12-2007

Inverse Kinematics Based on Fuzzy Logic and Neural Networks for the WAM-Titan II Teleoperation System

Joong-kyoo Park
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Joong-kyoo Park entitled "Inverse Kinematics Based on Fuzzy Logic and Neural Networks for the WAM-Titan II Teleoperation System." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

William R. Hamel, Major Professor

We have read this thesis and recommend its acceptance:

Dongjun Lee, J. Wesley Hines

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Joong-kyoo Park entitled "Inverse Kinematics Based on Fuzzy Logic and Neural Networks for the WAM-Titan II Teleoperation System." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

William R. Hamel

Major Professor

We have read this thesis
and recommend its acceptance:

Dongjun Lee

J. Wesley Hines

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the
Graduate School

(Original signatures are on file with official student records.)

Inverse Kinematics Based on Fuzzy Logic and Neural Networks for the WAM-Titan II Teleoperation System

A Thesis Presented for
the Master of Science Degree
The University of Tennessee, Knoxville

Joong-kyoo Park
December 2007

Dedicated to my parents, wife, and two babies!

ACKNOWLEDGEMENTS

I would like to thank Dr. W. R. Hamel who was the academic advisor and the major professor for this thesis. Also I would like to thank Dr. D. Lee and Dr. J. W. Hines for their support and encouragement. In specially, I would like to thank Heather Humphreys who has supported my thesis works and her good friendship. Andrzej Nycz is a good friend and has always given me good advice and suggestion. I would like to thank Mark Noakes who has helped me with his broad knowledge.

I would like to thank my father who has supported me with encouragement and finance. I thank my wife and our two babies who give me love and support.

ABSTRACT

The inverse kinematic problem is crucial for robotics. In this paper, a solution algorithm is presented using artificial intelligence to improve the pseudo-inverse Jacobian calculation for the 7-DOF Whole Arm Manipulator (WAM) and 6-DOF Titan II teleoperation system. An investigation of the inverse kinematics based on fuzzy logic and artificial neural networks for the teleoperation system was undertaken. Various methods such as Adaptive Neural-Fuzzy Inference System (ANFIS), Genetic Algorithms (GA), Multilayer Perceptrons (MLP) Feedforward Networks, Radial Basis Function Networks (RBF) and Generalized Regression Neural Networks (GRNN) were tested and simulated using MATLAB. Each method for identification of the pseudo-inverse problem was tested, and the best method was selected from the simulation results and the error analysis.

From the results, the Multilayer Perceptrons with Levenberg-Marquardt (MLP-LM) method had the smallest error and the fastest computation among the other methods. For the WAM-Titan II teleoperation system, the new inverse kinematics calculations for the Titan II were simulated and analyzed using MATLAB. Finally, extensive C code for the alternative algorithm was developed, and the inverse kinematics based on the artificial neural network with LM method is implemented in the real system. The maximum error of Cartesian position was 1.3 inches, and from several trajectories, 75 % of time implementation was achieved compared to the conventional method. Because fast performance of a real time system in the teleoperation is vital, these results show that the

new inverse kinematics method based on the MLP-LM is very successful with the acceptable error.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1:Introduction

1.1. Overview

This thesis focuses on an investigation of inverse kinematics based on fuzzy logic and artificial neural network for the WAM-Titan II telerobotic system. The teleoperation system has a redundant mechanical manipulator, which serves as the master controller, and a non-redundant mechanical manipulator, which is the slave manipulator. This non-replica test bed was developed in the Robotics and Electro-Mechanical System Laboratory (REMSL) at the University of Tennessee. The system includes a 7 degree-of-freedom (DOF) Barrett WAM manipulator as a master controller and a 6-DOF Schilling Titan II salve manipulator as shown in Figure 1-1. Because of their difference in numbers of degrees of freedom, Cartesian space control is needed instead of joint space control.

In particular, this research addresses an alternative inverse kinematics design of the manipulators to reduce computations and to improve general performance. First, an investigation was performed to find identification of inverse kinematics for a 3-DOF planar redundant manipulator using a fuzzy logic with Adaptive Neuro-Fuzzy Inference System (ANFIS) and Genetic Algorithm (GA). Using a circle trajectory, the errors between desired outputs and actual outputs were compared. Second, similarly, an artificial neural network was used for finding a substitute inverse kinematics solution, using Multilayer Perceptrons - Back Propagation (MLP-BP) with a Levenberg Marquardt

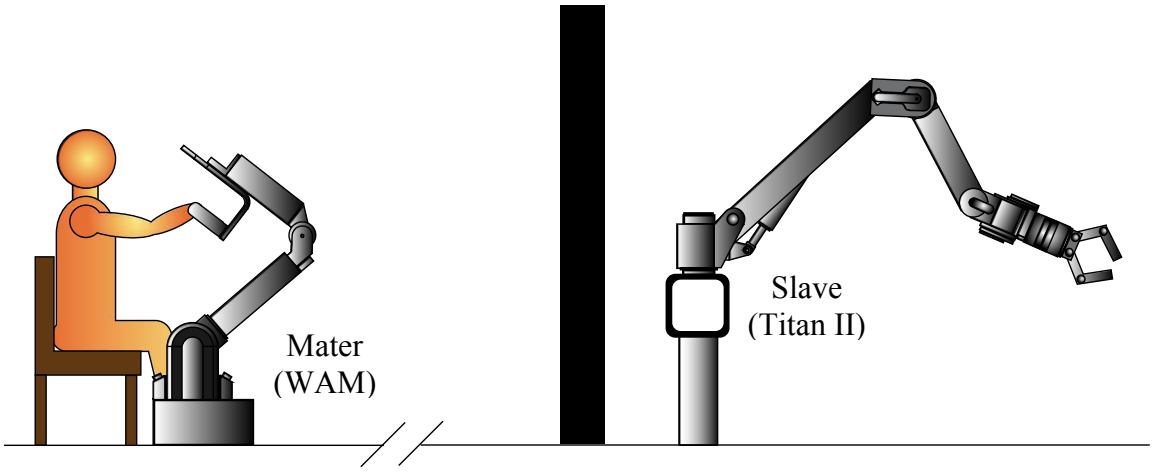

Figure 1-1. WAM-Titan II Teleoperation System

optimization method Radial Basis Function Network (RBF), and Generalized Regression Neural Network (GRNN). The optimal inverse kinematics solution, which was the MLP-BP, was compared with the results found from the experiments. An inverse kinematics solution for the Titan II based on the MLP-BP was tested and analyzed with current system. In the next section, background of teleoperation and artificial intelligence is discussed.

1.2. Background

From the ancient era, humans have discovered and invented tools to overcome their physical inability, and they have desired the existence of intelligent machines which can perform as human slaves. This led to the field of robotics, which has developed rapidly after development of computer technology. Various robots are built for the

purpose of physical aids, and they have been used in many areas. For instance, robotic manipulators have been used in manufacturing industry, hazardous material handling, and applications in dangerous environments such as oceans and space. However, due to the limitations in the current technology for sensing and controlling, human supervision is required in environments of unknown structure. In addition, in dangerous environments, humans cannot be present physically to perform work together with a robot. As a result, in the above cases and in many other applications, teleoperation is an interesting method to be achieved without position of an operator near the manipulator, where autonomous operation is not feasible. Traditionally, the term teleoperation refers to a human operator's use of a master controller to operate a slave manipulator at a distance. Therefore, many researchers have worked in the teleoperation field these days. It is likely that teleoperation will spread widely in the future.

An ideal teleoperation system for real situations should be capable of human-like performance of dedicated tasks with remote human assistance. The operator should have intelligent vision and tactile/kinesthetic feedback. This allows for efficient interaction without direct contact between the operator and the manipulator. Since this technology is not yet developed enough, performance of remote tasks like a real human is always limited in current era. Therefore, still many researchers are working on haptic telepresence, which involves transfer of feeling operation to the operator. There is a variety of other issues for teleoperation such as performance, stability, communication, and time delay between a slave and a master [1].

In teleoperation, two different approaches are used for achieving mapping: joint-to-joint mapping and Cartesian space mapping. In joint-to-joint mapping, an operator is

able to control each joint directly with the complete slave manipulator configuration. In Cartesian space mapping, however, position and orientation or force and moment of the end-effector are controlled by the operator. Joint-to-joint mapping, in general, requires a system with similar kinematic structure of the master and the slave. Both should be either identical or scaled to each other. In the system, the slave joint actuator responds directly to the kinematically corresponding master joint actuator. Because transformations from joint space to Cartesian space and vice versa are not needed in this case, a fast and reliable response is obtained easily. Cartesian space mapping has some benefits; it allows the referencing of the positions of the manipulators, and it provides the ability to operate a redundant manipulator. However, because it involves coordinate transformations, Cartesian space mapping is computationally intensive and has singularity problems. Computational complexities result in time delays and instability from the delays.

The master manipulator of the WAM-Titan II teleoperation system is a Barrett WAM, which is a 7-DOF cable-driven back-drivable arm. This manipulator has centralized cables to transmit power to every joint; this feature provides no backlash and low friction [12]. It has built-in sensors to measure angles, forces, and torques of each joint. The WAM also has other features like back-drivability, gravity compensation, and force feedback capability. The back-drivability allows the WAM joints to be actuated by external forces, and it makes the WAM suitable for a master. The force feedback uses the back-drivability to measure force applied at the arm. In addition, the gravity compensation gives the operator the ability to move the arm smoothly. The standard WAM has 4-DOF, and, for the WAM-Titan II teleoperation, 3-DOF gimbals are attached at the end of the arm. This allows the master to be a dexterous redundant manipulator.

For a redundant manipulator, redundancy resolution can be undertaken to use the benefits of the extra DOF. The WAM is attached to the Compact Remote Console (CRC) platform in the teleoperation system. The CRC is an integrated vision assist system, which has four LCD video monitors, two LCD computer monitors, and video control units to achieve a broad range of remote operations. The CRC provides an ergonomic teleoperation workstation for viewing and controlling manipulators.

The slave manipulator is a 6-DOF Schilling Titan II manipulator. This slave was originally designed and manufactured for underwater applications. This hydraulic manipulator originally had a parallel jaw gripper at the end, but the gripper was replaced with Barrett's three fingered hand, called the Wraptor, shown in Figure 1-2. At each joint, a resolver is used for measuring the rotation. It is a non-back-drivable manipulator, unlike the WAM, due to its hydraulic characteristics. The slave arm's six joints are azimuth, yaw, shoulder pitch, elbow pitch, wrist pitch, wrist yaw, and wrist rotation.

Figure 1-2. The Barrett's three fingered hand

In the WAM-Titan II teleoperation system, for achieving spatial association between the two manipulators, a Cartesian space mapping technique is used to control the slave robot following the same trajectory of the master controller, due to the kinematic dissimilarity between these two devices [2, 3]. The mapping technique uses two differential kinematic methods: forward kinematics and inverse kinematics. This mapping is described by a Jacobian matrix. The Jacobian is useful for finding singular configurations, analyzing redundancy, and determining inverse kinematics algorithms.

As shown in Figure 1-3, the WAM-Titan II teleoperation system uses forward kinematics followed by inverse kinematics. The forward kinematics calculate the Cartesian space velocities of its end-effector from the measurement of the WAM joint angles, and then the velocities are used as command inputs to the Titan II inverse

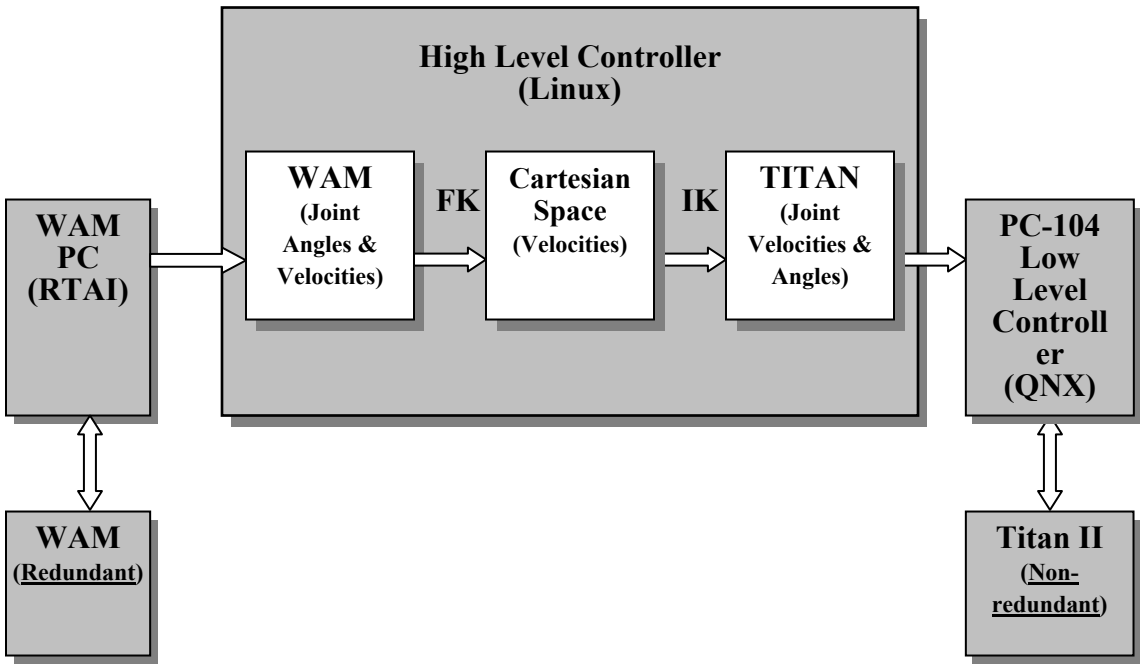

Figure 1-3. WAM-Titan II Teleoperation system

kinematics, thereby producing the joint velocities. The joint angles are finally used as Titan II command inputs.

Artificial intelligence is a research area that seeks to implant human-like intelligence. The artificial intelligence research is generally required in many areas like robotics, image and voice recognition, decision-making, non-linear controls, and uncertain or complex systems. Recently the field of artificial intelligence covers a number of technologies, including artificial neural networks, fuzzy logic, genetic algorithms, Bayesian networks, and chaotic theory [4]. Most of these technologies have developed significantly in recent years, gaining well-known use due to showing significant promise in several engineering applications. However, artificial intelligence is still limited in terms of general purpose applications, and more research is needed to solve many problems.

The majority of current applications are supported by fuzzy logic and artificial neural networks. Both methods are commonly used to control complex and uncertain systems. Fuzzy logic is the theory to adapt a rule-and-inference based reasoning approach to represent fuzzy sets, rather than crisp sets, of input and output numbers in linguistic forms [4, 5]. The advantages of fuzzy logic are robustness from noise or uncertain failures and the ability to handle nonlinear systems. However, the main disadvantage is the lack of a formal process to define a rule base, especially in unknown system. Artificial neural network theory represents a system by training exact input and output data, and formulates an approximation model of the system. This method is very effective when the data sets are exact and collected from all possible ranges of the system. Also, artificial neural networks require long training time to optimize weights and biases. These

two methods can be combined as integrated systems to aid each other mutually. In this paper, fuzzy logic systems and neural networks are introduced to solve inverse kinematics. Additionally, a hybrid system like ANFIS and Fuzzy-GA is proposed to define fuzzy rules numerically. In the next section, motivation of this thesis is presented.

1.3. Motivation

The main issue which should be addressed is the inverse kinematics. The solution of the inverse kinematics is complex because of the nonlinearities; as a result, a closed form solution may not be found. Multiple or infinite solutions may exist when the Jacobian matrices are rank deficient or manipulators are kinematically redundant. Even if the inverse kinematics has a closed form solution, unstable movements may happen near the singularities.

In inverse kinematics of redundant manipulators, the extra degrees of freedom can be effectively used to improve the manipulator's ability to avoid obstacles or singular points. On the other hand, inverse kinematics of redundant manipulators is more complex than non-redundant manipulators. Mapping between position space and joint space has always been difficult for redundant manipulators because of the presence of a non-square Jacobian matrix. Therefore, some constraints are needed to make the Jacobian a square and non-singular matrix.

To solve the inverse kinematics problems, various computational schemes have been developed. However, a major difficulty in solving inverse kinematics is associated with demanding computations required to solve pseudo-inverse calculations. To reduce

the computational complexity of inverse kinematics and redundant resolution, fuzzy and neural network methods are used in this paper. The next section outlines the structure of this thesis.

1.4. Thesis Outline

Chapter 2 discusses a general concept of the kinematics. Homogeneous transformation matrices and Denavit-Hartenberg parameters are introduced. The forward kinematics is formulated using the transformation matrix, and differential kinematics is presented with the Jacobian matrix. The inverse kinematics is also analyzed and its differential kinematics is described by the geometric Jacobian matrix. Singular configurations are characterized, and several methods are discussed to avoid the singularities.

Chapter 3 introduces the WAM-Titan II teleoperation, and the specifications are discussed. DH parameters for the WAM and Titan II manipulators are developed, and kinematic transformations for both manipulators and the formulation of the corresponding Jacobian are provided. Moreover, the architecture of the teleoperation system is explained.

Chapter 4 presents the fuzzy logic and the artificial neural network. Brief reviews of the fuzzy logic and the artificial neural network are provided. For the fuzzy logic, ANFIS and GA are introduced. For the artificial neural networks, MLP-BP with LM, RBF, and GRNN are discussed.

Chapter 5 describes the simulation approach for inverse kinematics of a 3-DOF planar manipulator using the methods explained in Chapter 4. Kinematics of the planar manipulator is analyzed, and details for the each method are explained.

Chapter 6 discusses the simulation results for the planar manipulator, and they are analyzed to find the best method. The MLP-BP is chosen for the real teleoperation system, and the final simulation experiments for the Titan II manipulator are performed using MLP-BP. The results are discussed in detail. A summary and suggestions for future work are given in Chapter 7.

CHAPTER 2:Kinematics

2.1. Introduction

This chapter is devoted to the kinematics of serial type robotic manipulators. The study is general, but is focused in the WAM-Titan II Teleoperation System manipulators in Chapter 3. Kinematics is the study of dynamics and generally deals with the motion of bodies. Therefore, the kinematics of robotic manipulators involves the geometric or time-based properties of the motion, such as position, velocity, acceleration, or higher order derivatives [6]. A robotic manipulator is considered as a set of chain links connected by joints, and a joint appears between adjacent links. One end of the kinematic chain is fixed to a base, and the other end, called an end-effector generally is positioned. The links are numbered from the base to the end of the chain. The joints also are numbered in same manner. Generally, manipulators have revolute joints or prismatic joints. The revolute joint is usually considered as one axis of rotation of connected links. The prismatic joint, which can slide, is considered as one axis of translation of connected links. Therefore, generally a joint has a single degree of freedom, and n joints of a manipulator have n degrees of freedom.

Typically, a manipulator needs 6-DOF to operate in three-dimensional space. Therefore, at least six joints are required in order to manipulate: three for position and three for orientation. In particular, a planar manipulator needs at least three joints: two for

position and one for orientation. Therefore, a function in terms of joint angles characterizes the position and the orientation of the end-effector. If a manipulator has more degrees of freedom than those required to operate, then the manipulator is kinematically redundant, and it is called a redundant manipulator. Redundant manipulators easily achieve more dexterous motions.

Forward kinematics determines position and orientation from joint angles. In contrast to forward kinematics, inverse kinematics determines joint angles from position and orientation. Forward kinematics always has a unique solution, but inverse kinematics may have infinite solutions. The relationship between the joint velocities and the linear/angular velocity of the end-effector is given by differential or velocity kinematics. This can be described by a Jacobian matrix. The Jacobian matrix has the current manipulator configuration, and it can be computed by differentiating the forward kinematics with respect to the joint variables. From the velocity relationship with the Jacobian matrix, the forward and inverse kinematics can be also described instead of position and orientation. Indeed, the differential kinematics is useful when the manipulator characteristics need to be analyzed. Especially, in the inverse kinematics, the Jacobian matrix is the main tool to perform and determine inverse kinematics algorithms. Calculation of kinematics using homogeneous transformations is introduced in the next section.

2.2. Homogeneous Transformation

To determine the kinematic relationship between joint angles and position/orientation, a transformation from a base to an end-effector is necessary. For this,

a homogeneous transformation is defined in [6~8]. Typically a position of any joint is expressed with respect to a base reference frame, and the orientation is also expressed in terms of the three unit vectors with respect to the same reference frame. This expression is presented by the relationship between the coordinates of the same point in two different fames, *o* and *a*. If $\mathbf{x}^o$ is the vector of coordinates of an arbitrary point *x* with respect to the reference frame *o*, then it can be expressed as

$$\mathbf{x}^o = \mathbf{x}_a^o + \mathbf{R}_a^o \mathbf{x}^a \tag{2.1}$$

where $\mathbf{x}_a^o$ is the vector which describes the origin of frame *a* with respect to the reference frame *o*, and $\mathbf{R}_a^o$ is a rotational matrix of frame *a* with respect to the reference frame *o*. $\mathbf{x}^a$ is the vector of coordinates of the point *x* with respect to frame *a*. It is shown in Figure 2-1.

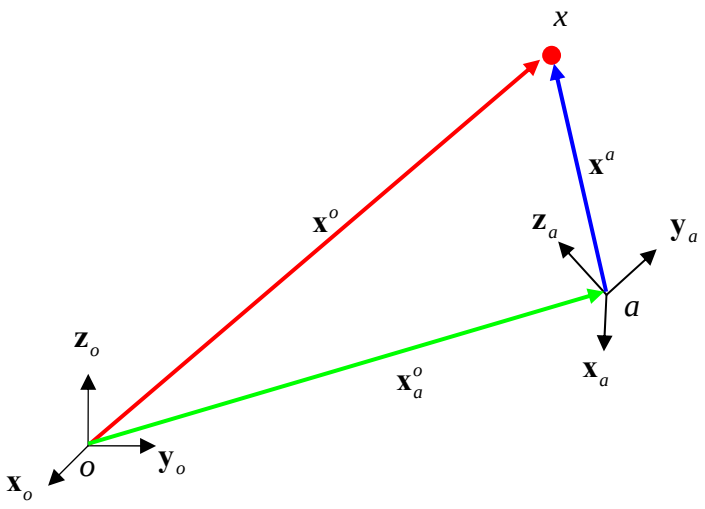

Figure 2-1. Homogeneous transformation

The equation (2.1) can be represented as

$$\tilde{\mathbf{x}}^o = \begin{bmatrix} \mathbf{R}_a^o & \mathbf{x}_a^o \\ \mathbf{0}^T & 1 \end{bmatrix} \tilde{\mathbf{x}}^a \tag{2.2}$$

where the $\tilde{\mathbf{x}}^o = \begin{bmatrix} \mathbf{x}^o \\ 1 \end{bmatrix}$ and the $\tilde{\mathbf{x}}^a = \begin{bmatrix} \mathbf{x}^a \\ 1 \end{bmatrix}$. This representation is called as the homogeneous representation, and the (4x4) matrix is called as homogeneous transformation $\mathbf{A}_a^o$. Therefore, the equation (2.2) can be rewritten as

$$\tilde{\mathbf{x}}^o = \mathbf{A}_a^o \tilde{\mathbf{x}}^a \tag{2.3}$$

For more analysis of the equation (2.1), it is differentiated with respect to time as

$$\dot{\mathbf{x}}^o = \dot{\mathbf{x}}_a^o + \mathbf{R}_a^o \dot{\mathbf{x}}^a + \dot{\mathbf{R}}_a^o \mathbf{x}^a \tag{2.4}$$

This expression can be rewritten with skew-symmetric matrix and constant $\mathbf{x}^a$ as

$$\begin{aligned} \dot{\mathbf{x}}^o &= \dot{\mathbf{x}}_a^o + \mathbf{S}(\boldsymbol{\omega}_a^o)\mathbf{R}_a^0 \mathbf{x}^a \\ &= \dot{\mathbf{x}}_a^o + \boldsymbol{\omega}_a^o \times \mathbf{R}_a^0 \mathbf{x}^a \end{aligned} \tag{2.5}$$

where $\dot{\mathbf{R}} = \mathbf{SR}$ and $\mathbf{S}(\omega)$ is a skew-symmetric matrix in term of angular velocities. This result is useful later for the Jacobian matrix [6, 8], and Denavit-Hartenberg parameters are shown in the next section.

