $y = \varphi(u + b)$

I/O are bounded in [0,1] for the activation to perform;

Pass 1: Forward Pass - Present inputs and let the activations flow until they reach the output layer.

Pass 2: Backward Pass - Error estimates are computed for each output unit by comparing the actual output (Pass 1) with the target output. Then, these error estimates are used to adjust the weights in the hidden layer and the errors from the hidden layer are used to adjust the input layer.



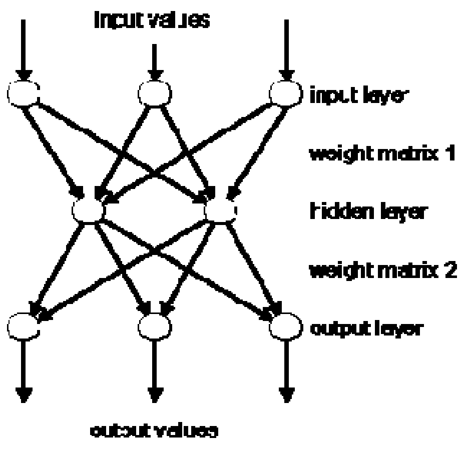


Fig. (5.2) Architecture of MLP Fig. (5.3) Architecture for back-propagation

The Back propagation Learning Routine:

As with perceptrons, the information in the network is stored in the weights, so the learning problem comes down to the question: how do we train the weights to best categories the training examples. We then hope that this representation provides a good way to categories unseen examples. In outline, the back propagation method is the same as for perceptrons:

1. We choose and fix our architecture for the network, which will contain input, hidden and output units, all of which will contain sigmoid functions.
2. We randomly assign the weights between all the nodes. The assignments should be to small numbers, usually between -0.5 and 0.5.
3. Each training example is used, one after another, to re-train the weights in the network. The way this is done is given in detail below.
4. After each epoch (run through all the training examples), a termination condition is checked (also detailed below). Note that, for this method, we are not guaranteed to find weights which give the network the global minimum error, i.e., perfectly correct categorization of the training examples. Hence the termination condition may have to be in terms of a (possibly small) number of miss-categorizations. We see later that this might not be such a good idea, though.

Learning rule for multilayer-perceptrons:

If the weight values are too large the net value will large as well; this causes the derivative of the activation function to work in the saturation region and the weight changes to be near zero. For small initial weights the changes will also be very small, which causes the learning process to be very slow and might even prevent convergence. The easiest method is to select the weights randomly from a suitable range, such as between (-0.1,0.1) or (- 2,2). The Learning Coefficient ή determines the size of the weight changes. A small value for ή will result in a very slow learning process. If the learning coefficient is too large the large weight changes may cause the desired minimum to be missed. A useful range is between 0.05 and 2 dependent on the problem. The influence of the ή on the weight changes is shown in Figure (17). The Momentum α causes the weight changes to be dependent on more than one input pattern. The change is a linear combination of the current gradient and the previous gradient. The useful range for this parameter is between 0 and 1. For some data sets the momentum makes the training faster, while for others there may be no improvement.

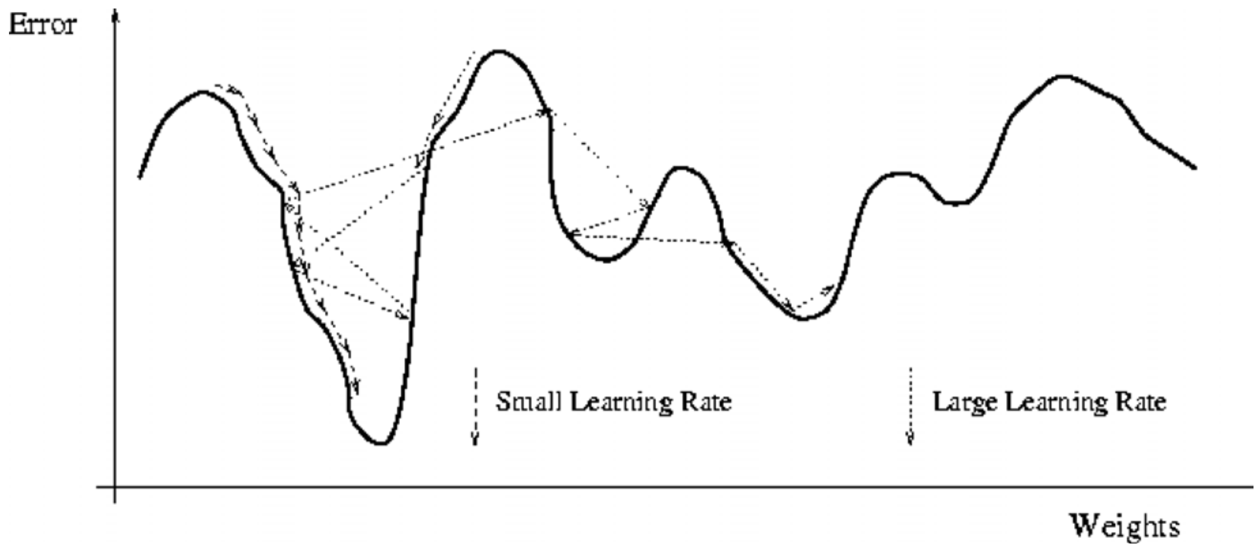


Fig (5.4) The Influence of the Learning Rate on the Weight Changes.