2.3. Denavit-Hartenberg Parameters

From the homogeneous transformation, a general method is required to describe link connections for a specific manipulator. The basic idea is that every manipulator can be described by Denavit-Hartenberg parameters with a kinematic link relationship. The DH parameters are the link length (a), the link twist (α), the link offset (d), and the

joint angles (θ). From the link relationship, the homogeneous transformation matrix is obtained to define the position/orientation of the current joint with respect to the previous joint. If every homogeneous transformation matrix of a manipulator can be defined from the DH parameters, then the position/orientation of the end-effector can be also defined with respect to the reference frame, which is generally its base.

As shown in Figure 2-2, the basic concept of defining the DH parameters is described follows [6~8]. The angle α_i is the distance between o_i and o_i', and the distance d_i is the coordinate of o_i' along axis $\mathbf{z}_{i-1}$ as shown in Figure 2-1. The link twist, α_i, is the angle between axis $\mathbf{z}_{i-1}$ and axis $\mathbf{z}_i$ about axis $\mathbf{x}_i$. The joint angle, θ_i, is the angle between axis $\mathbf{x}_{i-1}$ and axis $\mathbf{x}_i$ about axis $\mathbf{z}_{i-1}$. Counter-clockwise rotation is positive for α_i, and θ_i. The link length and the link twist are always constant at each

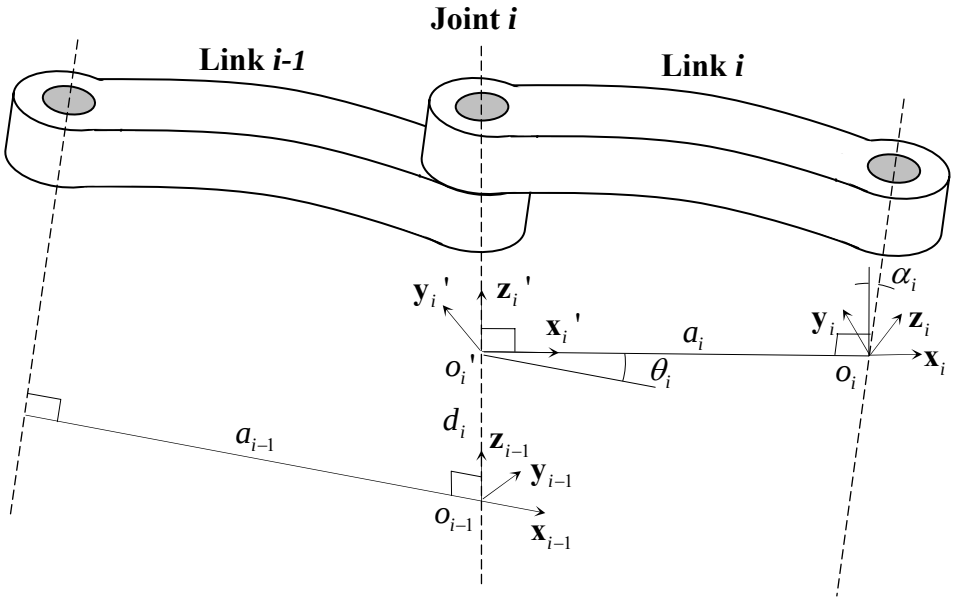

Figure 2-2. DH parameters

joint, but the other two parameters are variable depending on the joint type. If joint is revolute, then θ_i is variable. If joint is prismatic, then d_i is variable. From [8], there are two constrains for uniqueness of the DH parameters. The first is that axis $\mathbf{x}_i$ is perpendicular to axis $\mathbf{z}_{i-1}$, and the second is that axis $\mathbf{x}_i$ intersects axis $\mathbf{z}_{i-1}$. These two constrains give a unique homogeneous transformation matrix.

Form the DH parameters, the final homogeneous transformation matrix between two joints is

$$\mathbf{A}_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Therefore, the forward kinematics is provided by matrix multiplication of the homogeneous transformation calculated by the DH parameters. The coordinate transformation of an *n*-DOF manipulator is given by

$$\mathbf{T}_n^0 = \mathbf{A}_1^0\mathbf{A}_2^1\mathbf{A}_3^2...\mathbf{A}_n^{n-1} \tag{2.7}$$

where $\mathbf{T}_n^0$ is a (4x4) homogeneous transformation matrix from the base to joint *n*. From the calculation of the coordinate transformation, the forward kinematics can be determined and is described in the next section.

2.4. Forward Kinematics

Forward kinematics determines position and orientation of the end-effector from joint values as a variable parameter. By the above the DH parameters and the

homogeneous transformation matrices, the position and orientation of an *n*-DOF manipulator with respect to the base are expressed by

$$\mathbf{T}_n^0(q) = \begin{bmatrix} \mathbf{R}_n^0(q) & \mathbf{x}_n^0(q) \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.8}$$

where $\mathbf{R}_n^0$ is a rotation matrix and $\mathbf{x}_n^0$ is a position vector. In the rotation matrix, the column vectors should be orthogonal each other, and the three column vectors form the reference frame of the end-effector. The third column vector is in the direction of approach of the end-effector, the second column vector is normal to the third vector in the sliding direction, and the first vector is normal to the other two vectors in right-hand rule [6, 8].

2.4.1. Jacobian matrix

Another method to describe the forward kinematics is using a velocity relationship. This differential or velocity kinematics is presented by Jacobian matrix which is computed by differentiation of the forward kinematics function with respect to the joint variables. The velocity relationship can be written as

$$\dot{\mathbf{x}} = \mathbf{J}(\theta)\dot{\boldsymbol{\theta}} \tag{2.9}$$

where $\dot{\mathbf{x}} = \begin{bmatrix} linear\ velociy \\ angular\ velocity \end{bmatrix}$, $\mathbf{J}(\theta)$ is a $(6 \times m)$ Jacobian matrix in terms of joint angles, and $\dot{\boldsymbol{\theta}}$ is joint velocities. There are two types of Jacobian matrices: geometric and analytical Jacobian. Generally, the Jacobian matrix is considered to be the geometric Jacobian, which is slightly different from the analytical Jacobian. The main difference between these two Jacobian matrices is that they use different velocities of rotation. The

rotational velocities, the Euler angles for instance, rather than the angular velocities, are considered in the analytical Jacobian matrix. The analytical Jacobian is defined and the relationship between the geometric and analytical Jacobian matrices is described in [6, 8]. The Jacobian can be divided by two parts. The first three rows of the Jacobian matrix are related to linear velocity. The last three rows are related to angular velocity. Therefore, the Jacobian matrix can be rewritten as

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_v \\ \mathbf{J}_\omega \end{bmatrix} \tag{2.10}$$

In order to compute the Jacobian, it is necessary to distinguish each joint as prismatic or revolute. In the case of a prismatic joint *i*, the Jacobian matrix can be written as

$$\begin{bmatrix} \mathbf{J}_{v,i} \\ \mathbf{J}_{\omega,i} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix} \tag{2.11}$$

and in case of the revolute joint *i*, it can be written as

$$\begin{bmatrix} \mathbf{J}_{v,i} \\ \mathbf{J}_{\omega,i} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{x} - \mathbf{x}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix} \tag{2.12}$$

where $\mathbf{z}_{i-1}$ is the rotation axis of joint i, $\mathbf{x}$ is the position vector of end-effector with respect to the base, and $\mathbf{x}_{i-1}$ is the position vector of joint i with respect to the base. The axis $\mathbf{z}_{i-1}$ can be obtained from the third column of the rotation matrix of joint *i* with respect to the base frame. The vectors $\mathbf{x}$ and $\mathbf{x}_{i-1}$ can be obtained from the transformation matrix [6~8]. Inverse kinematics, as a more complex problem than the forward kinematics, is analyzed in the next section.

2.5. Inverse Kinematics

Inverse kinematics determines joint values from position and orientation of the end-effector as a variable parameter. Inverse kinematics is a useful method for commanding position and orientation or for teleoperation between different DOF manipulators. In addition, because controlling a manipulator is naturally executed in joint space, inverse kinematics is used in controls [9]. However, inverse kinematics is not as simple as the forward kinematics. Its solution may not have a closed form. Therefore, only simple manipulator geometries allow analytical inverse kinematics solutions to be computed. Furthermore, multiple solutions or infinite solutions may exist because of the nonlinear characteristics. The typical example of inverse kinematics with multiple solutions is an elbow up/down position in a planar manipulator which has three revolute joints. At a certain given position and orientation in operational space (or Cartesian space), the inverse kinematics of the manipulator determines two solutions: the elbow up position and the elbow down position. In this case, the more degrees of freedom a manipulator has, the more solutions it has. In order to overcome this problem, it is necessary to analyze manipulators in motion. Therefore, differential or velocity inverse kinematics with the Jacobian is required. If position and angular velocity are used as variables instead of position and orientation, static positioning is possible, where the Jacobian matrix has full rank. However, at certain points, the Jacobian matrix is not invertible. These points are called as singularities. When the Jacobian is rank deficient, one or more degrees of freedom of the manipulator are lost. Therefore, its end-effector can move only in a certain linear or angular directions. However, in real operation of a

manipulator, singularities can be avoided by the operator or software by avoiding configurations where the links are aligned. Another problem with inverse kinematics is approaching singularities. If an end-effector is moved close to its singularity, the Jacobian matrix still has full rank, but its condition number becomes a high number. Therefore, from the small velocity inputs, the large output is produced. In this case, the manipulator has jumps or the controller shuts down the operation because it cannot drive beyond the capabilities of the actuators. Therefore, it is also important to avoid this problem, but it is not easy. To avoid this, Principal Component Regression (PCR) or Damped Least Square (DLS) is used. To invert the Jacobian matrix, a pseudo-inverse method is used. Even though the end-effector reaches singularities, the solution of the inverse kinematics is attained from the pseudo-inverse method. If a manipulator is redundant, then its Jacobian matrix is not invertible because it is not a square matrix. Therefore, the pseudo-inverse is required to solve the inverse kinematics of a redundant manipulator. Singular Value Decomposition (SVD) is another method that is used to perform the pseudo-inverse and the PCR [22, 23].

2.5.1. Pseudo-inverse method

The pseudo-inverse is a common way to find the solution for an inverse problem. The equation (2.9) can be written as

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{\dagger}\dot{\mathbf{x}} \tag{2.13}$$

where the matrix $\mathbf{J}^{\dagger}$ is the pseudo-inverse, and this matrix is unique. The pseudo-inverse has the following properties. If the Jacobian matrix is square and full rank, then $\mathbf{J}^{\dagger} = \mathbf{J}^{-1}$.

If the Jacobian matrix is not full rank, then two types of the pseudo-inverse can be considered. For the first type, the Jacobian matrix has more rows than columns. In this case, a manipulator is kinematically insufficient, and there are more constraints than joint velocity variables. Therefore, normally no solution exists. The solution of the pseudo-inverse minimizes $\left\|\dot{\mathbf{x}} - \mathbf{J}\dot{\boldsymbol{\theta}}\right\|$, and gives the closest to the desired solution, which is a least square method. The second type is that the Jacobian matrix has more columns than rows. In this case, the manipulator is kinematically redundant, and there are less constraints than joint velocity variables. Therefore, generally infinite solutions exist. The pseudo-inverse minimizes the norm of $\left\|\dot{\boldsymbol{\theta}}\right\|$ in this case, and its solution is the particular solution [10]. If the Jacobian matrix is full rank, the pseudo-inverse can be calculated as

$$\mathbf{J}^{\dagger} = \mathbf{J}^{T}(\mathbf{J}\,\mathbf{J}^{T})^{-1}, \tag{2.14}$$

and it is called as the right pseudo-inverse. The pseudo-inverse can be generally calculated by the Singular Value Decomposition method explained below.

2.5.2. Singular value decomposition

Singular value decomposition (SVD) is a powerful tool for computing and analyzing the pseudo-inverse and damped least squares methods. Let $\mathbf{J}$ be the Jacobian matrix. A singular value decomposition of the Jacobian matrix consists of expressing $\mathbf{J}$ in the form [21]

$$\mathbf{J} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{T} \tag{2.15}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{\Sigma}$ is a diagonal matrix. If $\mathbf{J}$ is $(m \times n)$, then $\mathbf{U}$ is $(m \times m)$, $\mathbf{\Sigma}$ is $(m \times n)$, and $\mathbf{V}$ is $(m \times n)$. The singular value decomposition of any matrix exists even if the matrix is not full rank. The pseudo inverse of $\mathbf{J}$ is equal to

$$\mathbf{J}^{\dagger} = \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{U}^{T} \tag{2.16}$$

where $\mathbf{\Sigma}^{\dagger}$ is the inverse of a diagonal matrix whose diagonal entries are non-zero, and it takes the form

$$\mathbf{\Sigma}^{\dagger} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_r} \end{bmatrix} \tag{2.17}$$

If any of the singular values is zero, then a zero is replaced in the corresponding entry of $\mathbf{\Sigma}^{\dagger}$. If the Jacobian matrix is not full rank like the above first case, then one or more singular values will be zero. Finally the equation (2.16) can be rewritten as

$$\mathbf{J}^{\dagger} = \sum_{i=1}^{r} \sigma_i^{-1} \mathbf{v}_i \mathbf{u}_i^{T} \tag{2.18}$$

where $\mathbf{v}_i$ is the *ith* column vector of $\mathbf{V}$, $\mathbf{u}_i$ is the *ith* column vector of $\mathbf{U}$, r is the rank of the Jacobian matrix. From the maximum singular value and the minimum singular values, the condition number can be calculated as

$$Con(\mathbf{J}) = \frac{\sigma_{\max}}{\sigma_{\min}}. \tag{2.19}$$

If condition number is high, such that the Jacobian matrix is ill conditioned, then the Principal Component Regression (PCR) can be performed to reduce sensitivity of the

Jacobian matrix. The PCR method performs SVD of the Jacobian, and it examines its condition number. If the condition number is higher than a certain number, which is depends on the system, then its weak or smallest singular value is replaced with zero to eliminate the least square problem.

2.5.3. Damped Least Square (DLS)

The DLS solves inverse kinematics problems when target positions are near a singularity area or unreachable area. In this situation, it is not easy to handle robustly, and normal inverse methods will oscillate badly because the Jacobian is very sensitive to small changes in joint angles. The PCR method also can solve the singularity problem, but the DLS solves a discontinuity of the PCR when the condition number is high. The method performs the inverse kinematics, and optimizes the joint velocities to give the minimum position tracking error. The DLS is defined as follows [10, 11]. The joint velocities $\dot{\boldsymbol{\theta}}$ are found to minimize this value

$$\left\|\dot{\mathbf{x}} - \mathbf{J}\dot{\boldsymbol{\theta}}\right\|^2 + \lambda^2 \left\|\dot{\boldsymbol{\theta}}\right\|^2 \tag{2.20}$$

where λ is the damping constant, which is not zero, and the first term shows the actual error. The equation is equivalent to

$$\left\| \begin{pmatrix} \mathbf{J} \\ \lambda\mathbf{I} \end{pmatrix} \dot{\boldsymbol{\theta}} - \begin{pmatrix} \dot{\mathbf{x}} \\ \mathbf{0} \end{pmatrix} \right\| \tag{2.21}$$

The normal of this equation is

$$\begin{pmatrix} \mathbf{J} \\ \lambda\mathbf{I} \end{pmatrix}^T \begin{pmatrix} \mathbf{J} \\ \lambda\mathbf{I} \end{pmatrix} \dot{\boldsymbol{\theta}} = \begin{pmatrix} \mathbf{J} \\ \lambda\mathbf{I} \end{pmatrix}^T \begin{pmatrix} \dot{\mathbf{x}} \\ \mathbf{0} \end{pmatrix} \tag{2.22}$$

Therefore, this can be rewritten as

$$\left(\mathbf{J}^T\mathbf{J} + \lambda^2\mathbf{I}\right)\dot{\boldsymbol{\theta}} = \mathbf{J}^T\dot{\mathbf{x}} \tag{2.23}$$

Because the $\mathbf{J}^T\mathbf{J}$ is cannot be singular, the damped least square solution is

$$\dot{\boldsymbol{\theta}} = (\mathbf{J}^T\mathbf{J} + \lambda^2\mathbf{I})^{-1}\mathbf{J}^T\dot{\mathbf{x}} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda^2\mathbf{I})^{-1}\dot{\mathbf{x}} \tag{2.24}$$

From the above equation, $(\mathbf{J}^T\mathbf{J} + \lambda^2\mathbf{I})^{-1}\mathbf{J}^T$ and $\mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda^2\mathbf{I})^{-1}$ are identical. However, the size of $\mathbf{J}^T\mathbf{J}$ is larger than $\mathbf{J}\mathbf{J}^T$ if the Jacobian matrix is a redundant manipulator. This equation can be rewritten by SVD as

$$\mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda^2\mathbf{I})^{-1} = (\mathbf{V}\boldsymbol{\Sigma}^T(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T + \lambda^2\mathbf{I})^{-1}\mathbf{U}^T) = \sum_{i=1}^{r}\frac{\sigma_i}{\sigma_i^2 + \lambda^2}\mathbf{v}_i\mathbf{u}_i^T \tag{2.25}$$

From the equation, equation (2.18) and (2.25), the only difference is the singular value term, which is commonly called as a filter factor. If the singular value σ is much larger than the damping constant λ, then the DLS method is identical to the pseudo-inverse. However, if the singular value is smaller, then the singular value term gradually goes to zero as σ goes to zero. Therefore, the DLS method acts like the pseudo-inverse method if an end-effector is away from singularities, and acts like PCR if an end-effector is close to singularities. In Chapter 3, the WAM-Titan II teleoperation system is introduced, and the specifications are presented. Moreover, the architecture of the teleoperation system is discussed.

CHAPTER 3:WAM-Titan II Teleoperation System

In teleoperation, tasks are performed by a slave manipulator while controlled by a master manipulator remotely. Controlling the slave manipulator by the master manipulator can be achieved by either joint-to-joint control or Cartesian space control. In this chapter, the master and slave manipulators are introduced, and the issues with the kinematic coordination of the end-effector in Cartesian space are investigated.

3.1. Whole-Arm-Manipulation (WAM)

The master manipulator is a 7-DOF WAM which is consisted of 4-DOF arm and an attached 3-DOF gimbals as shown in Figure 3-1. Every joint is a revolute joint, and the first four joints are driven by cables. Since this robot arm is cable-driven, it does not use any gears for manipulating the joints. The gears are replaced by a cable drive, eliminating any backlash problems and allowing for improved speed and stiffness. The gimbals are attached to the end of the arm, and three potentiometers are put together to measure each joint angle. However, these joints are not drivable. The gimbals' three revolute joint axes intersect at the middle of a handle. This spherical wrist can decouple the position and orientation of the end-effector, which is the handle of the WAM arm. Because it is a 7-DOF manipulator, the WAM arm is a kinematically redundant manipulator, and this gives a benefit of more dexterous motion of the arm. As a master

Figure 3-1. WAM and Titan II

arm, the redundancy provides more controllable handling to the operator. The main features of the WAM arm are back-drivability and gravity compensation. The back-drivability is given by its advanced cable-drive system. Unlike other manipulators, for example the Titan II, it can be operated passively by an operator. The gravity compensation feature assists the operator to reduce fatigue. Gravity affects the whole arm, including the joints and links, and it causes them to drop under the arm's own weight. The gravity compensator is able to balance the effects of gravity, so that the master manipulator feels weightless to the operator during operation.

As mentioned previously, the WAM is a redundant manipulator, meaning that it has more degrees of freedom than the number of degrees of freedom required to define the position and orientation of the end-effector. In general, six degrees of freedom are required in 3D space. The extra degree of freedom of the WAM manipulator can by effectively used to improve the ability of the manipulator. On the other hand, this redundant manipulator is more complex to control than a non-redundant manipulator. Mapping between operational space and joint space has always been difficult for redundant manipulators because of the presence of a non-square Jacobian matrix. Since a direct pseudo-inverse does not work at all times, an additional optimization algorithm is required along with the Jacobian calculation. Namely, some constraints need to be implemented, which would make the Jacobian square and non-singular. Since the WAM has seven degrees of freedom, the redundant manipulator does not have a unique solution. Therefore, functional constraints like joint limit avoidance and obstacle avoidance constraints can be adapted to solve the redundancy.

The WAM comes with a PC with a Linux operating system, which is patched by the Real-Time Application Interface (RTAI). The PC includes the WAM controller software. This performs a 500Hz position/torque control closed-loop over CAN bus between the WAM and the PC, where the final controller commands to the WAM are the motor torques. Therefore, the WAM itself is entirely motor torque controlled [12]. The arm's weight ranges from 25.4 to 27.2 kg, and its payload varies from 3 to 4.5 kg depending on the configuration.

3.1.1. DH parameters and joint ranges for WAM

The WAM can be divided by three parts: shoulder, elbow, and wrist. This structure of the WAM is similar to a human arm. The first three joints intersect each other at the base frame, which is the WAM reference frame. The fourth joint is the elbow, and the last three joints also intersect each other at the middle of the handle. Therefore, the end-effector is located at the handle. This configuration and the dimensions are shown in Figure 3-2. The DH parameters for the WAM can be defined from the configuration and dimensions as shown in Table 3-1.

From the DH parameters, the homogeneous transformation matrix can be generated by the equation (2.6) and (2.7). Using the parameters in the Table 3-1, the transformation matrix from the base frame to frame 1 is

$$\mathbf{A}_1^0 = \begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.1}$$

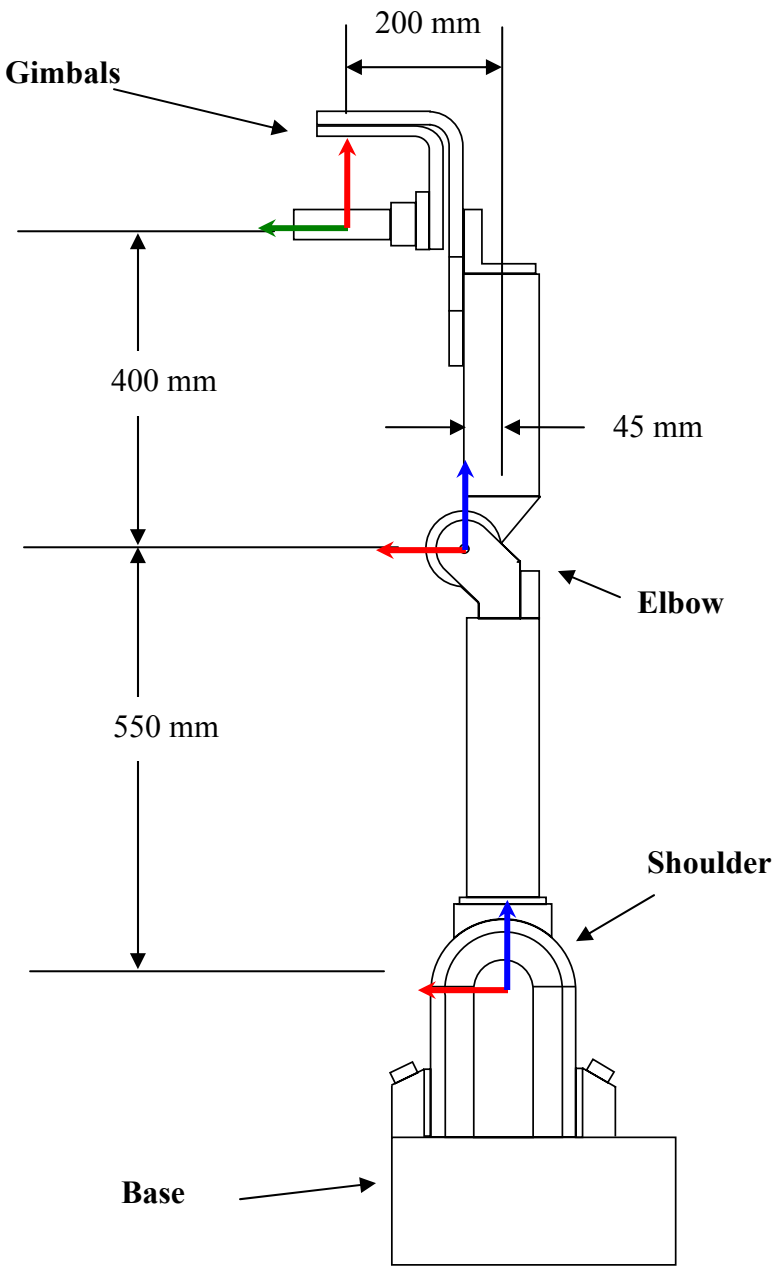

Figure 3-2. WAM DH parameters

Table 3-1. DH Parameters for WAM

i	a_i (m)	α_i (rad)	d_i (m)	θ_i (rad)
1	0	$-\frac{\pi}{2}$	0	θ_1
2	0	$\frac{\pi}{2}$	0	θ_2
3	0.045	$-\frac{\pi}{2}$	0.55	θ_3
4	0.4	$-\frac{\pi}{2}$	0	θ_4
5	0	$\frac{\pi}{2}$	0	$\theta_5 - \frac{\pi}{2}$
6	0	$-\frac{\pi}{2}$	0.1514	$\theta_6 - \frac{\pi}{2}$
7	0	0	0	θ_7

and other transformation matrices are shown in the Appendix A. Therefore, the final homogeneous transformation matrix is

$$\mathbf{T}_7^0 = \mathbf{A}_1^0 \mathbf{A}_2^1 \mathbf{A}_3^2 \mathbf{A}_4^3 \mathbf{A}_5^4 \mathbf{A}_6^5 \mathbf{A}_7^6. \tag{3.2}$$

The calculation of Jacobian matrix is based on each of the homogeneous transformation matrix relations. The joint ranges of the WAM are shown in Table 3-2. In the next section, the Titan II manipulator is discussed in detail.