Where are neural networks being used?

- Signal processing: suppress line noise, with adaptive echo canceling, blind source separation
- Control: e.g. backing up a truck: cab position, rear position, and match with the dock get
  converted to steering instructions. Manufacturing plants for controlling automated machines.
- Siemens successfully uses neural networks for process automation in basic industries, e.g., in
- rolling mill control more than 100 neural networks do their job, 24 hours a day
- Robotics - navigation, vision recognition
- Pattern recognition, i.e. recognizing handwritten characters, e.g. the current version of
- Apple's Newton uses a neural net
- Medicine, i.e. storing medical records based on case information
- Speech production: reading text aloud (NET talk)
  Speech recognition
- Vision: face recognition , edge detection, visual search engines
- Business, e.g... rules for mortgage decisions are extracted from past decisions made by
  experienced evaluators, resulting in a network that has a high level of agreement with human experts.
- Financial Applications: time series analysis, stock market prediction
- Data Compression: speech signal, image, e.g. faces
- Game Playing: backgammon, chess.

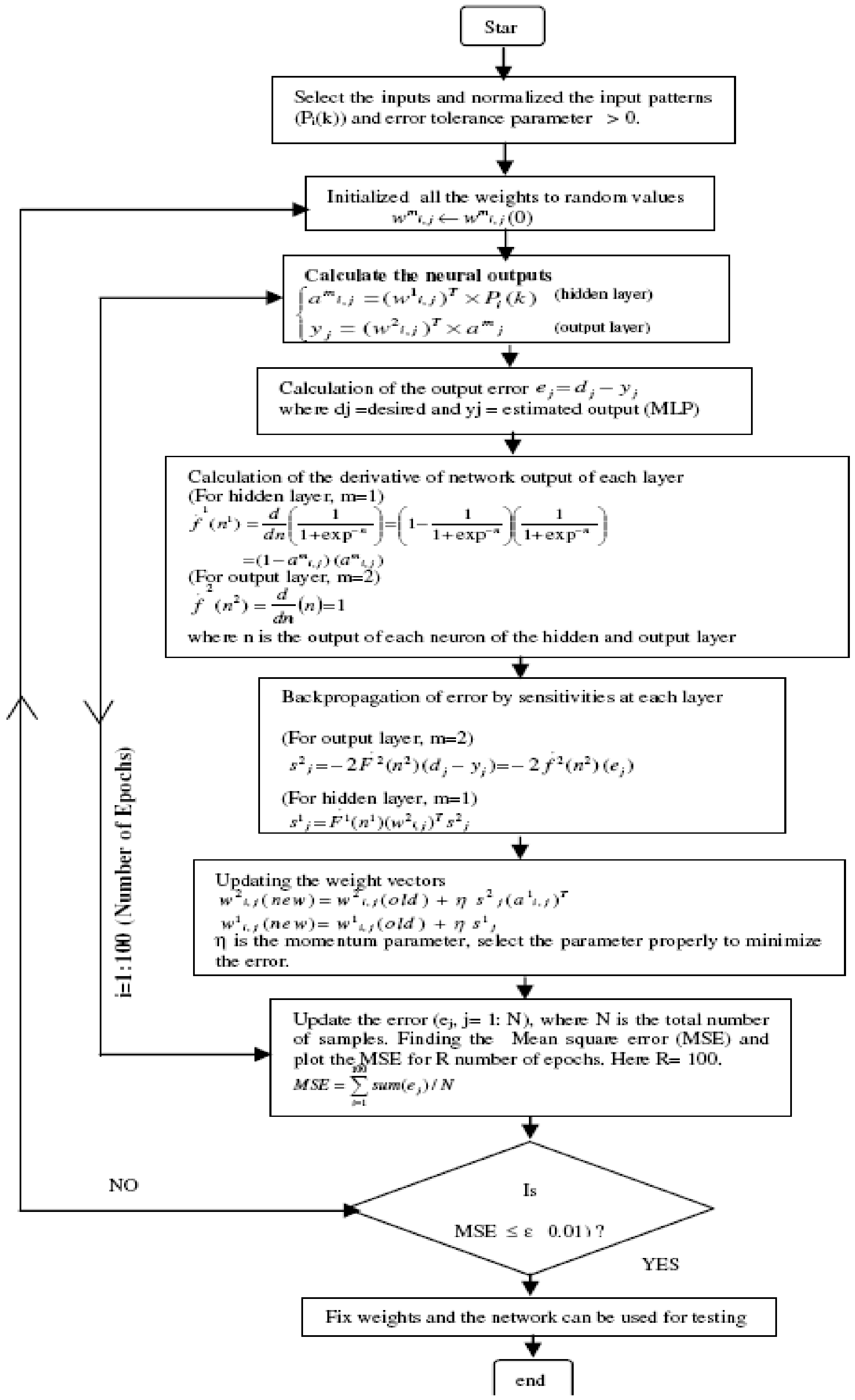

Star
Select the inputs and normalized the input patterns ($P_i(k)$) and error tolerance parameter > 0.
Initialized all the weights to random values $w^m{}_{i,j} \leftarrow w^m{}_{i,j}(0)$
Calculate the neural outputs
$a^m{}_{i,j} = (w^1{}_{i,j})^T \times P_i(k)$ (hidden layer)
$y_j = (w^2{}_{i,j})^T \times a^m{}_j$ (output layer)
Calculation of the output error $e_j = d_j - y_j$
where dj =desired and yj = estimated output (MLP)
Calculation of the derivative of network output of each layer
(For hidden layer, m=1)
$\dot{f}^1(n^1) = \frac{d}{dn}\left(\frac{1}{1+\exp^{-n}}\right) = \left(1-\frac{1}{1+\exp^{-n}}\right)\left(\frac{1}{1+\exp^{-n}}\right) = (1-a^m{}_{i,j})(a^m{}_{i,j})$
(For output layer, m=2)
$\dot{f}^2(n^2) = \frac{d}{dn}(n) = 1$
where n is the output of each neuron of the hidden and output layer
Backpropagation of error by sensitivities at each layer
(For output layer, m=2)
$s^2{}_j = -2\dot{F}^2(n^2)(d_j - y_j) = -2\dot{f}^2(n^2)(e_j)$
(For hidden layer, m=1)
$s^1{}_j = \dot{F}^1(n^1)(w^2{}_{i,j})^T s^2{}_j$
Updating the weight vectors
$w^2{}_{i,j}(new) = w^2{}_{i,j}(old) + \eta\, s^2{}_j (a^1{}_{i,j})^T$
$w^1{}_{i,j}(new) = w^1{}_{i,j}(old) + \eta\, s^1{}_j$