3.2. Titan II

The Schilling Titan II is a 6-DOF manipulator and a slave in the WAM-Titan II teleoperation system. Its material is titanium, so that this hydraulic manipulator is durable and versatile from precise operations to heavy duty industrial operations. Because it is

Table 3-2. Joint Ranges

Joint	Minimum	Maximum
1	$-150°(-2.6\,rad)$	$150°(2.6\,rad)$
2	$-113°(-2.0\,rad)$	$113°(2.0\,rad)$
3	$-157°(-2.8\,rad)$	$157°(2.8\,rad)$
4	$-50°(-0.9\,rad)$	$180°(3.1\,rad)$
5	$-160°(-2.8\,rad)$	$160°(2.8\,rad)$
6	$-160°(-2.8\,rad)$	$160°(2.8\,rad)$
7	$-150°(-2.6\,rad)$	$150°(2.6\,rad)$

driven by hydraulic power, it is possible not only to operate under water, but also to handle large payloads. The pressure of oil supplied into the arm is 300 psi, and the maximum lift capacity is 240 lb. The actuators of the Titan II are hydraulic linear or rotary actuators. Each joint has a servo valve to control the oil flow though the joint's chambers and a resolver to measure joint angles.

The Titan II has six revolute joints: azimuth, shoulder, elbow, pitch, yaw, and roll. The azimuth joint contains a hydraulic rotary vane actuator which rotates the arm horizontally. The shoulder joint has a linear actuator which is connected between the azimuth and the first link. It moves the arm vertically. The third joint, elbow, has the same type of hydraulic actuator as the azimuth. This joint moves the second link up and down. The pitch and yaw joints also have hydraulic rotary vane actuators, but they are located at $90°$ from each other. The pitch joint moves vertically, and the yaw joint moves

horizontally. The last joint has a hydraulic rotary vane actuator, which can continuously rotate in both directions.

The hydraulically actuated slave arm originally utilized a master controller called a Mini-Master. This small controller is kinematically similar to the slave arm, so that an operator controls the slave arm by joint-to-joint operation. It has a control panel and sends signals between the Mini-Master and the Titan II via RS232 serial communication. These signals activate a solenoid valve, and drive the hydraulic actuator. For this teleoperation system, this master is only used for diagnostic operation.

For the WAM-Titan teleoperation, PC/104, which is a small embedded computer, is used to host for a low level control. The low level controller is provided by Oak Ridge National Laboratory (ORNL), and it communicates with the Titan II via Ethernet. The low lever control is developed by QNX, which is a real time operating system, and closes 200Hz control loop with the Titan II. The controller receives joint angle commands, and executes low level servo control of the actuators.

3.2.1. DH parameters and joint ranges for Titan II

The dimensional and configuration diagram is shown in Figure 3-3. Each joint is revolute, and the base frame is located on the bottom of the azimuth. The shoulder joint connects the azimuth and the upper arm, but the joints do not intersect each other. Furthermore, the last three joints, pitch, yaw, and wrist, also do not intersect, so they do not form a spherical wrist. One reason for this configuration is that the Titan II is designed for joint-to-joint teleoperation. The DH parameters for the Titan II can be defined from the configuration as shown in Table 3-3.

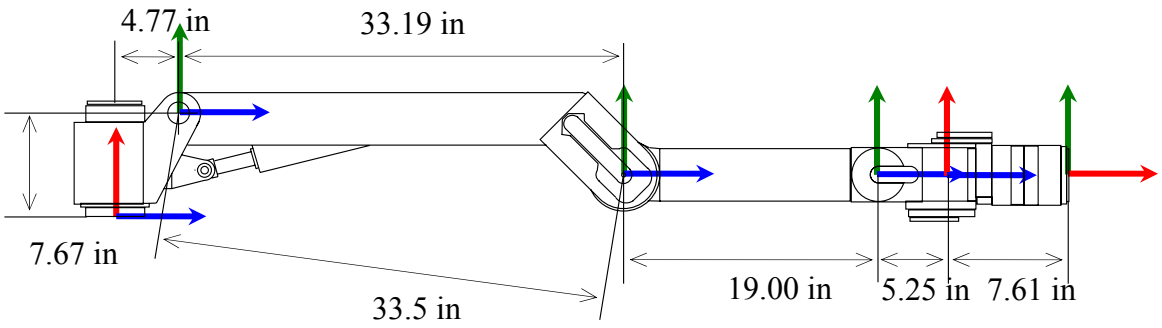

Figure 3-3. DH parameters for Schilling Titan II

Table 3-3. DH Parameters for Titan II

i	a_i (inch)	α_i (rad)	d_i (inch)	θ_i (rad)
1	4.77	$\frac{\pi}{2}$	7.67	θ_1
2	33.5	0	0	θ_2
3	19	0	0	θ_3
4	5.25	$-\frac{\pi}{2}$	0	θ_4
5	0	$\frac{\pi}{2}$	0	$\theta_5+\frac{\pi}{2}$
6	0	0	7.61	θ_6

From the DH parameters, the homogeneous transformation matrix is calculated like the WAM transformation matrix. Using the parameters in the Table 3-3, the transformation matrix from the base frame to frame 1 is

$$\mathbf{A}_1^0 = \begin{bmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & a_1\cos(\theta_1) \\ \sin(\theta_1) & 0 & -\cos(\theta_1) & a_1\sin(\theta_1) \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.3}$$

and other transformation matrices are shown in Appendix B. Therefore, the final homogeneous transformation matrix is

$$\mathbf{T}_6^0 = \mathbf{A}_1^0\,\mathbf{A}_2^1\,\mathbf{A}_3^2\,\mathbf{A}_4^3\,\mathbf{A}_5^4\,\mathbf{A}_6^5 . \tag{3.4}$$

The joint ranges of the Titan II are shown in Table 3-4. In the next section, the architecture of the WAM-Titan II teleoperation system is described.

Table 3-4. Joint Ranges for Titan II

Joint	Minimum	Maximum
1	$-135°\,(-2.4\,rad)$	$135°\,(2.4\,rad)$
2	$-41.4°\,(-0.7\,rad)$	$78.8°\,(1.4\,rad)$
3	$-180°\,(-3.1\,rad)$	$90°\,(1.6\,rad)$
4	$-90°\,(-1.6\,rad)$	$90°\,(1.6\,rad)$
5	$-90°\,(-1.6\,rad)$	$90°\,(1.6\,rad)$
6	*continuous*	*continuous*

3.3. Teleoperation

The current WAM-Titan II teleoperation system is a unilateral teleoperation system. This means that there is no force feedback between the master controller and the slave manipulator. The WAM manipulator has capabilities for force feedback, and the Titan II manipulator can provide end-effector force data, but the current system is not fully developed, and does not use these features yet.

The existing teleoperation system between the WAM master and the Titan II slave mainly runs on three computers: two for the low level controller of each manipulator and one High Level Controller (HLC) to perform the coordinate mapping. The three computers are connected by a local area network (LAN). Other computers are used for simulating the slave manipulator using a commercial program, RoboWorks, developing the Titan II low level software, and controlling with the HLC program remotely.

The HLC software is designed by REMSL, and the main function of this controller is to perform Cartesian space mapping and to improve the control strategy. The high level control software has been developed primarily to perform 1) the WAM forward kinematics, 2) the Titan II inverse kinematics, and 3) the Ethernet communication for the WAM PC, the PC/104, and the other computers. The simulation PC operates under a Windows OS and is used to simulate the Titan II using RoboWorks. There are two operation modes. The first is a simulation without actual operation of the Titan II. The second mode is used for monitoring the slave during actual operation. Therefore, an operator can easily test and observe the status of the Titan by its 3D model.

The development PC is based on QNX, which is a real-time OS and is used to program and upload to the PC/104 through the network. To control with the high level controller remotely, a touch screen PC is used. The PC is mounted in the Compact Remote Console (CRC) unit. The CRC was manufactured by Agile Engineering, Inc., and provides an ergonomic teleoperation workstation for viewing and controlling manipulators. A Windows/C++ based GUI was developed in order to select the control mode for the teleoperation system, and also to control the camera displays on the CRC monitors. The GUI is displayed on the touch screen computer in the CRC, along with the GUI for the Wraptor controls. The GUI is used to control the teleoperation system remotely. This interface allows the operator select from several operation modes, such as the orientation modes, the Cartesian position mode, etc. The interface is based on Ethernet TCP/IP communication between two different operating systems: the HLC uses Linux and the touch screen uses Windows 2000. This interface can also accept keyboard inputs to the HLC PC. The keyboard inputs have higher priority than that of the touch screen inputs.

The CRC video devices are a Pelco MX4000 Multiplexer and a CM6700 Switcher. The CRC has the capability to connect with up to eight cameras and display on four monitors. The Pelco Multiplexer can display a group of four or nine cameras on a single monitor, as well as display a picture-in-picture. Therefore, an operator sitting on the CRC can watch eight camera views at once and can change the monitor views at any time during operation. The communication between the interface and the video devices uses RS-232 with a Pelco ASCII protocol.

The main architecture of the WAM-Titan II teleoperation system involves the forward kinematics transformation of the WAM master joint angles to its end-effector

positions. At this phase, coordinate mapping is performed from the velocities of the WAM to the velocities of the Titan II. The Titan II inverse kinematics then transforms the end-effector position of the WAM to joint angles of the Titan II. For these algorithms of the HLC, in the first step, the WAM low level control program sends the measured joint angles of the WAM with time steps to the HLC, via Ethernet. The Titan II low level controller also sends current Titan II joint angles to the HLC via Ethernet. The HLC calculates the homogeneous transformation matrices and Jacobian matrices for the WAM and the Titan II. The linear and angular velocities of the WAM end-effector are generated by the forward kinematics, based on the WAM Jacobian. To execute the Titan II inverse kinematics, the HLC performs the SVD and calculates the pseudo-inverse. The joint velocities of the Titan II are calculated by the inverse kinematics with the Cartesian velocities of the WAM and the time steps. The HLC sends the joint velocities to the Titan low level controller, and the controller sends signals to the Titan hydraulic actuators, which move the joints. These steps comprise one cycle of the teleoperation control loop.

In Chapter 4, brief reviews of fuzzy logic and artificial neural network are presented. For fuzzy logic, ANFIS and GA are introduced, and for artificial neural networks, multilayer perceptrons, RBF and GRNN are presented.

CHAPTER 4:Fuzzy Logic and Artificial Neural Networks

4.1. Fuzzy Logic

4.1.1. Overview

Much recent research has focused on development for precision and accuracy, and the concept of fuzziness has been rejected by many scientists. However, this attitude does not reflect the human natural reasoning process. Humans use ambiguous linguistic meanings, and these meanings provide a big picture without a series of complex mathematical data which have unnecessary details. Therefore, fuzzy logic gives this conventional way of natural language to understand and to infer uncertain facts and their relationships. Fuzzy set theory was developed by Zadeh first in 1965, and the first fuzzy inference system was proposed by Mamdani in 1974 [28, 29]. Fuzzy logic uses this concept with if-then rules to interpret and apply humans' expert knowledge. Fuzzy logic is useful where [30]

1) Mathematical models are difficult to specify.
2) Rules which express knowledge and facts are linguistic in nature.
3) Classes of objects are more fuzzy than crisp in categorical data analysis.
4) Observations are expressed in linguistic terms to implement human control strategies in robotics.

Therefore, the fuzzy logic is another intelligent tool to understand and express a non-linear or complex system without massive mathematical forms. However, a disadvantage of fuzzy logic is the lack of a formal procedure to describe the fuzzy sets and membership functions of the control rules. Furthermore, for an unknown system without expert human knowledge, it is difficult to define rules with an ordinary fuzzy logic method.

The fuzzy logic concept uses mapping between input space and output space like other artificial intelligence methods. The input and output are described by if-then rules involving linguistic variables. The fuzzy rules may be derived from a mathematical model, expert knowledge, or an algorithm which automatically generates the fuzzy model. One of the significant differences of fuzzy logic is the fuzzy set and membership function (MF) represented by membership values. The fuzzy set contains elements with a partial degree of membership, unlike a crisp set, and the membership function maps the values of the universe of discourse onto the degrees of membership between 0 and 1 [4, 13].

The fuzzy inference system consists of five steps as shown in Figure 4-1. These process steps are explained in [5, 24, 30], and they are

Step 1: Fuzzifying input variables using membership functions.

In this step, the inputs are taken, and determined by the degree where they belong to each of the appropriate fuzzy sets via membership functions. The inputs are always crisp numerical values limited to the universe of discourse of the input variables. The output is a fuzzy degree of membership in the qualifying linguistic set, and always the interval between 0 and 1. The mapping can be written as

$$\mu : X \rightarrow [0,1] \tag{4.1}$$

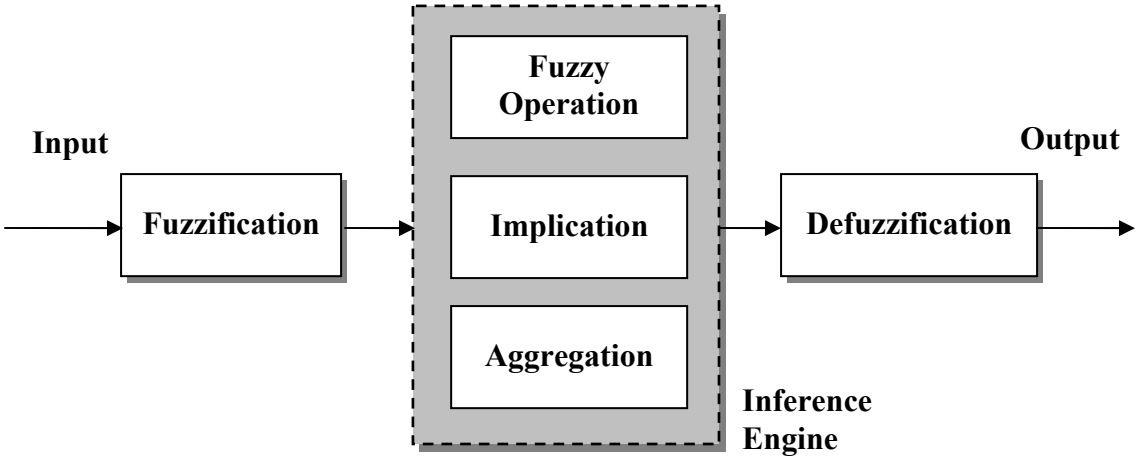

Figure 4-1. Fuzzy inference system

Step2: Applying the fuzzy operator.

The fuzzy operator is applied to resolve the antecedent if there are multiple inputs. For instance, the if-then rules are formed as

$$If \ \ (input_1 = x) \ \ AND \ \ (input_2 = y) \ \ THEN \ \ (output = z), \tag{4.2}$$

where $input_1$, $input_2$, and *output* represent fuzzy variables, and x, y and z are fuzzy values. There are two parts of the antecedent, and in this case, two parts of the antecedent are calculated together and produce a single number using the logical operators like *AND* or *OR* operation. The *AND* operator is usually *min* (minimum) or *prod* (product). The OR operator is either the *max* (maximum) or the *probor* (probabilistic OR), which is the algebraic $sum(a,b) = a + b - ab$.

Step3: Applying the implication method.

The implication reshapes and evaluates the consequent of the rule using the result of the antecedent of the rule. The input of implication is one single value and the output of implication is fuzzy sets. There are two commonly used methods: the *AND* method and the *prod* method. The *AND* method, which is *min* (minimum), truncates the output fuzzy set, and *prod* (product) scales the output fuzzy set.

Step 4: Aggregating all outputs.

Aggregation combines the fuzzy outputs, which represent the outputs of each rule, to result in a final fuzzy output set. The inputs of the aggregation process are the list of truncated or scaled output membership functions, and the output is one fuzzy set for each output variable. Commonly used methods are *Max* (maximum), *probor* (probabilistic OR), and *sum* (simply the sum of each rule's output set). An example of the fuzzy operation, the implication, and the aggregation is shown in Figure 4-2.

Step 5: Defuzzifying.

In this step, the fuzzy output set is converted to crisp output. There are several methods for defuzzification, such as (1) *centroid*, which finds the geometric center of area or gravity of the fuzzy set, (2) *bisector*, (3) *middle of maximum*, which is the average of the maximum value of the output set, (4) *largest of maximum*, and (5) *smallest of maximum*.

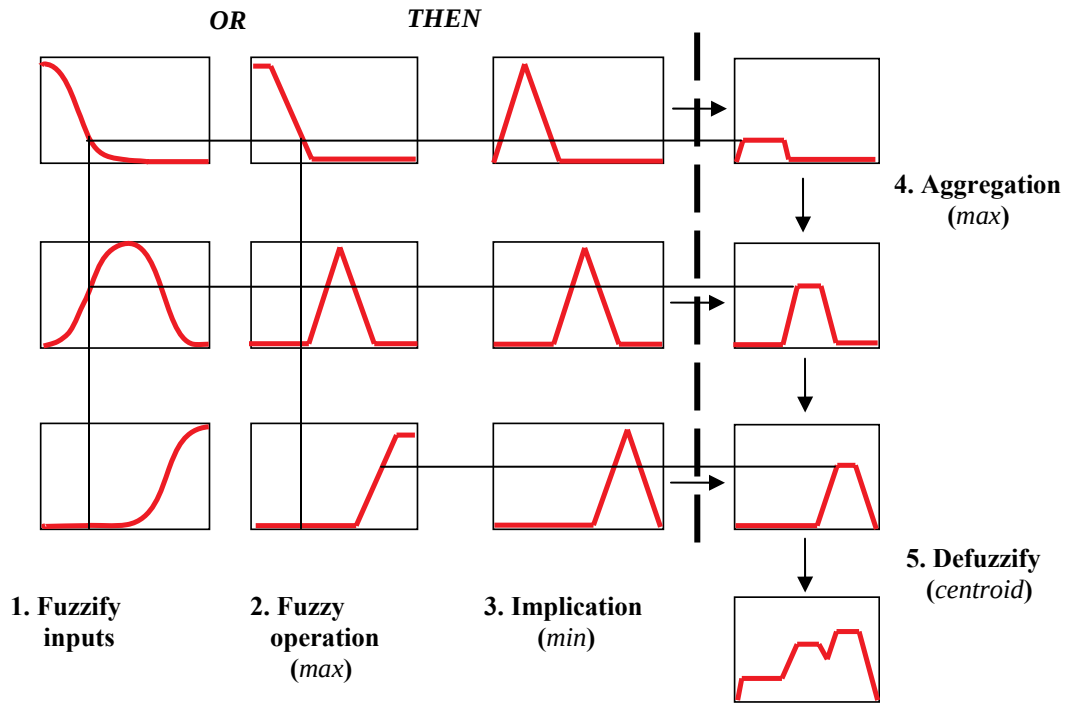

Figure 4-2. Example of fuzzy inference engine

4.1.2. Adaptive Neuro-Fuzzy Inference System (ANFIS)

ANFIS is a hybrid system with the best aspects of fuzzy logic and artificial neural networks, and was originally proposed by Jang in 1993 [31]. From the fuzzy logic, ANFIS represents past knowledge in a set of constrains to reduce the optimization process of the artificial neural network. From the artificial neural network, it adapts backpropagation to tune fuzzy logic parameters automatically to the network. Therefore, the model of ANFIS can be explained by past data and predicted for future behaviors. ANFIS has proven to be an excellent function approximation tool. It implements first or zeroth order Sugeno-type systems where output membership functions are either linear or

constant. The ANFIS constructs a fuzzy inference system, while membership parameters are adjusted by a backpropagation algorithm. Thus, before the optimization processes for the membership parameters, input and output data sets are required to be given [5].

There are some constrains of ANFIS because of its complexity, and the constraints are mainly associated with the Sugeno-type systems. The output membership function should be linear or constant. Therefore, the Sugeno-type system is either first order or zeroth order. The output of ANFIS is obtained by weighted average defuzzification, and the output should be single. Another constraint is that one rule connects only one output membership function. Rules cannot be shared. Therefore, the number of output membership functions is equal to the number of rules. The last constraint is that each rule has a single weight which is updated from backpropagation [4].

An example of an ANFIS structure is shown in Figure 4-3. There are two inputs and one output. The first input has three membership functions, and the other input has fifteen membership functions. The membership function can be any type, but it should be same for every input. The number of rules is equal to the number of all possible cases, which is fourteen. Each node in the first layer generates the membership of inputs. The second layer implements the fuzzy AND operator, and calculates the firing strength of each rule. The third layer calculates the ratio of the rule's firing strength to the sum of all the firing strengths at each node. The fourth layer generates the output level with the consequent parameters, and the output level is multiplied by the output of the third layer. The last layer aggregates the overall output by summing all the outputs of the fourth layer [4, 31, 33]. The tuning of adjustable parameters is a two-step procedure [31, 33]. In the first step, information is estimated in the network until the third layer, and the parameters

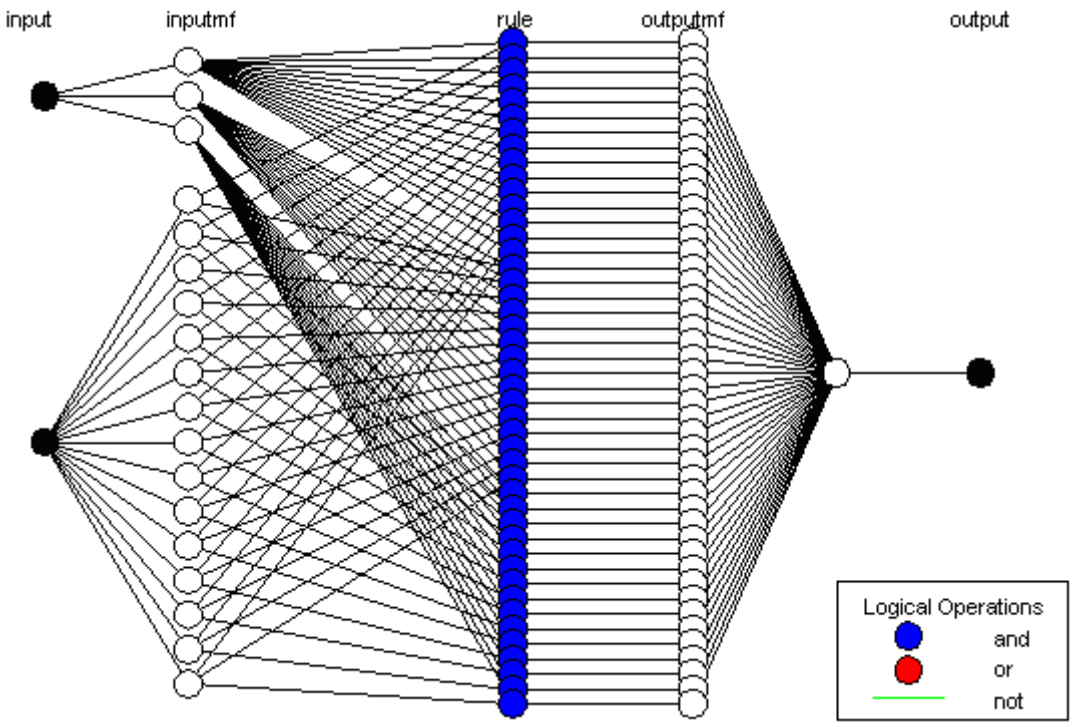

Figure 4-3. ANFIS structure

are determined by a least-squares method. Subsequently the parameters in the input membership functions are altered using the gradient descent method.

4.1.3. Genetic Algorithms (GA)

Genetic Algorithms are another type of artificial intelligence method, which is reliable and robust for optimizing and solving solutions. GA were proposed by Holland in 1975 [32]. This method originated from the biological concept of genetics and evolution theory [14]. As regarding that the most complete controllers are the human brain and other several astonishing natural controllers, natural selection, which is the process of evolution, is a highly successful design procedure. Genetic Algorithms use this scheme to

process and expand complex problems in which parameters interact. Individuals characterize potential solutions, and they are selected according to their fitness. They pass on their characteristics to following generations. Mating takes place between these individuals with sharing the characteristics of winning individuals, even fitter individuals can be created. Mutation also occurs to add new genes into a population. As in nature, most mutations are bad, but the infrequent valuable one can help improve the fitness of the individuals finding a better solution.

The basic ideas of Genetic Algorithms are chromosomes, population, inheritance, mutation, selection, and crossover. The chromosome is an encoded string map to be optimized. It can hold a float value or binary value. The population is a group of individuals. Each individual is evaluated by decoding the chromosome values. After the fitness of each individual is evaluated, the selection procedure is performed. Individuals are selected to create the next generation. The probability of the selection procedure is related to the fitness function. A popular selection algorithm is the Roulette Wheel algorithm. The crossover takes place between pairs of individuals. The strings, which are float values or binary values, are mixed. The most basic crossover algorithm is Single Point Crossover. The mutation changes bits or individuals randomly. This procedure is performed with a low probability. Mutation guarantees that the probability of searching a given part of the solution space is never zero [14]. The basic procedure of genetic algorithms is

1. Randomly generate an initial population.
2. Evaluate all individuals using a fitness function, or evaluation function.
3. Select a new population using a selection algorithm.

4. Perform crossover and mutation.
5. Evaluate the new population using the fitness function.
6. Repeat the above procedure until it reaches a goal error, or reaches the maximum number of a population.

Many researchers have applied GA theory to finding optimized fuzzy inference systems. For example, the GA method is applied to optimize an if-then rule base of fuzzy logic, where the membership functions are already created [15]. In another approach, a GA is used for determining the number of membership functions, the number of fuzzy rules, and the rule base [16]. In a third approach in [17], GA is implemented by characteristic parameters to automate fuzzy logic design. This method is applied in this thesis. In this case, the GA can design fuzzy logic flexibly, with the numbers and positions of membership functions determined by the GA as well as the rule base. In the next section, artificial neural networks are discussed for the last three methods which are MLP, RBF, and GRNN.

4.2. Artificial Neural Networks

4.2.1. Overview

Artificial neural networks are a method of computation and information processing that takes advantage of the structure of human brain. The human brain is embodied with neurons, and these neurons are linked each other through dendrites and axons. Signals transfer by chemical and electrical process in a synapse. The synaptic gap and its adjustment lead to the storage of information or learning [33, 35]. Mimicking these processes of biological neurons, the perceptron which is a mathematical model of

the neuron was developed by Rosenblatt in 1959 [37]. Artificial neural networks are used to predict and learn from a given set of data, and are a mapping process from input space to output space. All inputs are added with weights and biases, and passed to an activation function, which introduces nonlinearity to the network. The activation function, or a transfer function, makes networks capable of representing nonlinear characteristics. As shown in Figure 4-4, hard limit functions, linear functions and sigmoidal functions are common. The output from the activation function can be connected to another neuron's input. The weights and biases are trained to minimize errors between desired outputs and actual outputs. The training is a learning process to update weights and biases. The weights and the biases have important roles in artificial neural networks. The weights determine the position in the input space, and without biases, inputs are constrained to pass through the origin of the input space [33].

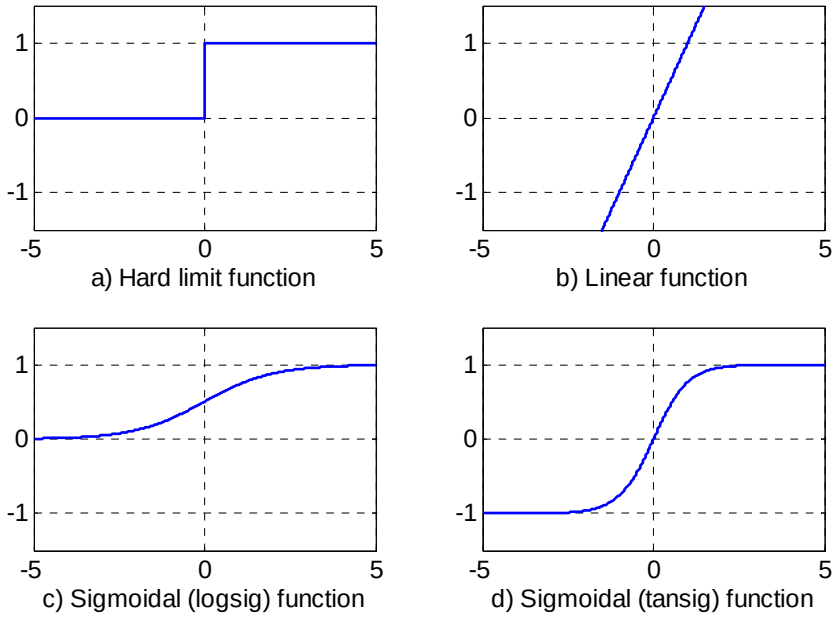

Figure 4-4. Activation functions

Important characteristics of an artificial neural network are the ability to perform nonlinear mapping, and less sensitivity to noise. Because any failure of neurons or weights will slightly affect the performance, it is fault tolerant. Furthermore, it is easily implemented in other systems. However, artificial neural networks have the some drawbacks. They require extensive training data for training, and the training process is very time consuming according to the task. In addition, their characteristics prevent heuristic knowledge, so that the reason why they reach the results can not be explained. [19]

The architecture of neural networks consists of neurons, layers, activation functions, and connections between layers. A single neuron may perform in certain cases, but several neurons are more effective. More than one neuron can be merged together in a layer, and a neural network can have multiple layers. Moreover, a single layer neural network may solve a simple problem, but multiple layers can solve more complex problems. In multiple layers, each neuron in each layer has an individual weight and bias, yet it has a common activation function. Multilayer feedforward networks are commonly used. This network type has three kinds of layers: an input layer, hidden layers, and an output layer. The input layer does not have any neuron which contains a weight, a bias, and an activation function. Therefore, the input layer may not be called a layer. The neurons in the input layer accept inputs and distribute them to a subsequent hidden layer. Each hidden layer produces summations of weighted inputs and biases, and sends them to a transfer function. The result of the transfer function becomes inputs to the next layer. Finally, the last output layer produces outputs. The number of neurons in input and output layers is already decided by the respective problem statement. Therefore, only hidden

layers are considered in designing the architecture of the neural network. The number of hidden layers is not easy to decide, and it depends on training data and complexity of the problem. There is no straightforward rule to determine the number of hidden layers and the nodes in the hidden layer.

4.2.2. Multilayer Perceptrons Network (MLP)

A multilayer perceptrons network is a feedforward network with nonlinear activation functions and a linear output layer. This type of network can approximate any function between input and output association with enough neurons. For a training method, the MLP generally uses backpropagation to optimize network error. The MLP may be trapped in local minima instead of global minima. In this case, the best ways to avoid local minima are to repeat the training until acceptable error is found or to increase the number of neurons. For the basic procedure of the MLP, weights are initialized to random small values and inputs are weighted by these weight matrices, added with the biases, and acted upon by the activation function. The final outputs are compared with the desired output, and the error is calculated. This error is back propagated through the network, and weights and biases are adjusted to minimize the error. This is repeated until the error goal is met. A structure of a basic neuron with multiple inputs is shown in Figure 4-5. It shows the weights *w(n)*, the bias *b*, the summation of weighted incoming signals, and the activation function **F***( )*. The cell inputs are *n* times signals *p,* and the output is the scalar *a*. They can be expressed as

$$a = \mathbf{F}(\sum_{j=1}^{n} w_j p_j + b) \tag{4.3}$$

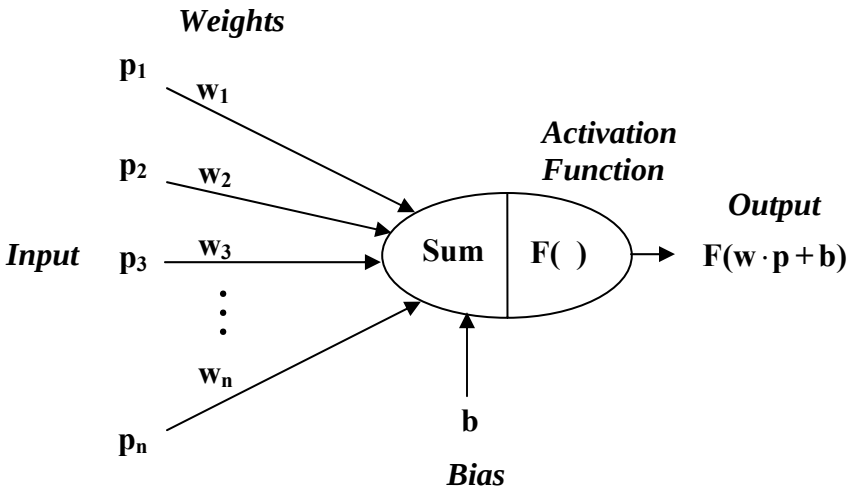

Figure 4-5. Structure of a basic neuron

Levenberg Marquardt (LM)

The Levenberg Marquardt (LM) method is one of optimization or nonlinear least squares solution, and it is the fastest training method in the backpropagation training method. Levenberg [26] and later Marquardt [27] suggested a damped Gauss-Newton method. LM searches the minimum of a multivariable function which is a performance function and the sum of the squares of the error in artificial neural network. LM is iterative like other nonlinear optimization methods, and it does not require computation of the Hessian matrix. The performance function can be expressed with the Jacobian gradient and the Hessian matrix. If the Jacobian gradient is zero at a stationary point or a saddle point and the Hessian matrix is positive definite, then the input variables, which

are weights and biases in artificial neural networks, are in local minima. LM can be computed as

$$\Delta \mathbf{x} = -[\mu \mathbf{I} + \mathbf{J}^T \mathbf{J}]^{-1} \mathbf{g} \tag{4.4}$$

where μ is scalar, $\mathbf{g} = \mathbf{J}^T e$, $\mathbf{J}$ is the Jacobian matrix, and $\mathbf{J}^T \mathbf{J}$ is an approximate Hessian matrix. From the equation, the value of μ is decreased or increased depending on how the performance function changes. Therefore, LM can be thought of as a combination of steepest descent and the Gauss-Newton method. When the current solution is far from the correct one, μ is increased, and the algorithm behaves like a steepest descent method, which is slow but guaranteed to converge. When the current solution is close to the correct solution, μ is decreased, and LM becomes a Gauss-Newton method. The μ can be regarded as a learning rate [34].

The main disadvantage of the LM method is that it requires storage memory for some matrices which is quite large for certain problems. A memory reduction method can reduce the memory usage, but there is a drawback to using memory reduction. The memory deduction method performs that a large matrix is broken up into submatrices, and a significant computational overhead is associated with computing the large matrix in submatrices [18].

4.2.3. Radial Basis Function Network (RBF)

A Radial Basis Function (RBF) is another type of artificial neural networks. RBF has a similar architecture of multilayer networks, but it uses a distance between weights and inputs instead of weighted inputs. The distance is multiplied by biases. RBF has two

layers, a hidden layer and an output layer. For the hidden layer, a Gaussian nonlinear transfer function is used. The outer layer neurons are activated by a standard linear function like a multilayer perceptrons method. If the distance is close to zero, the nonlinear function at the hidden layer has maximum output, and the hidden layer neuron is activated. A spread constant value is used for RBF, and the value is related to the bias in the hidden layer. RBF is good for identifying a function with less training time. However, it requires more neurons than other methods [36, 38].

4.2.4. Generalized Regression Neural Network (GRNN)

Generalized Regression Neural Network (GRNN) is a kind of RBF, but has a special linear layer. The first hidden layer is a radial basis layer, which is the same as RBF. The special linear layer is the summation layer, which consists of the summation neurons and one division neuron. The summation neuron calculates the sum of the weighted outputs of the pattern layer, and the division neuron calculates the sum of the un-weighted outputs of the pattern layer. The output layer divides the output of the summation neuron by the output of the division neuron. This method also has good results with appropriate spread constants outside of its training range, and it has the fastest training method [39].

In Chapter 5, the simulation approach for inverse kinematics of a 3-DOF planar manipulator using the five methods explained in this chapter is described. Kinematics of the planar manipulator is analyzed, and details for the each method are explained.

CHAPTER 5:Simulation Approach

5.1. Introduction

In order to choose an appropriate and efficient algorithm to execute the inverse kinematics, a series of simulations were conducted using a 3-DOF planar manipulator as shown in Figure 5-1. The purpose of the inverse kinematics is to determine the values of joint variables given in Cartesian space. The differential kinematics is described by a Jacobian matrix. The Jacobian is useful for finding singular configurations, analyzing redundancy, and determining inverse kinematics algorithms. The main issue which should be addressed is the complexity of the inverse kinematics solution that results from the nonlinearities; as a result, a closed form solution may not be found. Multiple or infinite solutions may exist when the Jacobian matrices are rank deficient or the manipulators are kinematically redundant. Even when the inverse kinematics has a closed form solution, unstable movements may happen near the singularities.

This simulation introduces several techniques based on fuzzy logic and artificial neural network systems, like ANFIS, Fuzzy-GA, MLP-LM, RBF, and GRNN, to solve for the inverse kinematics solution of the 3-DOF planar manipulator. Computing the inverse kinematics using fuzzy logic and artificial neural networks overcomes the disadvantage of the large amount of calculations, so that the real-time performance can be improved to suitable accuracy. The methods also may solve singularity problems.

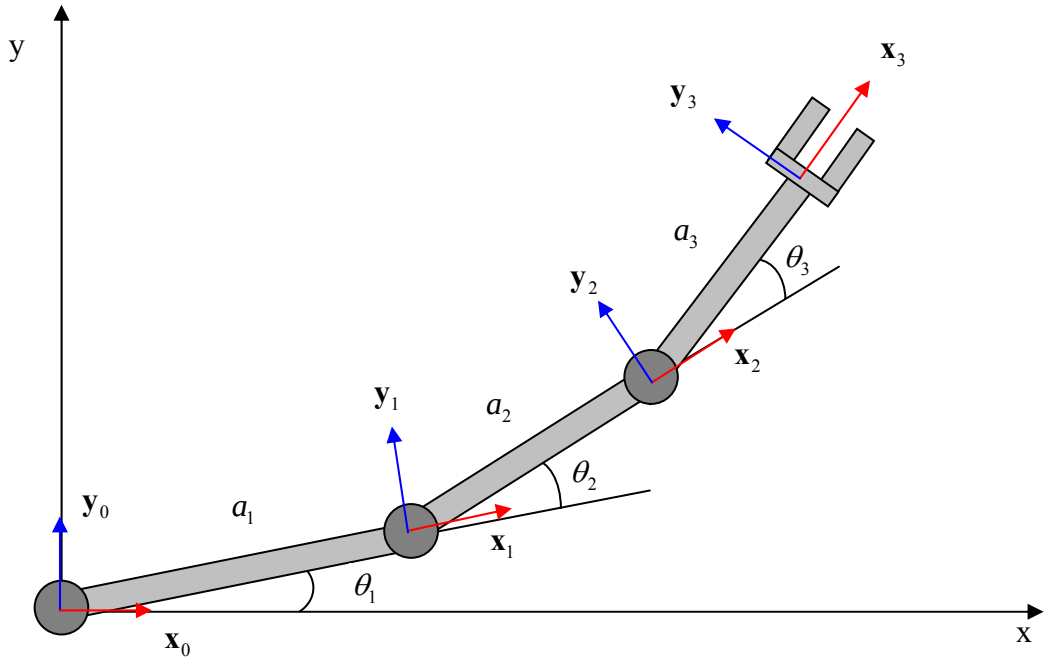

Figure 5-1. 3-DOF planar manipulator

The 3-DOF planar manipulator was chosen to perform the simulation because it has a simpler architecture and less number of degree of freedom compared to the real 6-DOF manipulator. With the 3-DOF manipulator, it is possible to simulate several artificial intelligent methods in less time and easily to modify the artificial intelligent methods to improve their performance. If a greater number of DOF is chosen, then this manipulator requires complicated architecture of each artificial method. Therefore, more training or modification time is required. If a smaller number of DOF manipulator is chosen, then it is possible to simulate easily and quickly, but it may not adaptable to move on the real simulation due to its small number of DOF. Furthermore, the 3-DOF manipulator is a redundant manipulator in 2-D work space because the simulation is only concerned in a position operation. Therefore, the Jacobian matrix of the 3-DOF

manipulator is not square due to its extra degree of freedom, so that the simulation of the 3-DOF manipulator can be adapted to a redundant manipulator simulation too.

In order to solve the inverse kinematics problems, much research has been pursued in artificial intelligence. For instance, many researchers applied fuzzy logic in 1990's. For example, fuzzy logic was applied for the inverse kinematics solution of a 3-DOF planar manipulator based on the gradient method. The outputs of a fuzzy logic system are joint velocities, and the inputs are a transpose of Jacobian matrix with current errors [24]. In another approach in [40], Fuzzy Associative Memory (FAM) is used as a regular fuzzy logic rule-base, and applied to a 4-DOF planar manipulator. The outputs are joint velocities, and the inputs are Jacobian and Cartesian velocities. For hybrid fuzzy logic which is combined with other artificial intelligence, [41] uses ANFIS for a 2-DOF planar manipulator, and [42] uses GA for Stanford and puma 260 robots. For another instance for the artificial neural networks, inverse kinematics and geometrically bounded singularities prevention are applied for a 3-DOF planar manipulator [43]. This approach uses two neural networks to perform redundancy resolution. The outputs are joint velocities and inputs are Cartesian velocity and current joint angles. [44] uses a Hopfield network, a recurrent network, to perform inverse kinematics for 4-DOF planar manipulator. The outputs are joint accelerations from an energy function, and inputs are Cartesian velocity and current joint angles.

From the above work, it is easily seen that many kinds of investigations were performed by artificial intelligence for inverse kinematics calculations. However, these artificial intelligence methods were mostly applied to low number of DOF planar manipulators. Especially, Cartesian velocities and current joint angles were used as inputs

to perform inverse kinematics. However, if an artificial intelligence method computes an inverse Jacobian matrix instead of Cartesian velocities and joint angles, then the inverse kinematics can be performed simpler and faster. In this case, inputs of the new artificial intelligence are current joint angles only, and the inverse kinematics is performed separately. This method allows fast execution time and fewer number of inputs. Furthermore, this novel method can be implemented by other inverse techniques such as pseudo-inverse, DLS, and redundancy resolution.

The planar manipulator is a combination of three links and three joints with one end fixed and the other end free. The joints are all revolute and are driven by actuators. In order to move the free end, called the end-effector, along a certain path, the joints are to be moved to track the desired path. It is necessary to know the displacements of the joints at each instant of time. In terms of robotics, kinematics is the study of motion of manipulators with position, velocity and acceleration of each link and the end-effector without consideration of masses and torques/moments [20]. The forward kinematics approach determines the position and orientation of the end-effector in a Cartesian space, given the joint displacements and the link parameters. This approach always has a unique solution. The other approach, called the inverse kinematics, deals with finding the joint displacements for a given position and orientation of the end-effector. The inverse kinematics approach in robotics is essential to robot motion panning and control. While forward kinematics is simple, straightforward, and has a unique solution, the inverse kinematics can be complex depending on the structure of the robot and its number of degrees of freedom. The inverse kinematics of the 3-DOF manipulator normally has multiple or even infinite possible solutions and it is not always obvious which set of joint

angles to choose. This makes it difficult for the manipulators to track the Cartesian commands in real-time.

Figure 5-2 shows a typical block diagram of an industrial robot, which has two phases, a planning phase and an execution phase. In the planning phase, a desired trajectory is represented by inverse kinematics. The error between the outputs of the inverse kinematics and actual joint angles is delivered to the controller in the execution phase. The controller drives the manipulator to track the converted joint movements. In the simulation, only the planning phase is considered.

DH parameters for the 3-DOF planar manipulator are shown in Table 5-1. From the DH parameters, the homogeneous transformation matrix from base frame to the end- effector frame is calculated as

$$\mathbf{T}_3^0 = \begin{bmatrix} c_{123} & -s_{123} & 0 & a_1 c_1 + a_2 c_{12} + a_3 c_{123} \\ s_{123} & c_{123} & 0 & a_1 s_1 + a_2 s_{12} + a_3 s_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

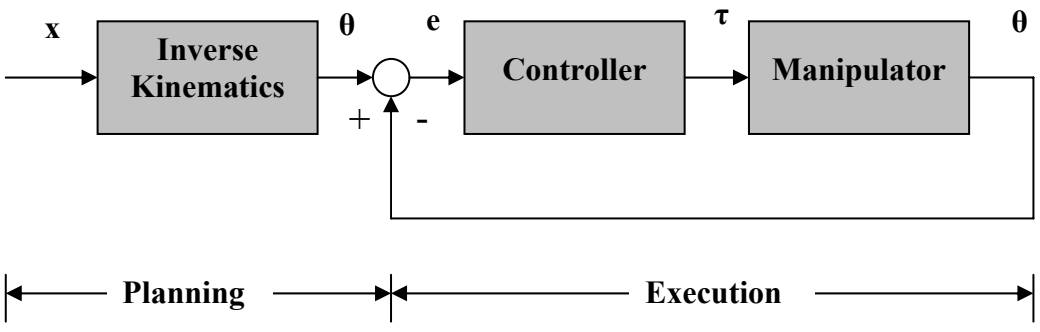

Figure 5-2. General block diagram of a manipulator

Table 5-1. DH Parameter for the 3-DOF Manipulator

i	a_i (m)	α_i (rad)	d_i (inch)	θ_i (rad)
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

where $\mathbf{J} \in \mathbf{R}^{2\times 3}$ is the end-effector Jacobian of the manipulator. The Jacobian matrix is calculated as

$$\mathbf{J} = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \end{bmatrix} \tag{5.3}$$

where $J_{11} = -a_1 \sin(\theta_1) - a_2 \sin(\theta_1 + \theta_2) - a_3 \sin(\theta_1 + \theta_2 + \theta_3)$

$J_{21} = a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3)$

$J_{12} = -a_2 \sin(\theta_1 + \theta_2) - a_3 \sin(\theta_1 + \theta_2 + \theta_3)$

$J_{22} = a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3)$

$J_{13} = -a_3 \sin(\theta_1 + \theta_2 + \theta_3)$

$J_{23} = a_3 \cos(\theta_1 + \theta_2 + \theta_3)$.

The above equation can be solved for $\dot{q}$ by means of the pseudo-inverse $\mathbf{J}^{\dagger}$ as

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{\dagger}(\theta)\dot{\mathbf{x}} \tag{5.4}$$

where $\mathbf{J}^{\dagger}$ is the pseudo-inverse of $\mathbf{J}$. If the rank of the Jacobian matrix is 2, then the right pseudo-inverse is $\mathbf{J}^{\dagger} = \mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1}$. The right pseudo-inverse of $\mathbf{J}$ can also be found using singular value decomposition. If the rank is less than 2, the pseudo-inverse can be solved as $\mathbf{J}^{\dagger} = \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{U}^{T}$ using SVD. In the redundancy resolution, it is crucial to compute $\mathbf{J}^{\dagger}\dot{\mathbf{x}}$ for each of the relevant joint space configuration. However, it is very difficult to achieve the desired real-time operation with conventional digital and sequential computational methods.