η is the momentum parameter, select the parameter properly to minimize the error.
Update the error (e_j, j= 1: N), where N is the total number of samples. Finding the Mean square error (MSE) and plot the MSE for R number of epochs. Here R= 100.
$MSE = \sum_{i=1}^{100} sum(e_j) / N$
i=1:100 (Number of Epochs)
NO
Is MSE ≤ ε 0.01) ?
YES
Fix weights and the network can be used for testing
end

Fig (5.5). Flow chart for MLP

5.3 THE POLYNOMIAL PERCEPTRON NETWORK:

Weierstrass approximation theorem states that any function which is continuous in a closed interval can be uniformly approximated within any prescribed tolerance over that interval by some polynomial. Considering a binary PAM system (i.e. K = 2 in (1)) the channel equalization becomes a two class classification problem, and a decision boundary can be established between the pattern classes. Fig.(20) depicts a PPN network where X is the input pattern given by

$$X = [x_1, x_2, x_3, ...x_m]^T$$

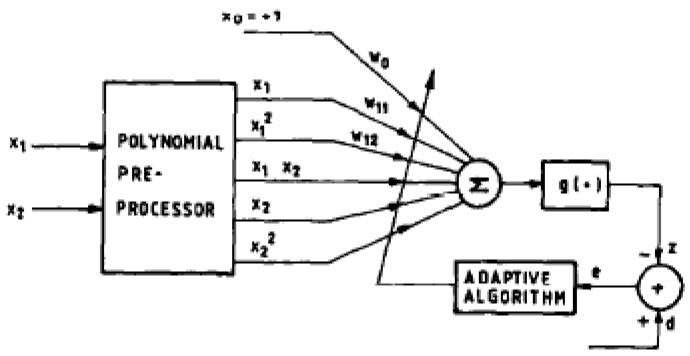


Fig.(5.6) Polynomial perceptron network.

Considering a two-dimensional pattern X = [x1 x2]’ and polynomial order 2, the decision function (14) may be written as

$$DF(X) = W^T X^T$$

Where,

$$W = [w_0, w_{11}, w_{12}; w_{22}, w_1; w_2]^T$$

And

$$X^* = [1, x_1{}^2, x_1, x_2, x_2{}^2, x_1, x_2]^T$$

For output layer (P = L),

For other layers $(p = L - 1, L - 2, . . . , 1)$

The general quadratic case can be formed by considering all combination of components of X which forms terms of degree two or less. Thus, for an M-dimensional pattern,

$$DF(X) = \sum_{j=1}^{M} w_{jj} x_{jj}^{\;2} + \sum_{j=1}^{M-1} \sum_{k=j+1}^{M} w_{jk} x_j x_k + \sum_{j=1}^{M} w_j x_j + w_0 = W^T X^*$$

The number of terms needed to describe a polynomial decision function grows rapidly as the polynomial degree r and the dimension ***M*** of the pattern increases. For the M-dimensional case, the number of coefficients in a function of rth degree is given by

$$N_{M_r} = {}^{M+r}C_r = \frac{(M+r)!}{M!r!}$$

The input pattern X to the PPN at time n is the channel output vector X(n). This is then converted into X*(n) by passing it into a polynomial preprocessor. The weighted sum of the components of X*(n) is passed through a nonlinear function sigmoid and purelinear function to produce the output z (n). The output of the PPN is compared with the desired response to generate an error s (n) which is then used to update its weights by the BP algorithm.