The Figure 5-3 shows the workspace of the 3-DOF planar manipulator. The home position of the manipulator is $\theta_1 = 18°$, $\theta_2 = 70°$, and $\theta_3 = 70°$ which is an elbow up

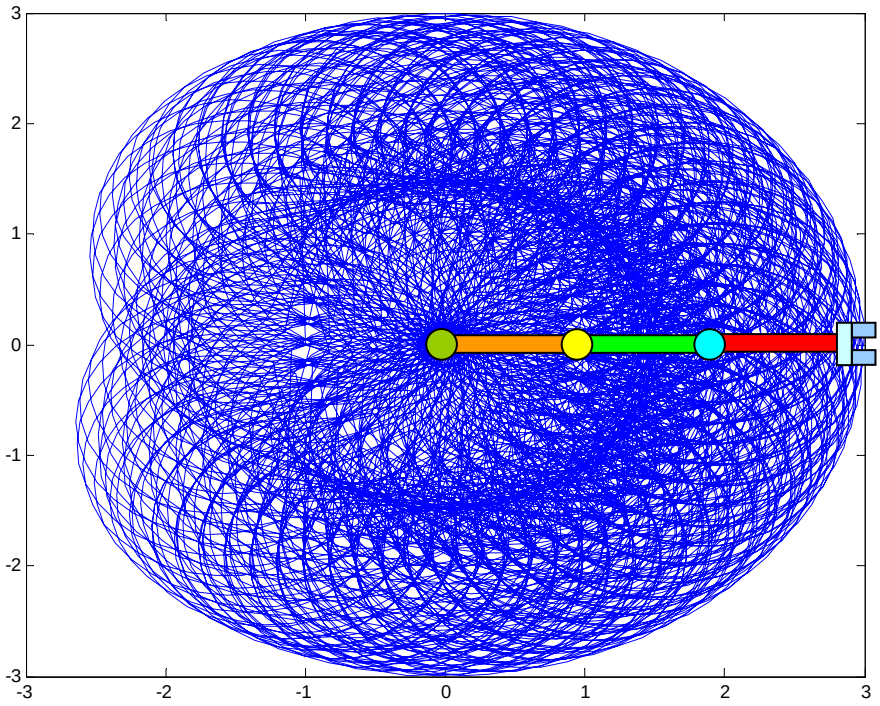

Figure 5-3. Workspace of the 3-DOF manipulator

configuration. Joint limit ranges are arbitrarily chosen as $-130 \le \theta_i \le 130$ where $i = 1, 2, and\ 3$ each revolute joint. A length of one meter is selected for all the links. The next section discusses the fuzzy logic for the 3-DOF planar manipulator simulation.

5.2. Fuzzy Logic

5.2.1. ANFIS

As shown in Figure 5-4, the fuzzy logic inference system takes as inputs the elements of the Jacobian matrix, which is calculated from the current joint variable values. From these inputs, the fuzzy logic inference system generates as outputs the elements of the Jacobian pseudo-inverse matrix, so that the inverse kinematic system calculates the

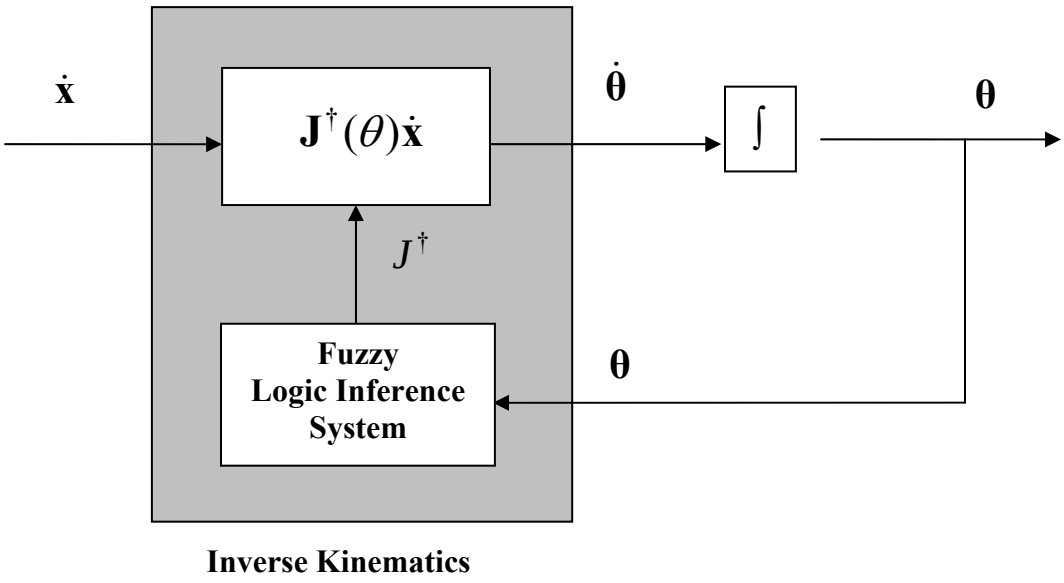

Figure 5-4. Inverse Kinematics system for ANFIS

necessary trajectories for the joint variables based on the location of the end-effector. Instead of performing inverse kinematics, the fuzzy inference system generates the Jacobian pseudo-inverse matrix first, and then performs the inverse kinematics calculations, so that the number of inputs can be reduced. If the inference system performs inverse kinematics, generally eight inputs are required: two for the location of the end-effector and six for the elements of the Jacobian matrix. The number of inputs strongly influences the training and performance time.

As mentioned in Chapter 4, ANFIS has some constraints, unlike a regular fuzzy logic method. The most critical one is that ANFIS can accept only one output. Therefore, the inverse kinematic system needs six fuzzy logic inference systems for each element of the Jacobian pseudo-inverse matrix. As the number of degrees of freedom of the manipulator increases, the number of required inference systems also increases. The number of membership functions is nine at each input, and triangular membership functions are chosen as shown in Figure 5-5. For the output membership, a linear function is chosen as the first order Sugeno-type fuzzy model. To apply the fuzzy operators, prod is selected for the AND operator, and probor is selected for the OR operator. The final output of the system is the weighted average of all rule outputs for defuzzification.

To train the weights for the rule-base, first 10,000 data sets of joint angles are generated randomly. The data set is large enough to cover all possible joint angles, even though they are randomly collected. However, for the exact universe of discourse, the maximum and minimum joint angles are also added manually. From the data set, the

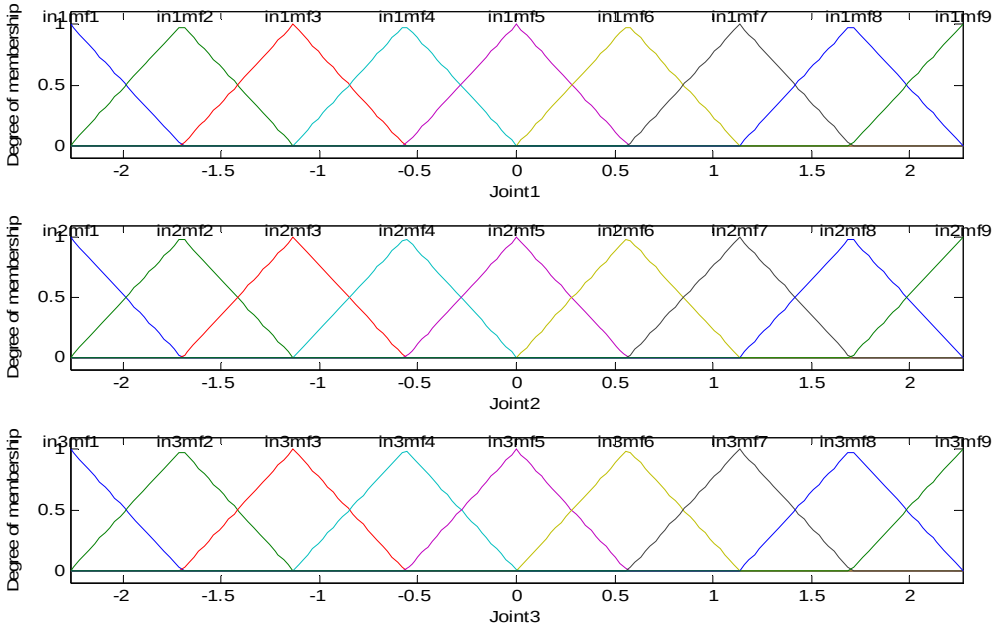

Figure 5-5. Membership functions for fuzzy inputs ($J_{11}^{\dagger}$)

fuzzy input / output sets are calculated. The fuzzy inference system needs six inputs, which are elements of the Jacobian matrix, as well as a single output, which is an element of the Jacobian pseudo-inverse matrix.

5.2.2. Genetic Algorithm

For an alternate method of generating and adjusting fuzzy membership functions and the fuzzy rule-base automatically, a Genetic Algorithms (GA) method is used in the inverse kinematics solution. The main structure of the fuzzy inference system used has the same configuration as shown in Figure 5-4. The inputs of the fuzzy inference system are elements of the Jacobian matrix, which are calculated from the current manipulator

joint angles, and the output is the pseudo-inverse matrix. Therefore, without mass computation of the Jacobin matrix, the pseudo-inverse and SVD, this system can perform inverse kinematics.

To apply a Genetic Algorithm to the fuzzy inference system, a means of evaluating different designs is required. This evaluation or fitness needs to be performed relatively quickly as a GA needs to be able to process large numbers of different combinations of parameters. The evaluation function is a function called by a GA to calculate the fitness of parameters from a chromosome, as shown in Figure 5-6. The parameters are passed to the evaluation function, which processes and returns a value corresponding to how well the parameters performed. For this, an evaluation function

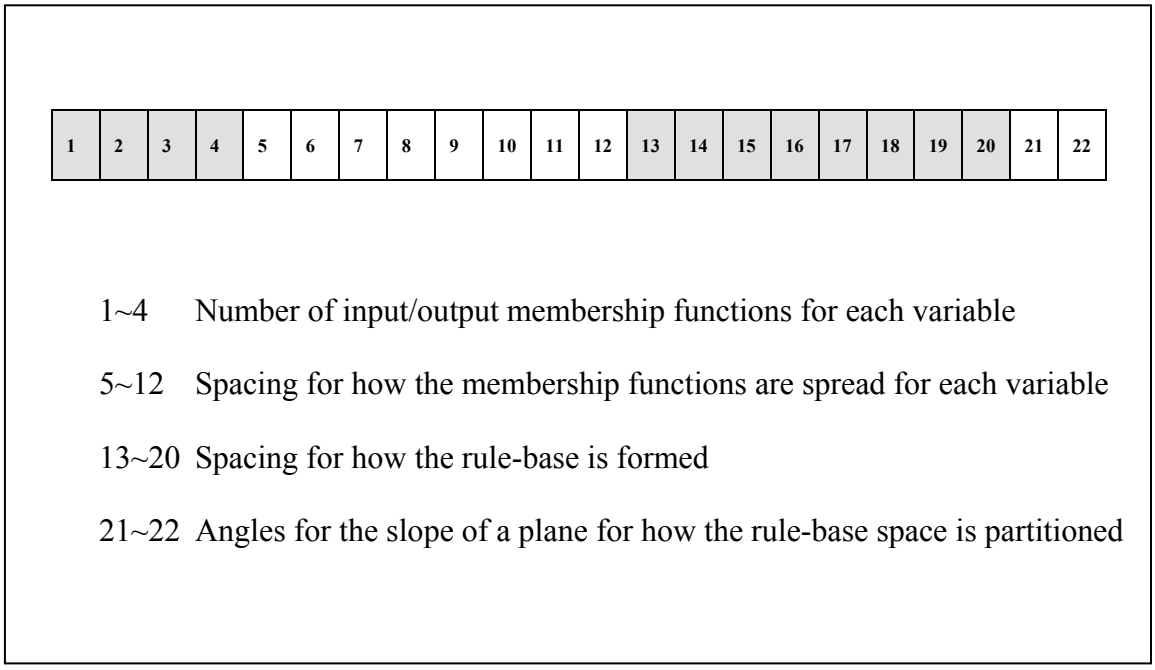

Figure 5-6. Chromosome

program is written, and the evaluation function of a rule is expressed as

$$Fitness = \frac{1}{1 + \frac{\sum |error|}{N}} \tag{5.5}$$

where *error* is $J^{\dagger}_{desired} - J^{\dagger}_{fuzzy}$, and N is the total number of the error. After performing error checking, the parameters are used for creating fuzzy inference system files, and setting the appropriate scaling factors.

To run a GA, a suitable encoding for each of the parameters and bounds for each of them must be selected. Binary encoding is used to allow the GA algorithm to search the solution more precisely. The numbers of membership functions are limited to the odd integers between three and nine. Therefore, the information can be captured in two bits per variable. The spacing parameters specify how the memberships are spaced out across the universe of discourse. The value of the parameters indicates whether the membership functions are close together at the center of the range or spread out at the limits. The rule-base also needs to be specified. Characteristic spacing parameters for each variable and characteristic angles for each input variable are used to construct the rules. The spacing parameters use a spacing method similar to the one used in the membership functions. The angle parameters determine the slope of a line through the origin on which seed points are placed. From those parameters, the membership functions and the rule-base can be generated by the GA. In the next section, the artificial neural network is presented in similar approach as the fuzzy logic.

5.3. Artificial Neural Network

Multilayer neural perceptrons (MLP) networks are applied to the inverse kinematics problem, and the networks are trained with mapping between elements of the Jacobian matrix and elements of the Jacobian pseudo-inverse matrix, as shown in Figure 5-4. The Jacobian matrix is calculated from the DH parameters and the homogeneous transformation matrix. The set of all possible training data is acquired randomly, like the fuzzy logic inference system. To cover the entire workspace, 15,000 data are generated for inputs / outputs. Unlike ANFIS, a neural network can have more than one output, but if the number of outputs is increased, the training time and error are also increased. Therefore, it is important to find an optimal number of outputs while comparing to the training time and overall error. After training, the performance of the system is tested by having the network generate joint angles for arbitrary end-effector trajectories.

Neural networks have been applied for a variety of applications which involve non-linear relations between the input and output patterns. The inverse kinematics application is highly nonlinear, as it involves the inverse of the Jacobian matrix. Neural networks are more precise for the inverse kinematic system than other artificial intelligence methods if given data set is well collected and exact. Therefore, it is important to collect all possible and general data without noise or disturbances. For this, overcoming singularities by several methods are required even though most data are calculated rather than collected from simulation experiments. The DLS method is chosen in the simulation. DLS is a powerful method not only to approximate a rank deficient matrix, but also to prevent sudden jumps of a manipulator.

In order to reduce performance time, six outputs from the artificial neural network are used. This allows for only one network to be required to execute the inverse kinematics. However, the training time is longer and more neurons are required to reduce the sum of the squared errors. For the 3-DOF planar manipulator, 48 neurons are used for six outputs. If the number of outputs is reduced, the number of neurons can be reduced, but more networks are needed, which results in slower performance. The number of hidden layers is also essential to increase performance. If the number of hidden layers is raised, the calculation of the inverse kinematics will be extended. However, especially in order to identify a nonlinear system, increasing hidden layers helps to reduce errors. Because there are no formal processes to decide the number of neurons and hidden layers, several experiments are necessary.

Another consideration in artificial neural network design is the activation functions. A specific activation function is required for a certain neural network. For instance, a perceptron network uses a hard limit function, and a linear network requires a linear activation function. LM also needs a linear activation function for the output layer. However, for other layers, LM does not require specific activation functions. In general, if the inputs and outputs have nonlinear relationships, then nonlinear activation functions are used like sigmoidal functions. MATLAB has two kinds of sigmoidal functions: logsig and tansig. For the simulation, tansig functions are used, though they are not significantly different.

LM uses an approximate Hessian matrix, and this allows for fast searching for local or global minima. However, if many neurons or large training data sets are used, the Hessian matrix becomes very large, which requires large memory space in order to

perform the LM method. Therefore, there is a limitation on increasing neurons and training data sets. In MATLAB, a memory reduction parameter can be used for reducing memory usage. However, the training time will be slower if that parameter is used. The basic idea of the memory reduction option is that the Hessian matrix can be calculated by dividing into submatrices. Therefore, the whole Hessian matrix cannot be stored in memory, but the calculation time is increased.

RBF and GRNN are similar to MLP, but they use a spread constant. The larger spread constant the function approximation is smoother. If the smaller spread constant than the normal distance between inputs and weights, then data are fit too closely. In Chapter 6, the simulation results for the 3-DOF planar manipulator are discussed and analyzed to find the best method. The MLP is chosen for the real teleoperation system, and the final simulation for the Titan II manipulator is performed using the MLP method.

CHAPTER 6:Simulation Results

6.1. Introduction

The previous chapters provided a variety of simulation approaches for each artificial method to perform inverse kinematics to obtain Cartesian space and joint space mapping. The ultimate goal of this simulation is to manipulate the Titan II by using the inverse kinematics. In this chapter, the simulation results will be presented and discussed for each of the methods that were proposed to perform the inverse kinematics for the 3-DOF manipulator. Based on performance and error, the best of these methods will be identified, and that method will be adopted for the 6-DOF Titan II manipulator.

Using MATLAB, a circle trajectory is generated and simulated for each method as shown in Figure 6-1. For each method, the generated trajectory will be performed by the 3-DOF manipulator, and the results will be analyzed. After finding the best method, a more complex trajectory will be simulated in MATLAB and RoboWorks for the Titan II manipulator. It is important that the final simulation is as similar to the real system as possible, so that eventually the best method can perform as part of the WAM-Titan II teleoperation system.

MATLAB, which uses a matrix computing environment, has powerful tools for fuzzy logic and artificial neural networks. Its programming language is easy to manipulate and plot any numerical calculation compared to other languages. However,

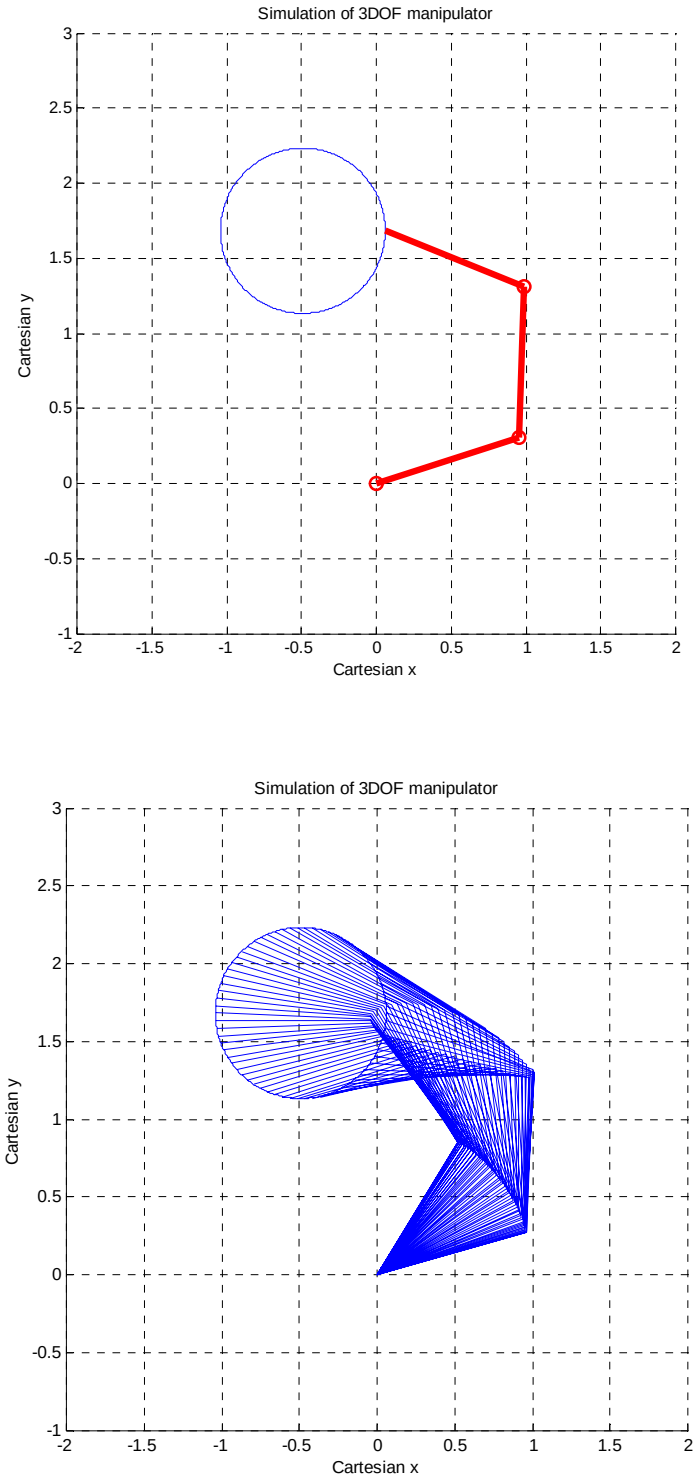

Figure 6-1. A circle trajectory for a 3-DOF manipulator

symbolic calculations are not easy to perform using MATLAB. MATLAB has many useful toolboxes, such as the Signal Processing Toolbox, the Control System Toolbox, the Fuzzy Logic Toolbox, and the Neural Network Toolbox. For this simulation, the Fuzzy Logic Toolbox and the Neural Network Toolbox will be used.

The RoboWorks robot simulator, created by Newtonium, can be used for modeling and animating 3D mechanical objects. This simulator has several tools for creating and modifying 3D objects. There is a hierarchy of objects that allows for a lower rank object to inherit position and orientation from a higher rank object. For example, if the higher rank object is moved or rotated, all lower rank objects are moved or rotated together, like a manipulator's links. Therefore, this program makes it easy to model and simulate manipulators. Furthermore, it has a communication tool called RoboTalk that allows for communication via a keyboard, a data file, or the Ethernet. RoboTalk allows for precise simulations and communication between different platforms. However, the program does not have any function to simulate physical interaction or other phenomena. It only accepts kinematic situations which are suitable for this simulation. In the next section, results of fuzzy logic for the 3-DOF manipulator are presented.

6.2. Fuzzy Logic Results

6.2.1. ANFIS

For simulating the fuzzy inference system using ANFIS, nine triangular membership functions were defined at each fuzzy inference system. The total number of

inference systems was six for each element of Jacobian pseudo-inverse matrix. After training for several epochs, ANFIS optimized the weights of the Sugeno-type fuzzy system. The sum of squared errors is shown in Table 6-1, and the output surfaces are shown in Figure 6-2.

The range of the training data set was selected to be the range from −130° (-2.27 radian) to 130° (2.27 radian), for every joint, which means that the 3-DOF manipulator can easily reach a singularity. Therefore, the number of membership functions should be increased until each fuzzy inference system meets an acceptable sum of squared error (SSE). The maximum and minimum joint angles were added to the training data, so that the universe of disclosure covered all possible joint angles. Therefore, the number of training data points was 10,002 instead of 10,000.

Table 6-1. Parameters Used for Training and Errors

FIS	Pseudo-inverse of J	Training data	Membership function	Type of MF	SSE
1	$J_{11}^{\dagger}$	10002	9-9-9	triangular /linear	0.04437
2	$J_{12}^{\dagger}$	10002	9-9-9	triangular /linear	0.05827
3	$J_{21}^{\dagger}$	10002	9-9-9	triangular /linear	0.05450
4	$J_{22}^{\dagger}$	10002	9-9-9	triangular /linear	0.06546
5	$J_{31}^{\dagger}$	10002	9-9-9	triangular /linear	0.07233
6	$J_{32}^{\dagger}$	10002	9-9-9	triangular /linear	0.08914

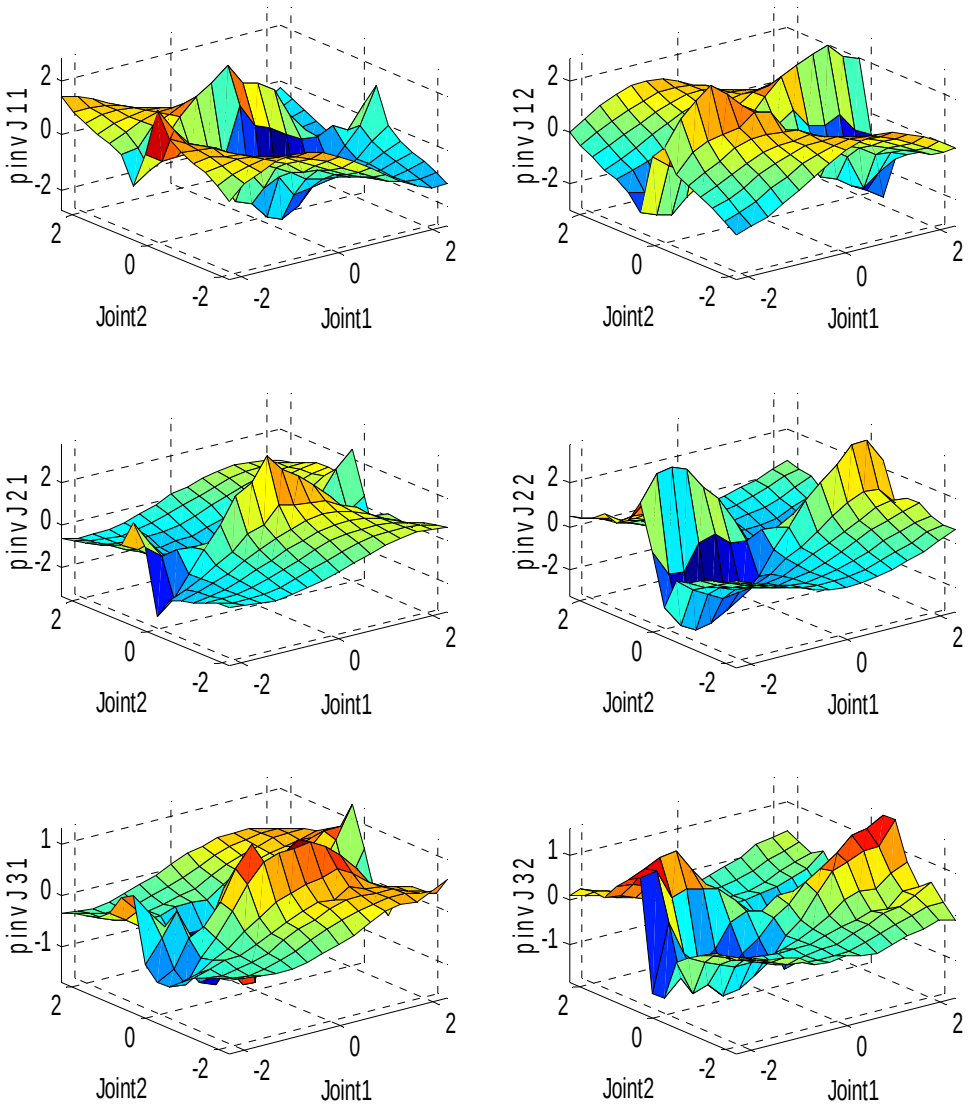

Figure 6-2. Output surfaces for the fuzzy inference system

After training all the fuzzy inference systems, the 3-DOF manipulator was again simulated with a circle trajectory. The error of the Jacobian pseudo-inverse is shown in Figure 6-3. The error was defined as $\mathbf{J}^{\dagger}_{desired} - \mathbf{J}^{\dagger}_{fuzzy}$, and the maximum error was 0.04 for $J^{\dagger}_{21}$. Two-norm of errors is also calculated.

Figure 6-4 and Figure 6-5 show the joint velocity error and Cartesian velocity error of the 3-DOF manipulator end effector. The largest error generally occurs at the middle of the iteration time because of the shift that occurs there. It increases until the end effector passes the halfway point through the circle trajectory; after that point, it decreases. Another possible reason for the error is that the fuzzy inference systems may not have been trained enough near singularities. The maximum error of the joint velocity was $1.9\times10^{-4}\ rad/\mathrm{s}$ at joint two, and the maximum error of the Cartesian velocity was $3.3\times10^{-4}\ m/\mathrm{s}$ in the *y* direction. Figure 6-6 depicts the Cartesian position error between the desired trajectory position and the actual trajectory position. The maximum error was 0.03 meters.

6.2.2. Genetic Algorithm

To perform the simulation of the fuzzy inference system using GA, MATLAB was used with the open source Genetic Algorithm Optimization Toolbox (GAOT), which is provided by Houck et al. [25]. The evaluation function has to be provided for the toolbox, and equation (5.5) was applied. The encoding for each of the parameters and bounds for each of them were selected and used as discussed in Chapter 5.The numbers

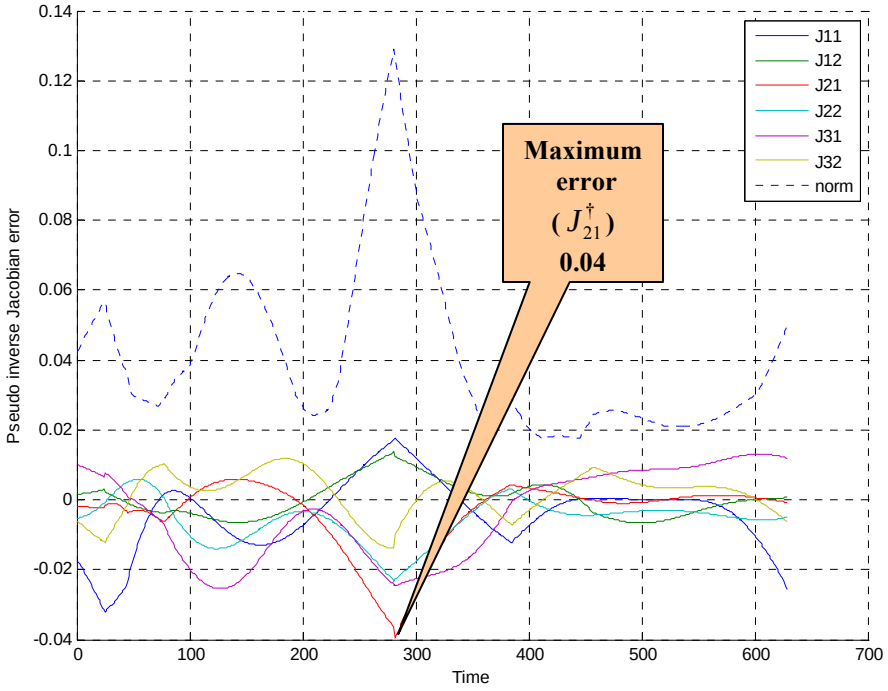

Figure 6-3. ANFIS Pseudo-inverse Jacobian error

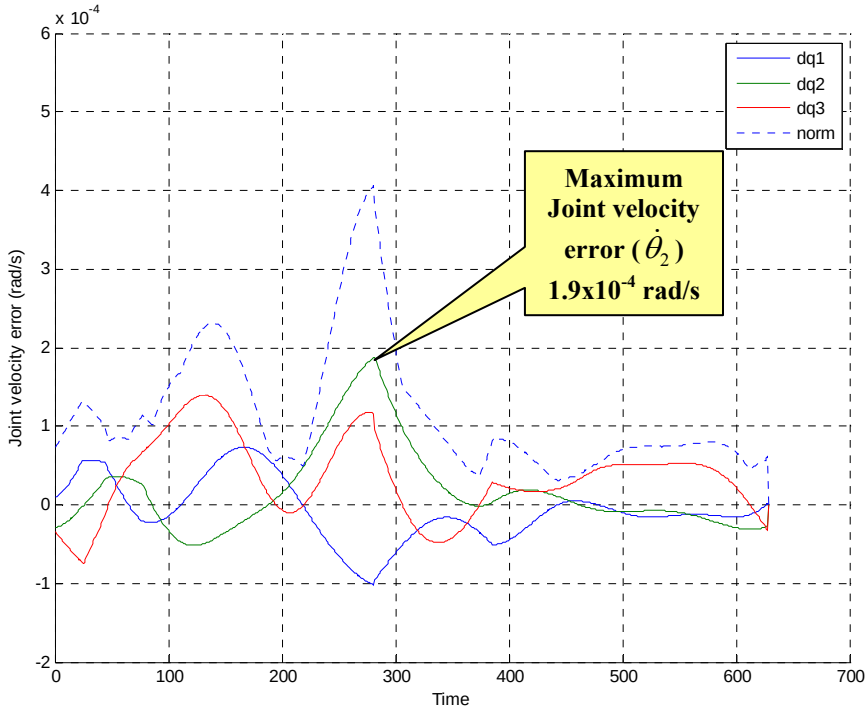

Figure 6-4. ANFIS Joint velocity error

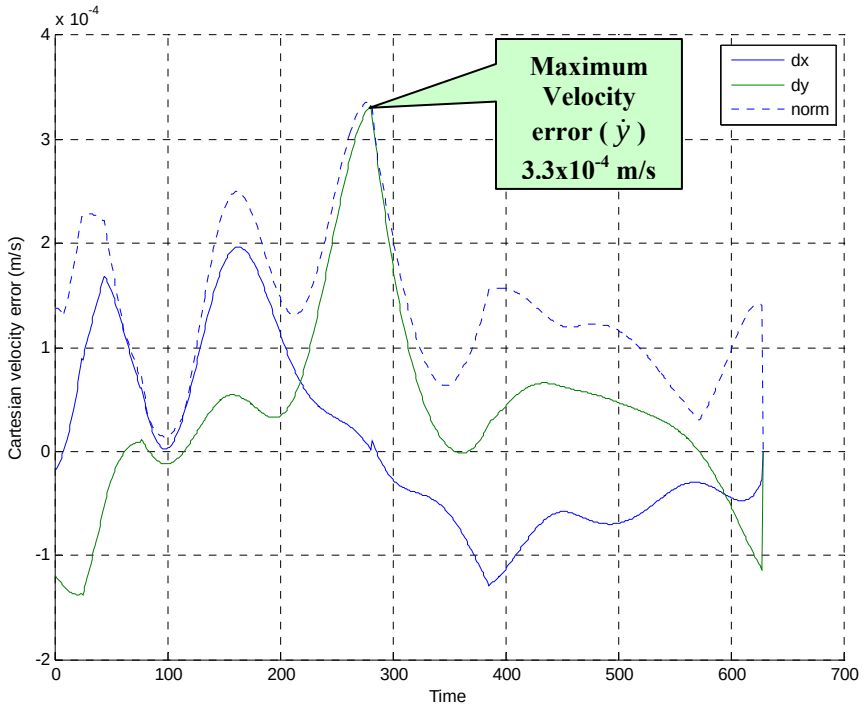

Figure 6-5. ANFIS Cartesian velocity error

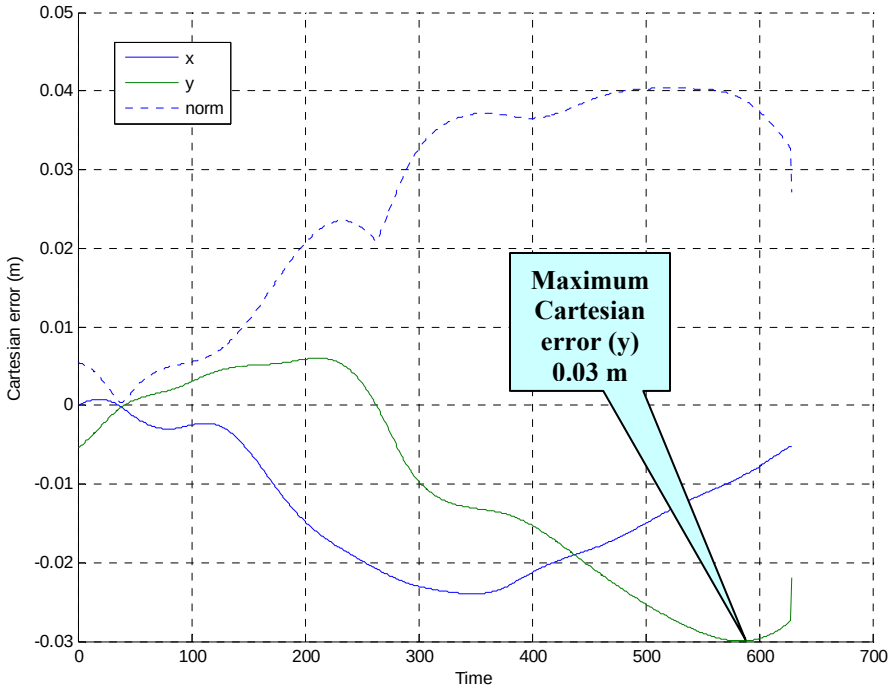

Figure 6-6. ANFIS Cartesian error

of membership functions are limited to the odd integers, and this requires just two bits per variable. The other parameters, along with their ranges and precisions, were selected as given in Table 6-2.

The same architecture of the fuzzy inference system was used for GA. The total number of the inference system was also six for each element of Jacobian pseudo-inverse matrix. 10002 data sets of inputs and outputs were used for running the GA. Several runs of the GA were performed for a hundred generations each. A plot of the Jacobian pseudo-inverse error is shown in Figure 6-7. The maximum error was 0.05, and the shapes of the curves are generally smoother than those from ANFIS. The joint velocity and Cartesian errors are shown in Figure 6-8 and Figure 6-9. Unlike ANFIS, there is no large error at the middle of the iteration time. However, the overall error of the GA is bigger than that of ANFIS. The maximum error of the joint velocity was $2.8\times10^{-4}\ rad/\mathrm{s}$ at joint one, and the maximum error of the Cartesian velocity was $5\times10^{-4}\ m/\mathrm{s}$ in the *x* direction. The maximum Cartesian position error was 0.035 meters, which is slightly bigger than that of ANFIS, as shown in Figure 6-10.

Table 6-2. Parameters Used for Encoding

Parameter	Range	Precision	Number of Bits
Number of MF	3~9	2	2
MF Spacing	0.1 ~ 1.0	0.01	7
MF Spacing (exponent)	-1 ~ 1	2	1
Rule-base spacing	0.1 ~ 1.0	0.01	7
Rule-base spacing (exponent)	-1 ~ 1	2	1
Rule-base angle	$0 \sim 2\pi$	$\pi/512$	11

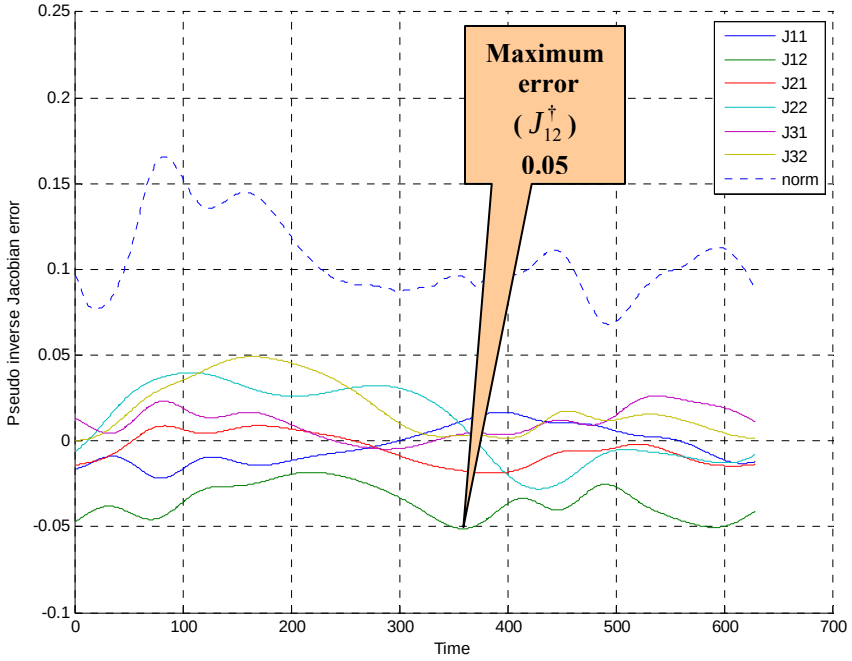

Figure 6-7. Fuzzy-GA Pseudo-inverse Jacobian error

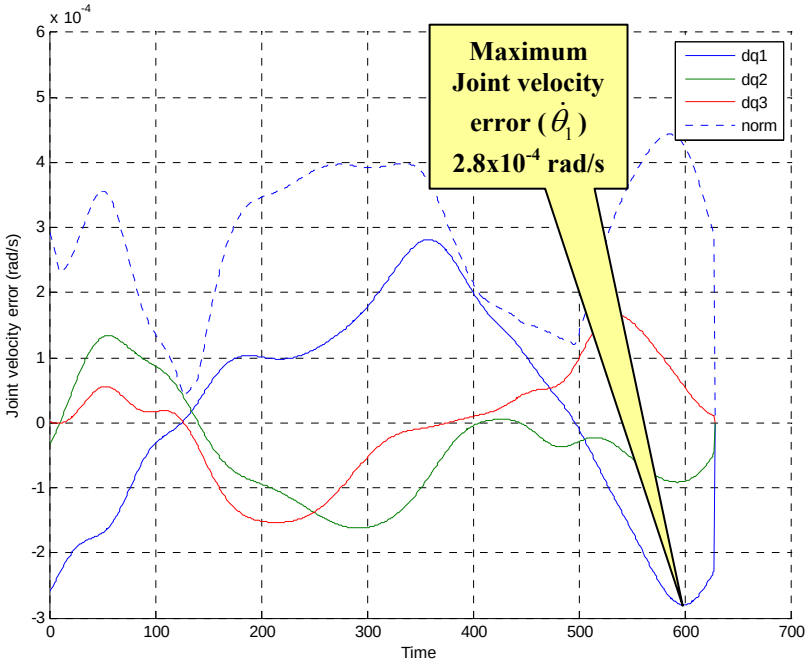

Figure 6-8. Fuzzy-GA Joint velocity error

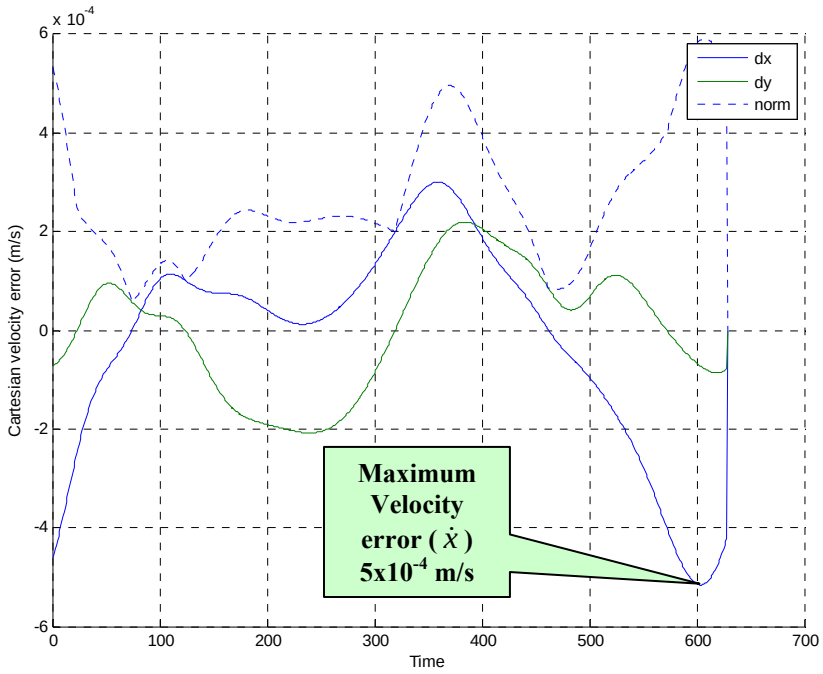

Figure 6-9. Fuzzy-GA Cartesian velocity error

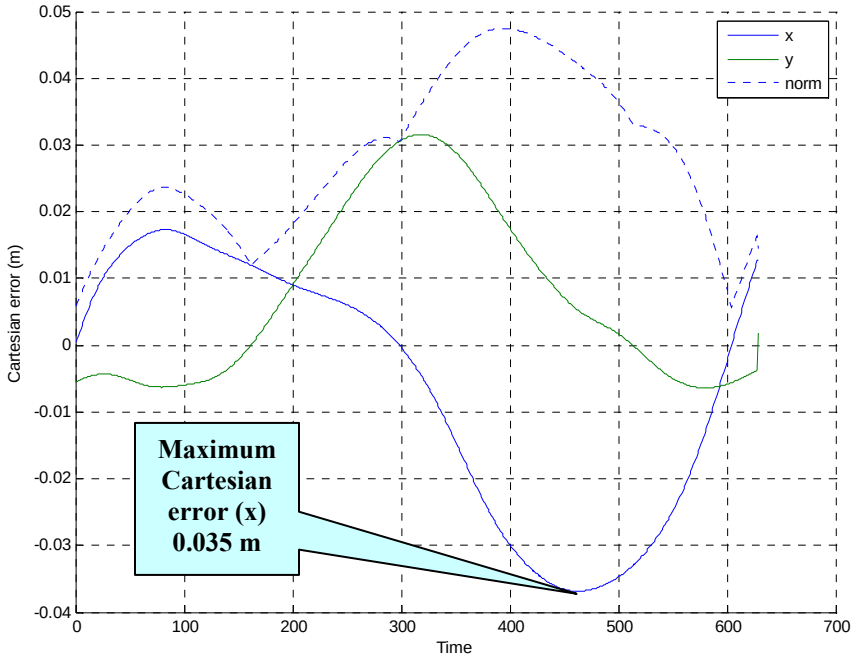

Figure 6-10. Fuzzy-GA Cartesian error

The next section shows results of the artificial neural network for the 3-DOF manipulator by each proposed method.

6.3. Neural Network Results

6.3.1. Multilayer perceptrons network

The simulation for the artificial neural network with the multilayer perceptrons network was performed with an architecture similar to that of the fuzzy inference system. The network identified the pseudo-inverse Jacobian matrix with current joint values as inputs. Multilayer Perceptrons were used for the structure of the network, and an LM optimization method was adopted for training. The greatest benefit of the artificial neural network is that the number of outputs is not constrained, so that the six elements of the Jacobian pseudo-inverse matrix could be used for one output. Therefore, only one artificial neural network system was required to achieve the simulation of the 3-DOF planar manipulator.

The same numbers of input/output data were collected, like fuzzy logic and GA, and they were trained with four hidden layers. At each hidden layer, thirty neurons were used to meet the acceptable SSE. However, due to the large number of neurons, a tremendous amount of memory was required for the computations needed to train the network. To solve the problem, the memory reduction parameter was setup as fourteen in MATLAB. However, the memory reduction resulted in very long computational time. After training the network, the simulation time of the ANN was fastest among the other

methods. For the four hidden layers and one output layer, the *tansig* and *purelin* activation functions as shown in Figure 4-4 were chosen.

Figure 6-11 shows the Jacobian pseudo-inverse error of the circle trajectory; the maximum error was 5×10^{-3}. The maximum error of the joint velocity and the Cartesian velocity are $2.7\times10^{-5}\ rad/\mathrm{s}$ and $0.7\times10^{-4}\ m/\mathrm{s}$, as shown in Figure 6-12 and Figure 6-13. The maximum position error is 0.013 meters, as shown in Figure 6-14. It is clear from the graphs that the end effector tracking with the 3-DOF manipulator was very good with small errors compared to the previous methods. It is also evident from the graphs that the two-norm errors do not have high peaks like the fuzzy logic results and are smoother curves than the GA results. The errors could have been smaller if the ANN was trained with more time, or if the number of neurons in the hidden layers was increased. However, it was sufficient to show that the ANN is a better method for the inverse kinematics.