5.4 Summary

The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units. The hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents. Here in this paper two ANN architectures MLP and PPN are discussed. Both methods are widely used in present research scenario. In most of the field ANN models are preferred for predicting the values and optimizing the problems. ANN models especially MLP with back-propagation model can solve complex problem.

CHAPTER

6

SIMULATION RESULTS AND PERFORMANCE ANALYSIS:

To validate the performance of MLP and PPN for inverse kinematics problem, simulation studies are carried out by using MATLAB.

In this work the training data sets were generated by using equation 10-10(j). A set of 130 data sets were first generated as per the formula equation (10) for this the input parameter X and Y coordinates in inches. Using these data sets was basis for the training and evaluation or testing the MLP and PPN models. Out of the sets of 130 data points, 100 were used as training data and 30 were used for testing for MLP. Back-propagation algorithm was used for training the network and for updating the desired weights. In this work epoch based training method was applied.

6.1 Sumilation result of MLP

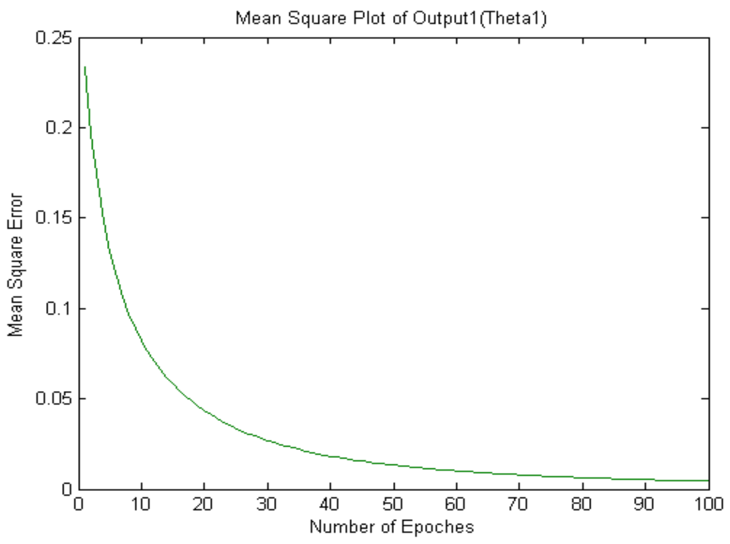


Fig (6.1) mean square error for θ_1

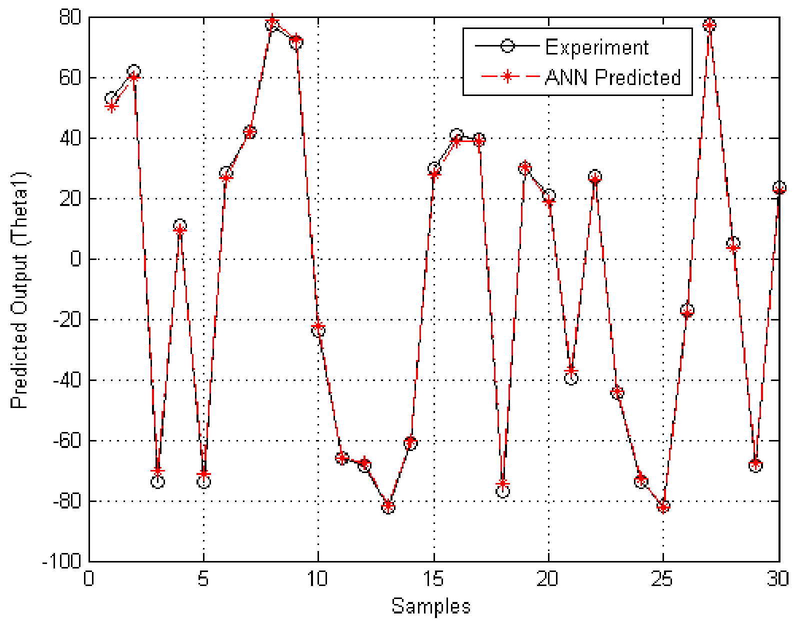
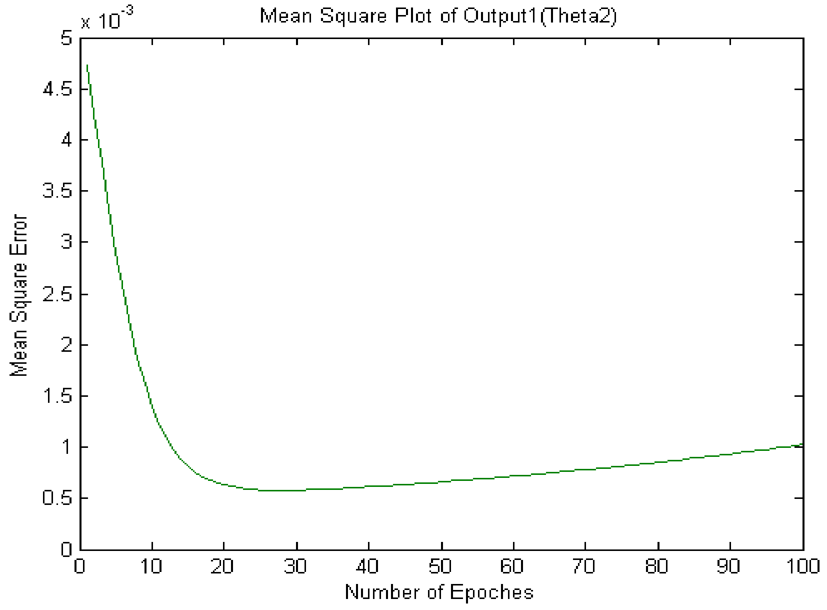