6.3.2. RBF and GRNN

For the Radial Basis Function Networks (RBF), the *newrbe* function was used for the simulation in MATLAB. The function designed an exact RBF quickly. The spread constant was chosen as 1.0, which is a default value, and the networks were trained with 5,000 input/output data sets. The data sets were smaller than other methods because the MATLAB function required large memory space. Therefore, the data sets should have been reduced to meet the memory requirement. As shown in the Appendix C, the maximum error of the Cartesian position was $0.070\,m$, which is the largest error for the 3-DOF manipulator. Furthermore, the number of neurons of RBF was too high. The *newrbe*

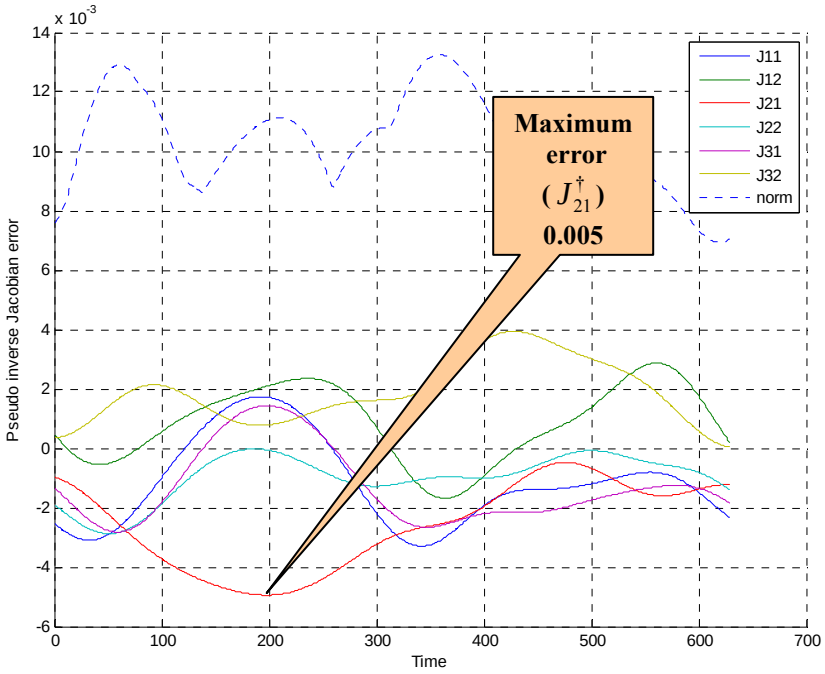

Figure 6-11. Pseudo-inverse Jacobian error for MLP

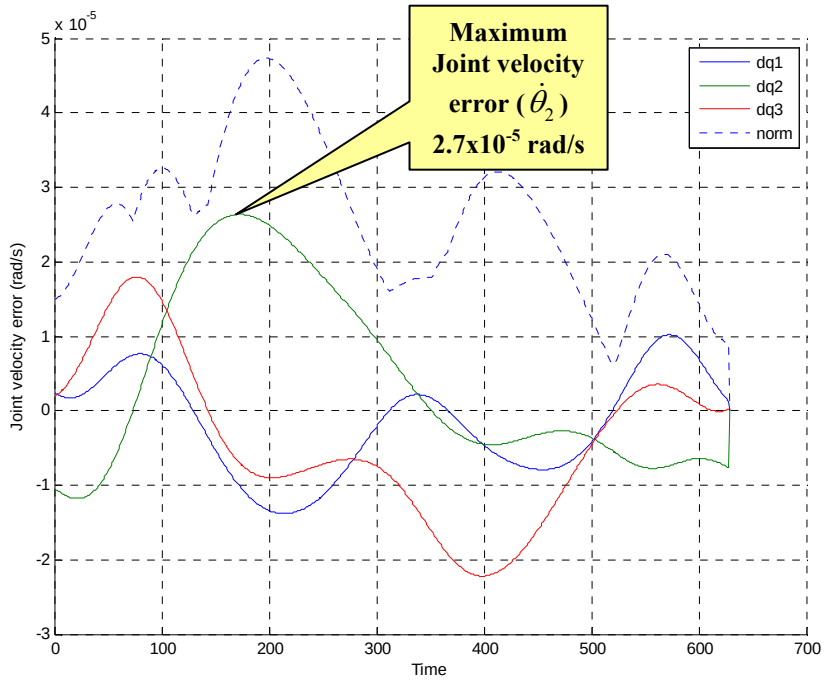

Figure 6-12. Joint velocity error for MLP

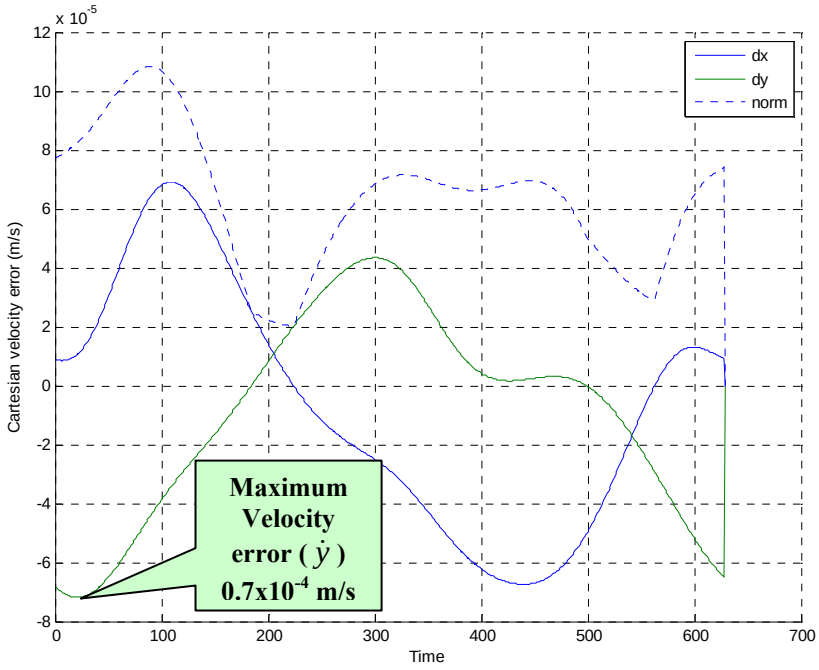

Figure 6-13. Cartesian velocity error for MLP

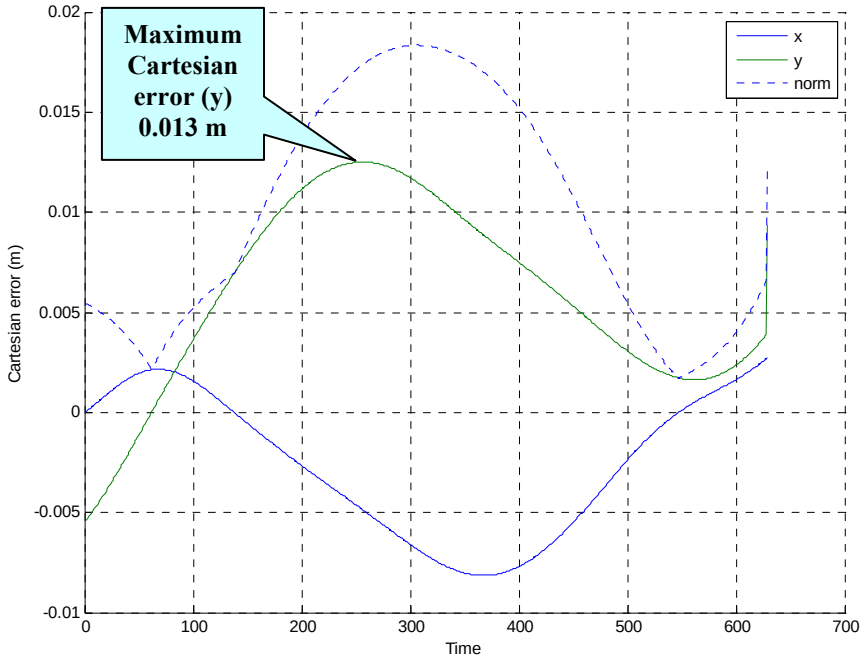

Figure 6-14. Cartesian error for MLP

function generated 5,002 neurons for the simulation. Another RBF function, *newrb*, was used in an effort to reduce the number of neurons, but this function generated nearly 5,000 neurons with poor performance.

For the generalized regression neural network (GRNN), the *newgrnn* function was used with 10,002 data sets. This function designed a GRNN faster than RBF. GRNN is a kind of RBF, so that spread constant was required. The spread has an important role in the design of a GRNN and significantly affects the results. The spread of the GRNN was 0.2, and the results were shown in the Appendix D. The maximum error of the Cartesian position was $0.020\,m$, and the overall errors were small enough to apply to the teleoperation system. However, the number of neurons is the same as the number of the training data set, which was 10,002 neurons. The number of neurons is important for the teleoperation system because the computation time normally depends on the number of neurons. If too many neurons are used, the overall performance is slower. The next section presents results of the five methods which are ANFIS, Fuzzy-GA, MLP, RBF, and GRNN for the 3-DOF manipulator, and discusses approaches and results of final simulation for the 6-DOF Titan II manipulator.

6.4. Results and Final Simulation

6.4.1. Results

In the simulation of the 3-DOF planar manipulator, five artificial intelligent methods were investigated, and the results are shown in Table 6-3. First, the results of the fuzzy logic method with ANFIS and GA showed that it successfully identified the complex

Table 6-3. Results of the 3-DOF Manipulator

Type	Maximum Error			
	Pseudo-inverse Jacobian	Joint Velocity	Cartesian Velocity	Cartesian Position
Fuzzy – ANFIS	0.04	$1.9\times10^{-4}\ rad/s$	$3.3\times10^{-4}\ m/s$	$0.030\,m$
Fuzzy – GA	0.05	$2.8\times10^{-4}\ rad/s$	$5\times10^{-4}\ m/s$	$0.035\,m$
ANN – MLP	0.005	$2.7\times10^{-5}\ rad/s$	$0.7\times10^{-4}\ m/s$	$0.013\,m$
ANN – RBF	0.15	$6\times10^{-4}\ rad/s$	$8\times10^{-4}\ m/s$	$0.070\,m$
ANN – GRNN	0.025	$1.2\times10^{-4}\ rad/s$	$1.8\times10^{-4}\ m/s$	$0.020\,m$

nonlinear inverse kinematics. The overall position and velocity error was minimal. However, the fuzzy logic showed slower performance, which is not sufficient for application in a real system. Furthermore, the artificial neural network method shows better accuracy. The most important reason why the fuzzy logic is not suitable for inverse kinematics is that it was too complicated to apply the fuzzy rule-base to the real teleoperation system.

Second, the results of RBF and GRNN show that the accuracy was not better than MLP, and the networks required many neurons. The number of neurons was same as the number of the input/output data sets. Therefore, RBF and GRNN are not appropriate for the real system. However, RBF and GRNN require less time to build a network than the LM optimization method. They work well if fast computation is not needed or if many data sets are required for training. Therefore, it is a good technique to use RBF or GRNN first for training with many data sets before standard multilayer perceptrons network is trained.

Last, from the above results, a multilayer perceptrons network with LM was determined to be the best solution for the WAM-Titan II teleoperation system. The maximum errors were the smallest among the others, and due to its simple architecture, it is easy to substitute the new inverse kinematics into the current system. However, one drawback of the multilayer feedforward perceptrons network is time consumption. A multilayer perceptrons network needs abundant time to train a network with many data sets, even though LM is used for optimization. Furthermore, LM needs a large amount of memory for approximate Hessian matrix to optimize.

In summary, the multilayer perceptrons method is chosen for a final simulation, which is inverse kinematics of the Titan II manipulator, and the other four methods are excluded due to the above reasons. However, the four methods may have better performance for the Titan II manipulator than the MLP method. On the other hand, because the 6-DOF manipulator for the final simulation is more complicated and has higher dimension for its workspace than the 3-DOF manipulator cases, the chances of this are slight.

6.4.2. Final simulation

For the application of the inverse kinematics for the Titan II based on a multilayer perceptrons - backpropagation artificial neural network, a 6-DOF revolute manipulator was created in MATLAB as shown in Figure 6-15. The manipulator has the same dimensions as that of the Titan II, so that it has same DH parameters. Unlike the 3-DOF manipulator, a new workspace was created in 3-D space. Each joint limit of the Titan II was set for generating training data sets. A new trajectory was created to simulate the

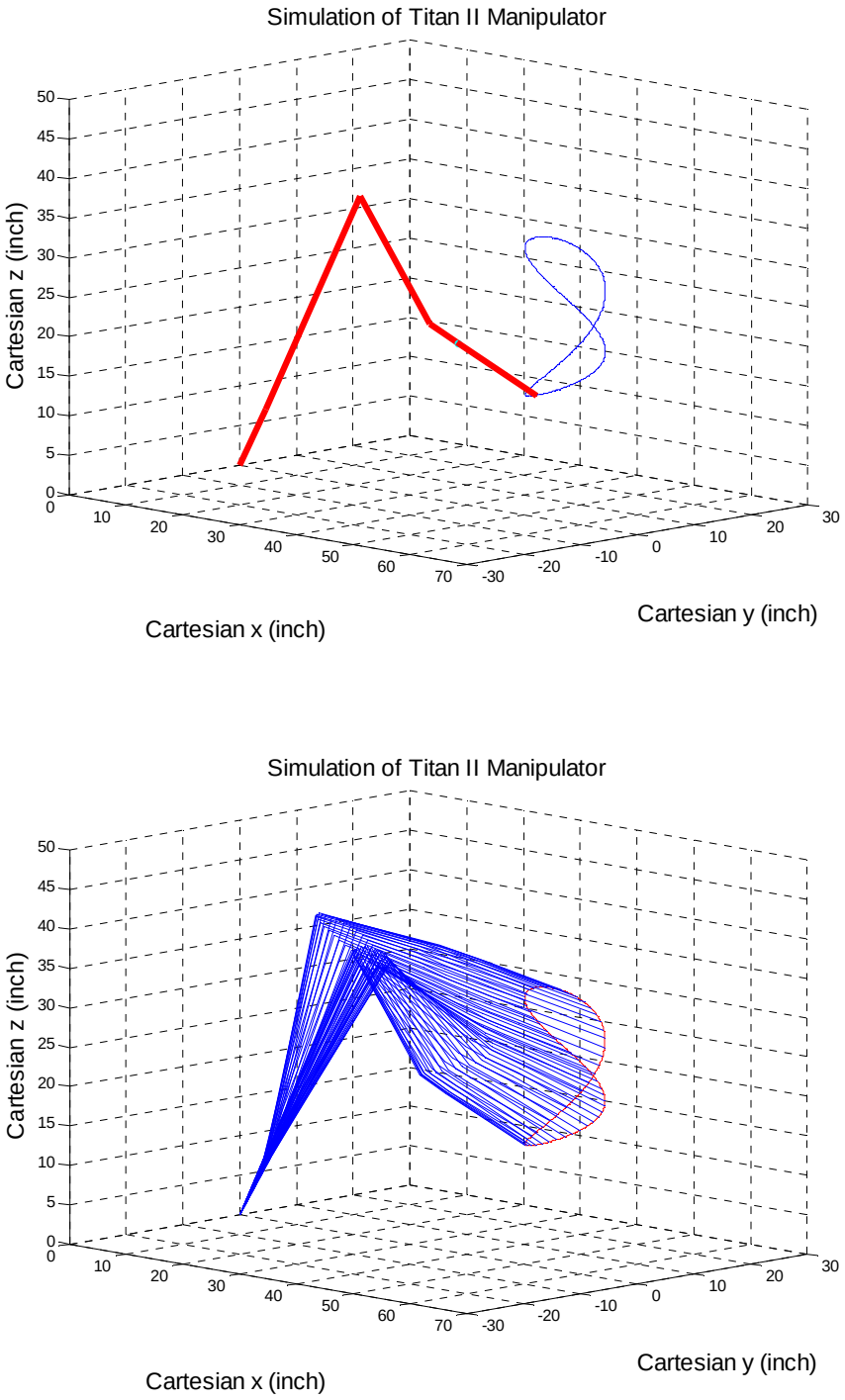

Figure 6-15. Simulation of Titan II manipulator

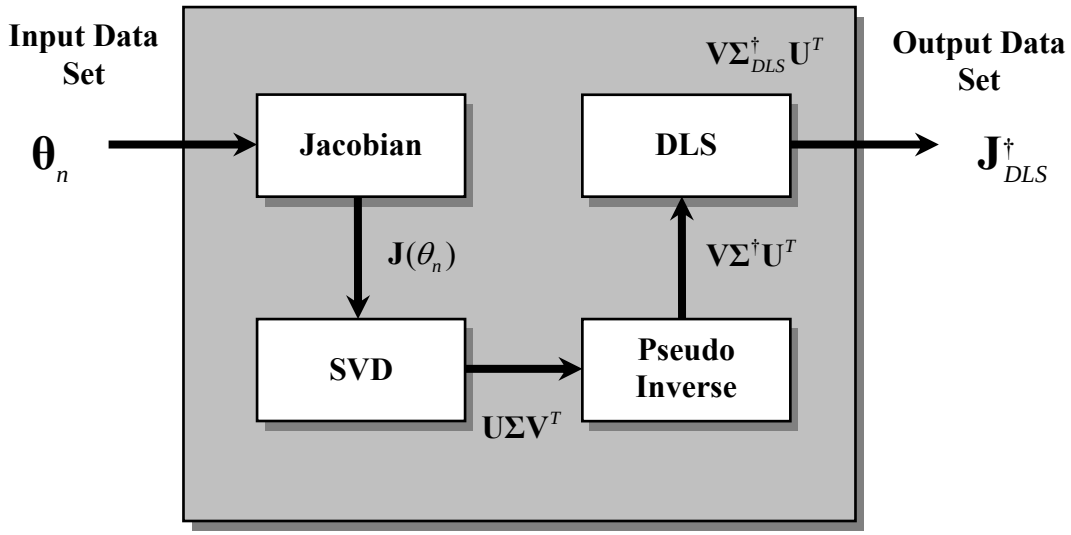

Figure 6-16. Architecture of generating training data sets

new inverse kinematics based on a multilayer perceptrons network, as shown Figure 6-15. The trajectory was more intricate than the previous one because the simulation was tested in 3-D space. New input/output data sets were generated as shown in Figure 6-16. The training data sets were made as 15,000 input/output pairs. The inputs were all possible joint angles of the Titan II, and from the inputs, the outputs were calculated as elements of the pseudo-inverse Jacobian matrix. The architecture of the artificial neural network has four hidden layers with *tansig* activation functions. For optimizing weights and biases, the LM method was used to train quickly, and when memory was insufficient for the large amount of data, memory reduction was used.

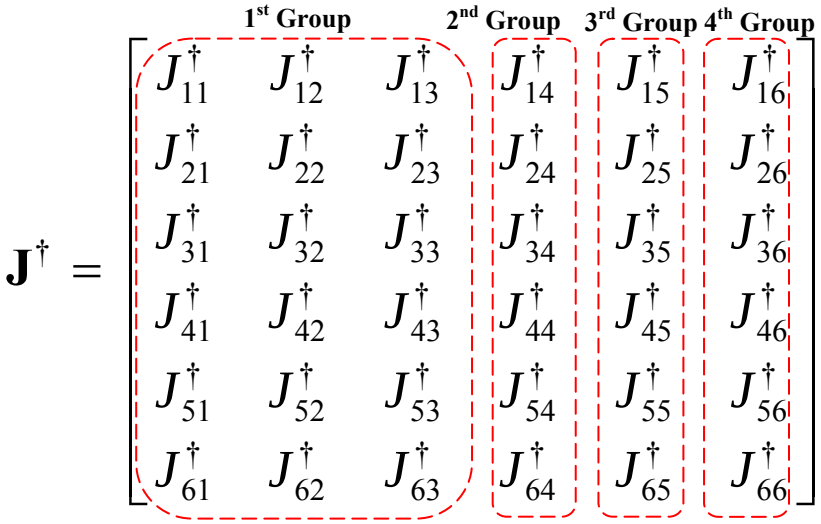

Figure 6-17. Four groups of outputs

The number of elements of the inverse Jacobian matrix is 36, so that the number of outputs is also 36 if one network is used. Since there are many outputs in a network, many neurons were required to meet acceptable SSE. To avoid this, the elements of the inverse Jacobian matrix were divided into four groups as shown in Figure 6-17. The first group is the half of the matrix, which is a position part, and the second to fourth groups are the other half of the matrix, which are rotation parts. Therefore, a total of four networks were used as shown in Figure 6-18. The main reason for this structure is that these groups reduced the total number of neurons in each network, so that memory usage and training time can be reduced greatly. This segmentation method was based on the required training time and the number of neurons. After several experiments, it was found that the position part, which is the first group, is easier to train with less number of neurons than the orientation parts, which are from the second to fourth groups.

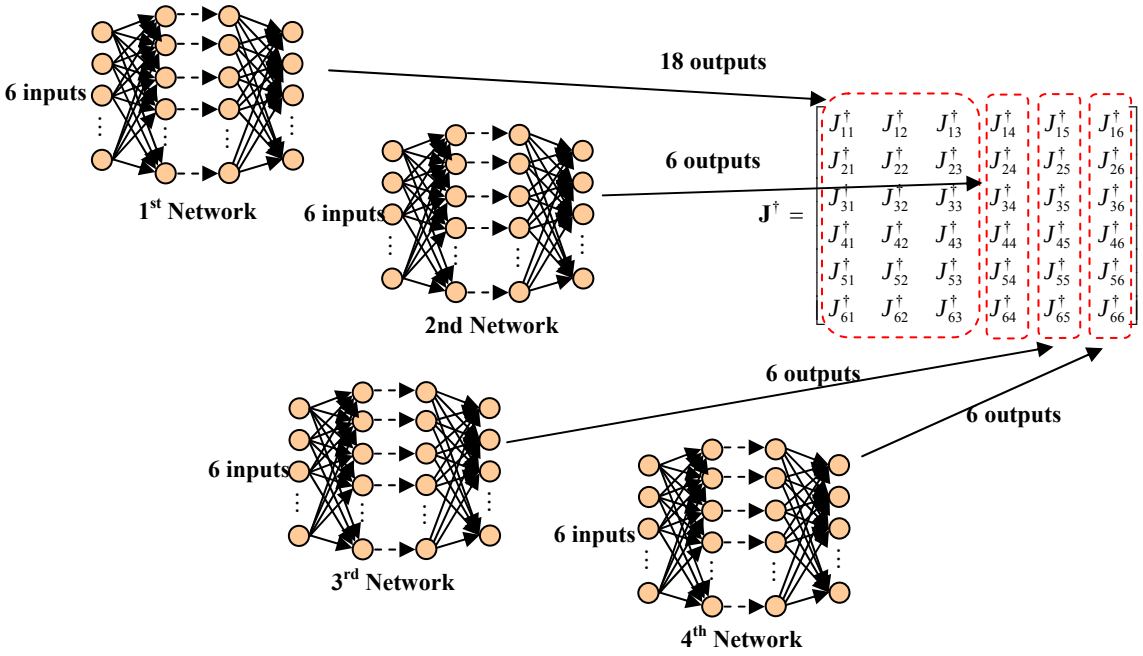

Figure 6-18. Structure of outputs for the MLP network

Furthermore, each element of the pseudo-inverse Jacobian matrix is independent from the other elements due to the fact that the Jacobian depends only on joint angles. Therefore, the structure of Jacobian affects the training time and the execution time by increasing one and decreasing the other. At each hidden layer of the first group, 18 neurons were employed, and at each hidden layer of the last groups, 30 neurons were applied. Consequently, 6 inputs and 18 outputs for the first group and 6 inputs and 6 outputs for the other groups were used.