Fig (6.2) graph for matching of desired and predicted values of



θ_1

Fig (6.3) mean square error for θ_2

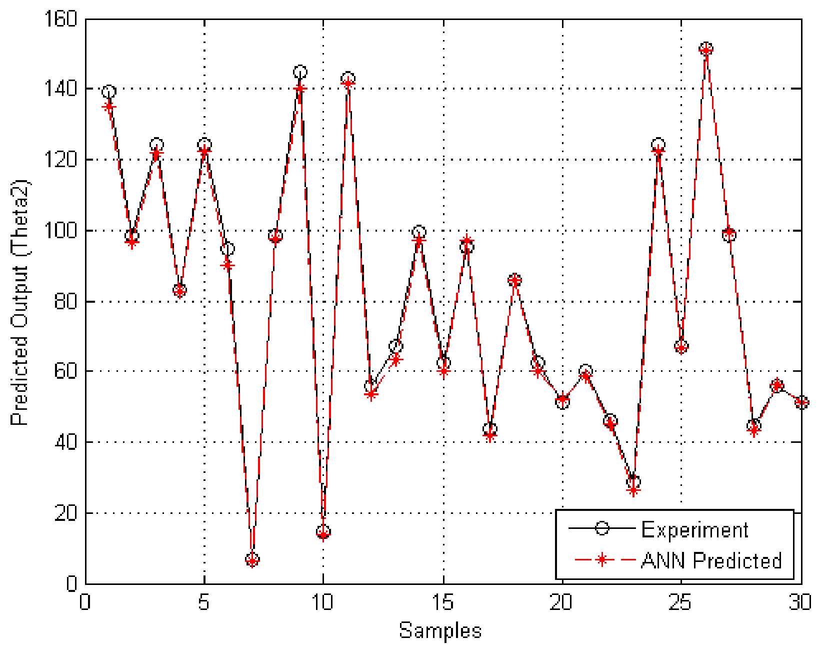


Fig (6.4) graph for matching of desired and predicted values of θ_2

Table (6.5) comparison table for desired data and predicted data by ANN

θ_2	Predicted	θ_1	Predicted
139.0499	135.0321	52.7038	50.3645
98.5943	96.5845	61.9182	59.6542
124.0105	122.0475	-73.9847	-70.2341
83.1976	82.65	10.7648	9.3426
124.0105	122.24	-73.9847	-71.5486
94.9685	90.15	27.9406	26.6451
6.8891	6.1521	42.0630	41.6572
98.6345	97.7523	77.0242	78.6512

144.9305	140.2147	71.2636	72.3542
14.8260	13.8241	-23.7413	-22.5412
142.7669	141.5321	-65.9324	-66.2145
55.7833	53.4712	-68.6521	-67.6423
66.9949	63.5342	-82.5974	-81.6421
99.3724	97.2136	-61.2022	-60.2341
62.4008	60.2145	29.8007	27.6324
95.0525	97.1254	40.7122	38.6134
43.6812	41.6532	39.1977	38.9423
86.0279	85.6542	-77.3040	-74.3692
62.4008	60.2451	29.8007	30.4765
51.1973	52.1234	20.8457	18.6324
59.9854	58.6945	-39.5220	-37.3245
46.0676	45.0214	27.1536	26.1348
28.8915	26.5412	-44.5580	-43.8945
124.0105	122.2345	-73.9847	-72.8643
67.0224	66.5321	-81.6532	-82.1536
151.4063	150.9527	-17.1316	-18.1142
98.6345	99.2145	77.0242	76.9850
44.5800	43.2587	4.9643	3.5843
55.7833	56.3210	-68.6521	-67.5216

51.4107	51.1024	23.6330	22.2435

6.2 Simulation result of PPN

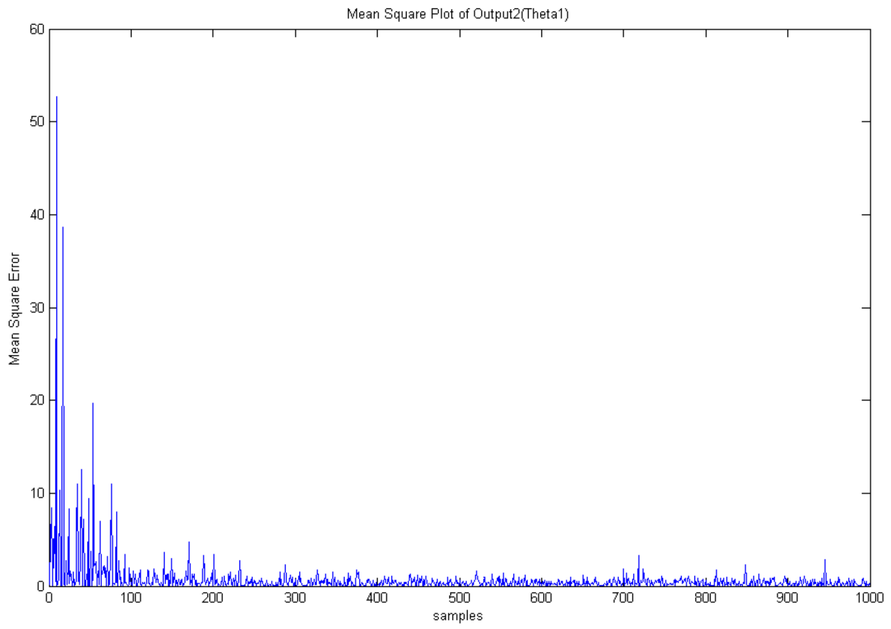


Fig (6.6) Mean square error for $\theta 1$

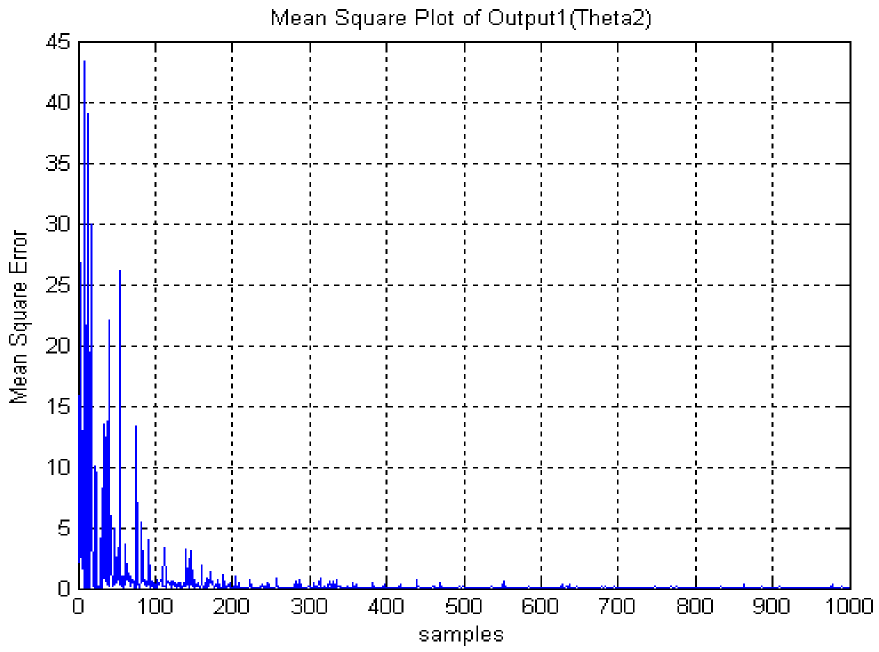


Fig (6.7) Mean square error for $\theta 2$

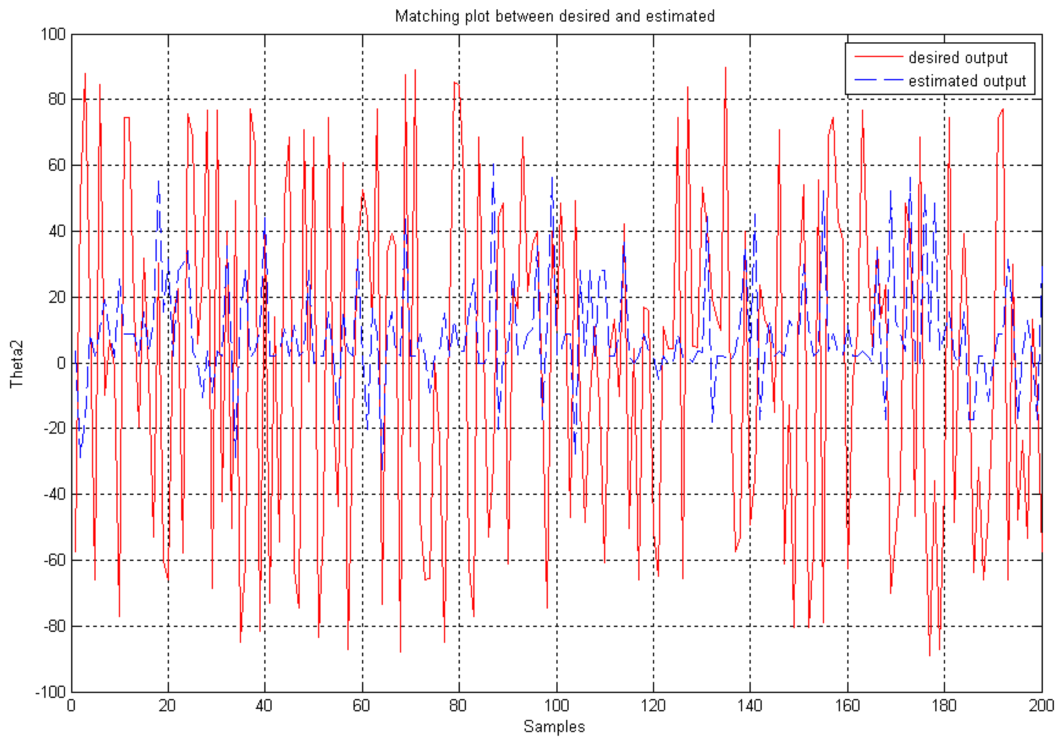


Fig (6.8) graph for matching of desired and predicted values of θ_2

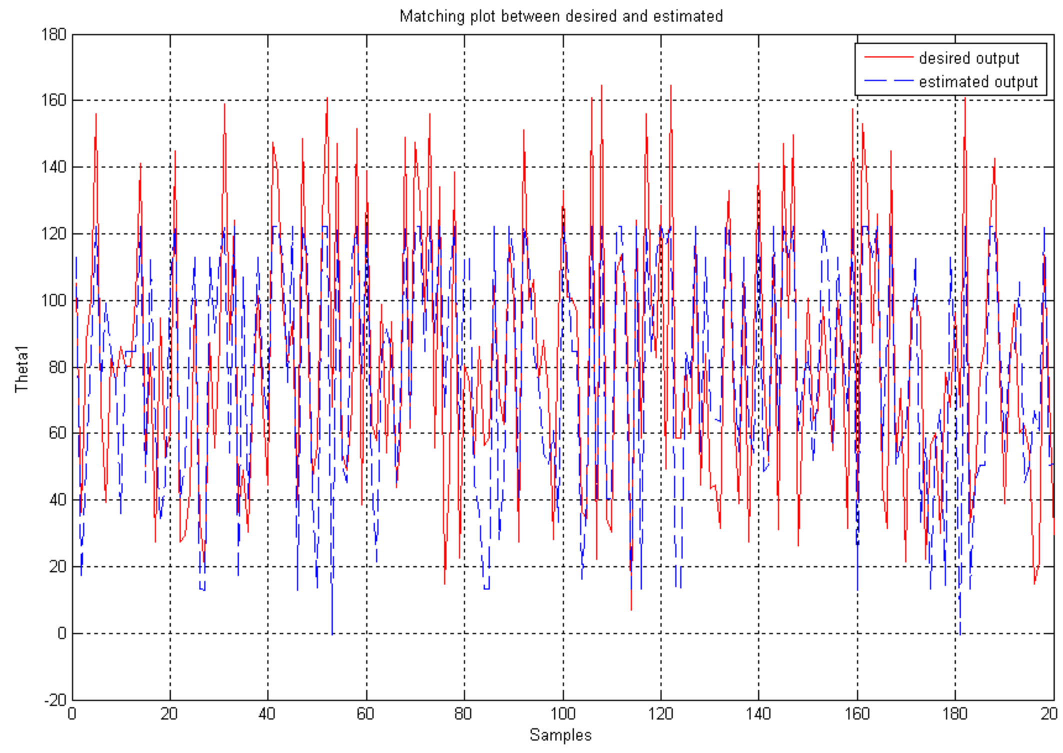


Fig (6.9) graph for matching of desired and predicted values of $\theta 1$

The performance of MLP network is discussed first, were 16 nodes selected in the hidden layer. The MSE (mean square error) of the MLP model for 100 epochs for θ_1 was represented in fig (21). MSE (mean square error) for second output (θ_2) for 100 epoch represented in fig (22). To test the stability of the models validation data or testing data is essential as discussed earlier 30 data points were selected randomly for testing the MLP model. Fig (23) represents the performance of the model for 30 testing samples or validation samples for output one (i.e. θ_1) and fig (24) represents the performance of output two (θ_2). The similar results are presented in table (1). It may be noted that the performance of MLP is very closer to the experimental result.

CHAPTER 7

CONCLUSION AND SCOPE FOR FUTURE WORK:

Mathematical models relay on assuming the structure of the model in advanced, which may be sub-optimal. Consequently, many mathematical models fail to simulate the complex behavior of inverse kinematics problem. In contrast, ANN (artificial neural networks) is based on the data input/output data pairs to determine the structure and parameters of the model. Moreover, ANN's can always be updated to obtain better results by presenting new training examples as new data become available. From the present study it was observe that the MLP gives the better results as compared to PPN for inverse kinematics problem. This artificial neural network based joint angles prediction model can be useful tool for the production engineer's to estimate the motion of the manipulator accurately.

FUTURE SCOPE:

In this study the MLP and PPN has been proposed for the solution of inverse kinematics problem of robot manipulator. However, it has some limitations. There are several types of soft computing methods are available which can be used for finding the solution, but this is beyond the scope of this thesis but this technique can be used for the future scope of the thesis. These methods are followed:

- Application of fuzzy inference system (FIS)
- Adaptive network based fuzzy inference system (ANFIS)
- Functional link artificial neural network (FLANN)
- Evaluation computation

CHAPTER 8

REFERENCE:

1. Alavandar S. and Nigam M. J. "Neuro-Fuzzy based Approach for Inverse Kinematics Solution of Industrial Robot Manipulators", Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844 Vol. III (2008), No. 3, pp. 224-234.
2. Morris A. S. And A. Mansor A. "artificial neural network for finding inverse kinematics of robot manipulator using look up table", Robotica (1997) volume 15 , pp 617 – 625 . Printed in the United Kingdom Ö 1997 Cambridge University Press.
3. Ahmad Z. and Guez A. "On the Solution to the Inverse Kinematic Problem", department of electrical and computer engineering Drexel University Philadelphia PA 19104.
4. Bekir Karlik, Serkan Aydin "An improved approach to the solution of inverse kinematics problems for robot manipulator".
5. *L.* Npyen. "Neural Network Architectures For The Forward Kinematics Problem in Robotics".
6. G. Jaein, "Robot Control Using Neural Network".
7. Jenhwa Guo and Vladimir Cherkassky "Developed A Solution to the Inverse Kinematic Problem in Robotics Using Neural Network Processing" University of Minnesota Minneapolis, Minnesok 55455.
8. Dali wang and ali zilouchain "Solutions of Kinematics of Robot Manipulators Using a kohonen Self- Organizing Neural Network". department of Elect. Engineering Florida Atlantic University Boca Raton FL 33431.

9. Youshen Xia and Jun Wang "A Dual Neural Network for Kinematic Control of Redundant Robot Manipulators".
10. Choon seng Yee, Kah-bin Lim "Forward kinematics solution of Stewart platform Using neural networks".
11. E. A. Al-Gallaf "Neural Networks for Multi-Finger Robot Hand Control", JKAU: Eng. Sci., Vol. 19 No. 1, pp: 19-42 (2008 A.D. /1429 A.H.)
12. Asko Houvinen and Heikki Handroos "ADAMS –Model for training of neural network for inverse kinematics of flexible robot manipulator".
13. H. Daniel Patiño "Neural Networks for Advanced Control of Robot Manipulators".
14. Ted Hesselroth, "Neural Network Control of a Pneumatic Robot Arm".
15. B. Benhabib, "A solution to the inverse kinematics".
16. Oyama and Tachi "Inverse kinematics computation using an artificial neural network".
17. Manocha and Canny "An algorithm and implementation for efficient inverse kinematics for a general *6R* manipulator".
18. Kozalziewicz C. Ogiso T. and Miyake *N.* "Partitioned Neural Network architecture for Inverse Kinematic calculation *of* a 6 dof robot manipulator".
19. Mayorga*V. and Sanongboon P. "Inverse kinematics and geometrically bounded singularities prevention of redundant manipulators: An Artificial Neural Network approach". Robotics and Autonomous Systems 53 (2005) 164–176.
20. Kieffe S. Morella V. and Donath M. "Neural Network Learning of the Inverse Kinematic Relationships for a Robot Arm" Proceedings of the 1991 IEEE Intemational conference on Robotics and Automation Sacramento, Califomia - April 1991.
21. Rao D. H. and Gupta M. M. "Performance Comparison of Dynamic Neural Networks as Applied to Robot Inverse Kinematic Computations". Proceeding of the American Control Conference Baltlmorr, Mayland June 1994.

22. Kim S.W. Lee J.J. and Sugisaka M. “Inverse kinematics solution based on fuzzy logic for redundant manipulators”. Proceedings of the 1993 IEEE/RSJ Intemational Conference on Intelligent Robots and Systems Yokohama, Japan July 2630,1993.
23. Velagic J. Osmic N. and Lacevic B. “Neural Network Controller for Mobile Robot Motion Control”. International Journal of Intelligent Systems and Technologies 3; 3 www.waset.org summer 2008.
24. H. Sadjadian , H.D. Taghirad *Member, IEEE* and A. Fatehi “Neural Networks Approaches for Computing the Forward Kinematics of a Redundant Parallel Manipulator”. International Journal of Computational Intelligence 2; 1 winter 2006.
25. Ding H., Member, IEEE, and Wang J. Senior Member, IEEE, “Recurrent Neural Networks for Minimum Infinity-Norm Kinematic Control of Redundant Manipulators”. IEEE Transaction on systems, man, and cybernetics part-A: systems and humans, vol.29, N0.3 May 1999.

BOOKS REFERRED:

- Robert J. Schilling for robotics
- John Craig for robotics
- S.R.Deb for robotics
- Haykins for neural network
- Hagen for neural network