6.4.3. Results

The final simulation was performed as shown in Figure 6-18. The initial positions of the Titan II were 0°, 60°, -110°, 30°, 0°, 0° from joint 1 to joint 6. The end effector

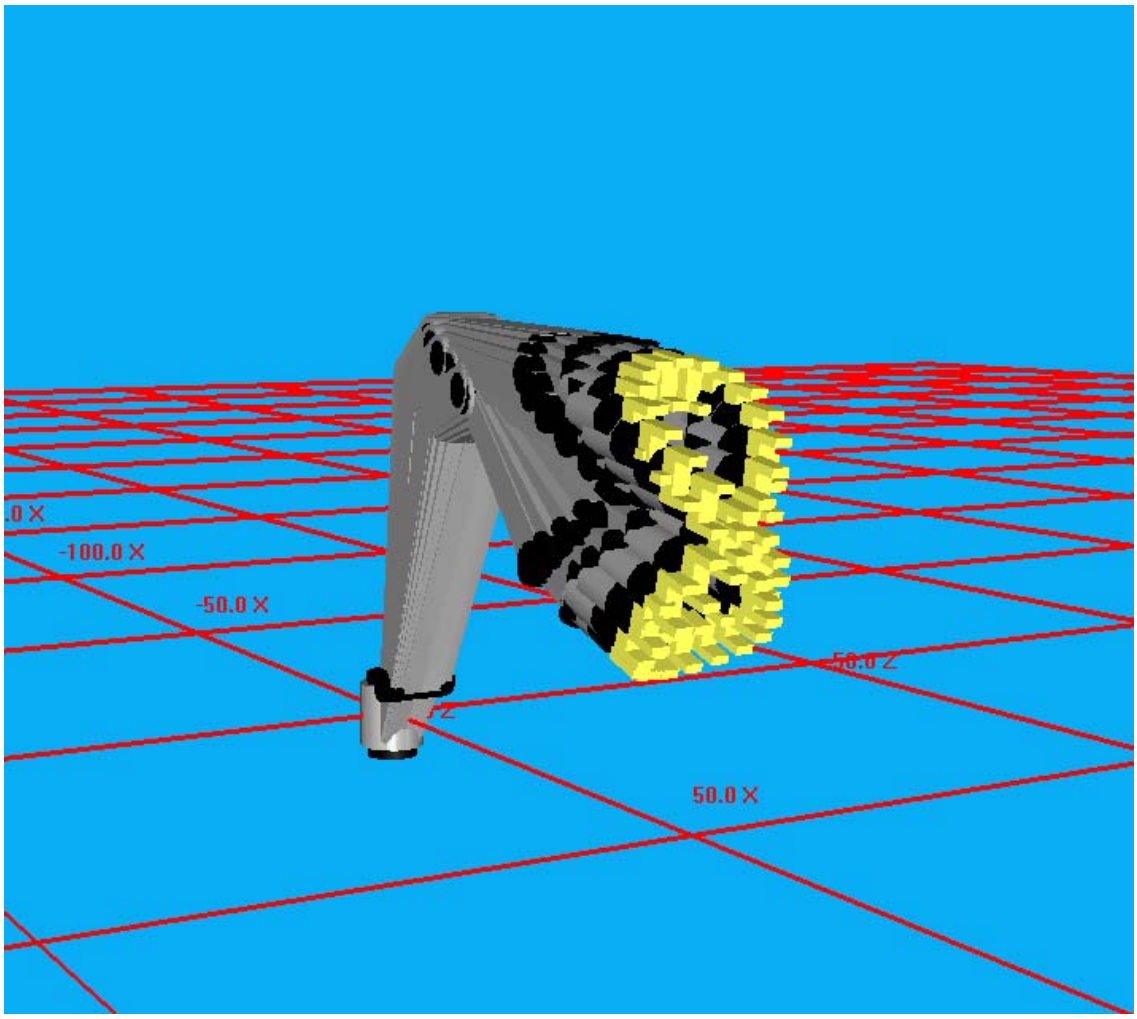

Figure 6-19. RoboWorks simulation for Titan II

followed the trajectory, which carried out position commands only. The unit of the simulation is inches rather than meters. Figure 6-19 shows the Jacobian pseudo-inverse error, and the maximum error was 0.03. As shown in Figure 6-20 and Figure 6-21, the maximum error of the joint velocity and Cartesian velocity are $2.9\times10^{-4}\,rad/\mathrm{s}$ and $0.01\,inch/\mathrm{s}$. The maximum position error is 1.3 inches as shown in Figure 6-22. This inverse kinematics based on artificial neural networks was successfully adapted to the real teleoperation system with the RoboWorks simulation. The MATLAB codes were converted into C programming language as shown in the Appendix E, and the weight and biases were saved to text files. This C code was customized for existing High Level Controller (HLC) of the WAM-Titan II teleoperation system. The C code performs the

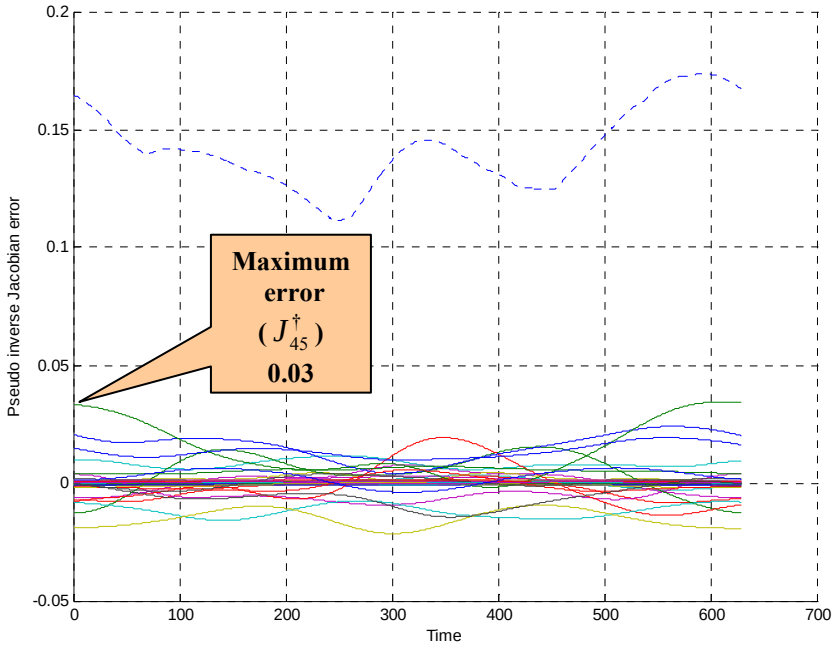

Figure 6-20. Pseudo-inverse Jacobian error for Titan II

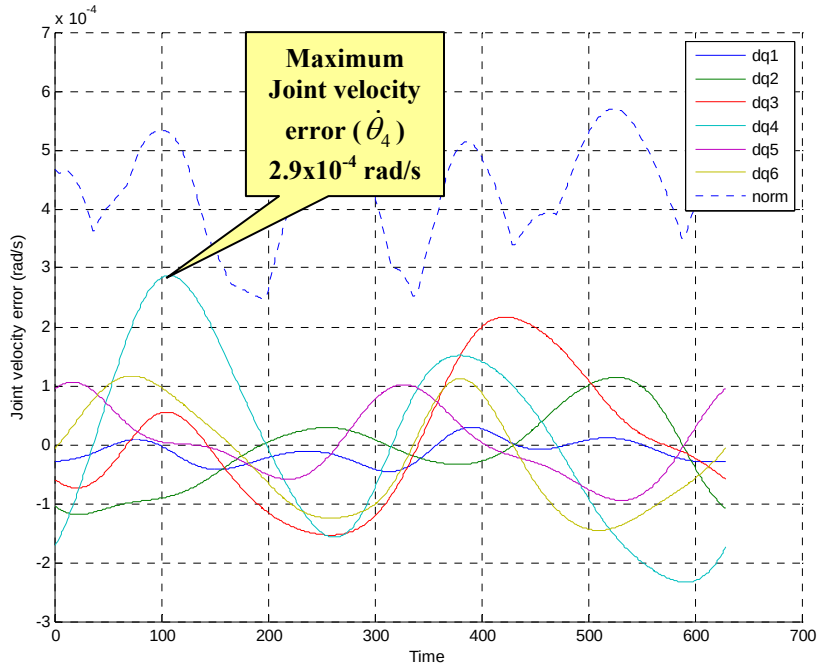

Figure 6-21. Joint velocity error for Titan II

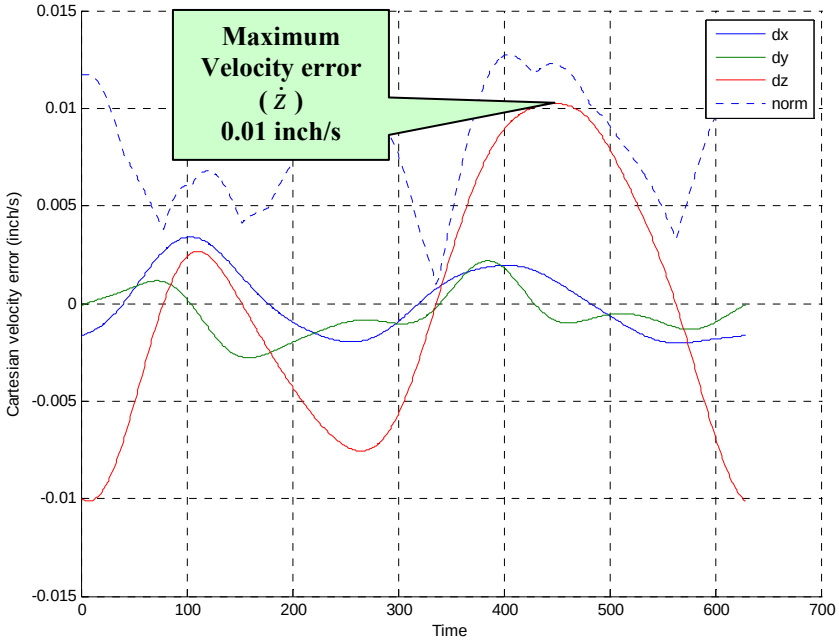

Figure 6-22. Cartesian velocity error for Titan II

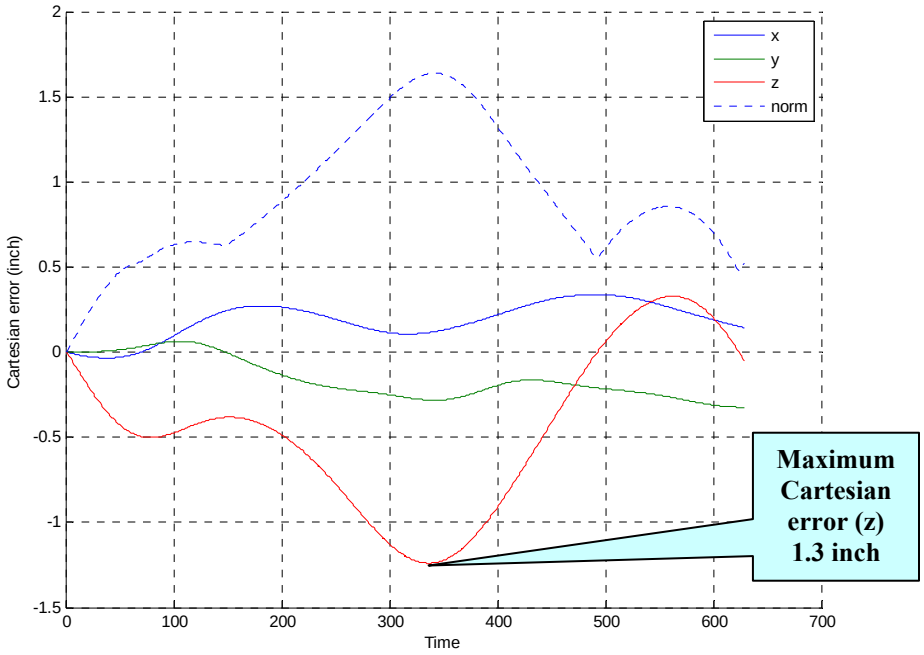

Figure 6-23. Cartesian error for Titan II

algorithm of the MLP and executes the inverse kinematics independently from other applications. In order to improve the execution time of the compile code, every weight and bias of the MLP are stored in 3 dimensional array pointers. Furthermore, a number of *for* statements and *if* statements are reduced to optimize the code. The steps of the main algorithm of the new inverse kinematics shown in Figure 6-24 are

1. Load the weight and bias files.
2. Get joint angles of Titan II from resolvers.
3. Generate a Jacobian pseudo-inverse matrix by ANN-MLP.
4. Perform inverse kinematics with Jacobian pseudo-inverse matrix and Cartesian velocity from WAM forward kinematics.
5. Integrate joint velocities from step 4 and send to Titan II.

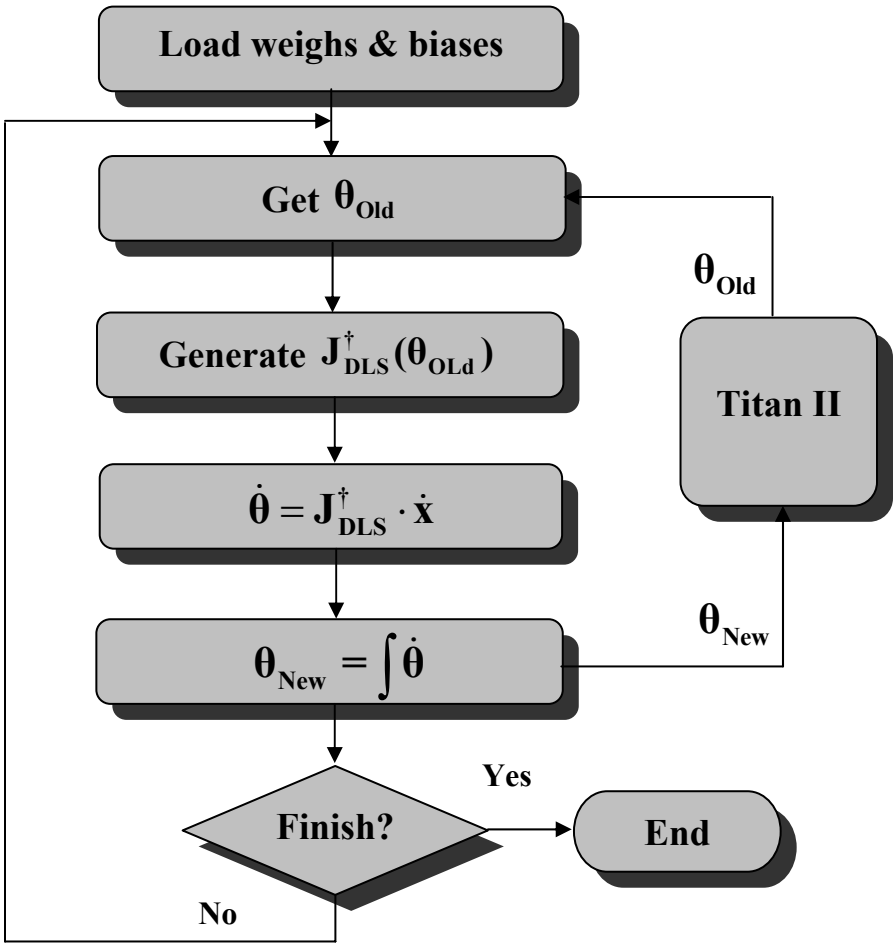

Figure 6-24. Steps of the main algorithm

Table 6-4. Time Results

Method	Trajectory Type		
	Original	**Circle**	**Rectangular**
Conventional	64 ms	45 ms	37 ms
ANN	15.6 ms	11 ms	9.06 ms

6. Repeat from step 2 until the operation is finished.

To measure the time performance for the new inverse kinematics, several trajectories were tested. First, the trajectory used in the above simulation was measured. Only the inverse kinematics time was measured for both the conventional method and the new method during the trajectory tracking. After that, the circular and rectangular trajectories were measured. The Table 6-4 shows the results. About 75 percent of the calculation time was improved. In the next chapter, final conclusions and the future work are discussed.

CHAPTER 7:Summary

7.1. Overall Conclusions

Inverse kinematics based on fuzzy logic and an artificial neural network was designed and implemented for the WAM-Titan II teleoperation system. This inverse kinematics design was based on the pseudo-inverse with SVD and DLS. This strategy automatically reduces the problem of singularities and sudden movements of the slave manipulator while eliminating the weak dimensions by gradually replacing the weak singular value with zero. From the inverse kinematics design, five methods were tested: ANFIS, GA, MLP-LM, RBF, and GRNN. From the simulation of the 3-DOF planar manipulator, MLP-LM was found to be the best method for the inverse kinematics. For the final simulation of the Titan II, MLP-LM was tested, and the results were successful. The maximum error of Cartesian position was 1.3 inches, and this error is acceptable for teleoperation. The computation time of the new inverse kinematics was also faster than that of the normal method. From several trajectory tests, the time was improved about 75 percent.

The downside of the MLP-LM was the computation time for training weights and biases. Normally the LM method is faster than other training methods; however, because of the large input/output data sets and many neurons, abundant computation time was required to meet the acceptable SEE. This negative aspect will be improved as computer

platforms with faster processing speeds are developed in the future. The future work is discussed in the next section.

7.2. Future Work

Although the inverse kinematics using artificial neural networks shows good results, more reliable and accurate results are be desired before performing real teleoperation tasks. For these, more experiments and investigations are required. Because only kinematics was considered in this thesis, analysis of the dynamics of the Titan II are also is needed for simulation of physical motion and design of control strategies.

An important enhancement to the WAM-Titan II teleoperation is an extension to bilateral operation. Force feedback is essential for a teleoperation system to feel the interaction with the remote environment, and it improves the ability of teleoperation. Since the WAM-Titan II teleoperation is ready to move to this stage, extended research of telepresence or haptics with performance control, stability control, and time delay control is required [47].

In inverse kinematics of redundant manipulators, the extra degrees of freedom can be effectively used to improve the manipulator's ability to avoid obstacles or singular points. Since the WAM has seven degrees of freedom, this redundant manipulator can provide a comfortable operational space to a human operator. Therefore, the inverse kinematics of WAM with redundancy resolution based on artificial intelligence methods is another recommended future investigation.

In order to reduce the number of neurons in neural networks, method like Bayesian regularization [45, 46] can be adapted to determine the optimal number of weights and biases automatically. This method modifies the regular performance function such as the mean sum of squared errors by adding the sum of squares of the network weights. Each term of the modified performance function is multiplied by regularization parameters, and the parameters are optimized by the Bayesian regularization. This method provides an optimal number of network parameters, which can be used by the MLP effectively.

LIST OF REFERENCES

LIST OF REFERENCES

[1] L. I. Slutski, "*Remote Manipulation Systems*," Kluwer Academic Publishers, 1998.

[2] W. R. Hamel, M. W. Noakes, "*Recent Telerobotics Systems Developments at the University of Tennessee*," Proceedings of the ANS 2006 International Joint Topical Meeting, Salt Lake City, Utah, February 12–15, 2006.

[3] R. Zhou, W. R. Hamel, A. S. Hariharan, M. W. Noakes, "*Using the WAM as a Master Controller*," Proceedings of the ANS 2006 International Joint Topical Meeting, Salt Lake City, Utah, February 12–15, 2006.

[4] L. H. Tsoukalas, R. E. Uhrig, "*Fuzzy and Neural Approaches in Engineering*," John Wiley & Sons, Inc. 1997.

[5] "*Fuzzy Logic Toolbox user's guide*," The MathWorks, Inc., 2006.

[6] L. Sciavicco, B. Siciliano, "*Modeling and Control of Robot Manipulators*," McGraw Hill Pnublications, 1996.

[7] J. J. Craig, "*Introduction to Robotics Mechanics and Control*," 3rd Edition, Pearson Prentice-Hall, 2005.

[8] M. W. Spong, S. Hutchinson, M. Vidyasagar, "*Robot Modeling and Control*," John Wiley & Sons, Inc., 2006.

[9] W.A. Wolovich, H. Elliott, "*A computational technique for inverse kinematics*," The 23rd IEEE Conference on Decision and Control, Vol. 23, pp. 1359-1363. Dec 1984.

[10] S. R. Buss and J. Kim, "*Selectively Damped Least Squares for Inverse Kinematics*," In Journal of Graphics Tools, vol. 10, no. 3 (2005) 37-49.

[11] C. W. Wampler, "*Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods*," IEEE Trans. on Syst., Man, Cyber., vol. 16, pp. 93-101, 1986.

[12] "*WAM arm User Guide*," Barrett Technology, Inc., 2006.

[13] D. W. Howard, A. Zilouchian, "*Application of Fuzzy Logic for the Solution of Inverse Kinematics and Hierarchical Controls of Robotic Manipulators*," Journal of Intelligent and Robotic Systems, vol. 23, No. 2-4, pp. 217 – 247, October 1998.

[14] D. E. Goldberg, "*Genetic Algorithms in Search, Optimization, and Machine Learning*," Addison-Wesley, 1989.

[15] F. Herrera, M. Lozano, J. L. Verdegay, "*Tuning Fuzzy Logic Controllers by Genetic Algorithms*," International Journal of Approximate Reasoning," vol. 12, pp. 299-315, 1995.

[16] M. A. Lee, Takagi, "*Integrating Design Stages of Fuzzy Systems Using Genetic Algorithms*," Proc. 2nd IEEE Int. conf. Fuzzy systems, San Francisco, 1993.

[17] Y. J. Park, H. S. Cho, D. H. Cha, "*Genetic Algorithm-Based Optimization of Fuzzy Logic Controller Using Characteristic Parameters*," Proceedings of the IEEE ICEC, pp. 831-836, 1995.

[18] "*Neural Network toolbox user's guide*," The MathWorks, Inc., 2006.

[19] G. A. Bekey, K. Y. Goldberg, "*Neural Networks in Robotics*," Kluwer Academic Publishers, 1993.

[20] J. L. Meriam, J. M. Henderson, "*Engineering Mechanics Dynamics*," 4th Edition, John Wiley & Sons, Inc., 1997.

[21] Gilbert Strang, "*Linear Algebra and its Application*," 4th Edition, Academic Press, New York, 2006.

[22] J. –J E. Slotine, "*Putting physics in control-the example of robotics*," Control Systems Magazine, IEEE, Vol. 8, No 6, pp. 12-18, Dec 1988.

[23] Y. Nakamura, H. Hanafusa, "*Inverse kinematic solution with singularity robustness for robot manipulator control*," ASME J. Dyn. Syst., Meus., Control, vol. 108, pp. 163-171, 1986.

[24] S. W. Kim, J. J. Lee, "*Inverse Kinematics Solution Based on Fuzzy Logic for Redundant Manipulators*," Proceedings of the 1993 IEEE/RSJ International Conference on, Vol. 2, No. 26-30, pp. 904 – 910, Jul 1993.

[25] C. R. Houck, J. Joines, M. Kay, "*A Genetic Algorithm for Function optimization: A MATLAB Implementation*," ACM Transactions on Mathematical Software, 1996.

[26] K. Levenberg, "*A Method for the Solution of Certain Non-Linear Problems in Least Squares*," Quart. Appl. Math. 2, pp. 164-168, 1944.

[27] D. Marquardt, "*An Algorithm for Least-Squares Estimation of Nonlinear Parameters*," SIAM J. Appl. Math. 11, pp. 431-441, 1963.

[28] L. A. Zadeh, "*Information and Control*," Vol. 8, pp. 338-353, 1965.

[29] M. Mamdani, "*Application of Fuzzy Algorithm for Control of Simple Dynamic Plant*," Proc. IEE, Vol. 121, No. 12, pp. 1585-1588, 1974.

[30] H. T. Nguyen, E. A. Walker, "*A First Course in Fuzzy Logic*," 3rd Edition, Chapman & Hall/CRC, Boca Raton, 2006.

[31] J.-S. Roger Jang, "*ANFIS: Adaptive-Network-Based Fuzzy Inference Systems*," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 03, pp. 665-685, May 1993.

[32] J. H. Holland, "*Adaptation in Natural and Artificial Systems*," University of Michigan Press, Ann Arbor, 1975.

[33] D. Nauck, F. Klawonn, R. Kruse, "*Foundations of Neuro-Fuzzy Systems*," John Wiley & Sons Ltd, 1997.

[34] F. L. Lewis, S. Jagannathan, A. Yeşildirek, "*Neural Network Control of Robot Manipulators and Nonlinear Systems*," Taylor & Francis, 1999.

[35] G. A. Korn, "*Neural Networks and Fuzzy-Logic Control on Personal Computers and Workstations*," Massachusetts Institute of Technology, 1995.

[36] A. M. S. Zalzala, A. S. Morris, "*Neural Networks for Robotic Control*," Ellis Horwood, 1996.

[37] S. Haykin, "*Neural Networks*," IEEE Press and Macmillan, New York, 1994.

[38] F. Girosi, T. Poggio, "*Neural Networks and the Best Approximation Property*," Biol. Cybernetics, 63, pp. 169-176, 1990.

[39] P. D. Wasserman, "*Advanced Methods in Neural Computing*," New York, 1993.

[40] A. Nedungadi, "*A Fuzzy Robot Controller – Hardware Implementation*," Fuzzy Systems, IEEE International Conference on, pp. 1325-1331, Mar 1992.

[41] L. Wei, H. Wang, Y. Li, "*A New Solution for Inverse Kinematics of Manipulator Based on Neural Network*," Machine Learning and Cybernetics, International Conference on, Vol. 2, No. 2-5, pp. 1201 – 1203, Nov. 2003.

[42] A. M. Eydgahi, S. Ganesan, "*Genetic-Based fuzzy Model for Inverse Kinematics Solution of Robotic Manipulators*," Systems, Man, and Cybernetics, IEEE International Conference on, Vol. 3, pp. 2196 – 2201, Oct 1998.

[43] R. V. Mayorga, P. Sanongboon, "*Inverse Kinematics and Geometrically Bounded Singularities Prevention of Redundant Manipulators: An Artificial Neural Network Approach*," Robotics and Autonomous Systems, 53, pp. 164-176, 2005.

[44] J. Guo, V. Cherkassky, "*A Solution to the Inverse Kinematic Problem in Robotics Using Neural Network Processing*," Neural Networks, IJCNN., International Joint Conference on, Vol. 2, pp. 299 – 304, Jun 1989.

[45] D.J.C. MacKay, "*Bayesian interpolation*," Neural Computation, Vol. 4, No. 3, pp. 415–447, 1992.

[46] D.J.C. MacKay, "*A Practical Bayesian Framework for Backpropagation Networks*," Neural Computation, Vol. 4, No. 3, pp. 448-472, 1992.

[47] D. A. Lawrence, "*Stability and Transparency in Bilateral Teleoperation*," IEEE Transactions on Robotics and Automation, Vol. 9, No. 5, pp.624~637, 1993.

APPENDIX

Appendix A: Transformation Matrix for WAM

From frame 1 to frame 2

$$\mathbf{A}_2^1 = \begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) & 0 \\ \sin(\theta_2) & 0 & -\cos(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 2 to frame 3

$$\mathbf{A}_3^2 = \begin{bmatrix} \cos(\theta_3) & 0 & -\sin(\theta_3) & 0.045\cos(\theta_3) \\ \sin(\theta_3) & 0 & \cos(\theta_3) & 0.045\sin(\theta_3) \\ 0 & -1 & 0 & 0.55 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 3 to frame 4

$$\mathbf{A}_4^3 = \begin{bmatrix} \cos(\theta_4) & 0 & \sin(\theta_4) & 0.4\cos(\theta_4) \\ -\sin(\theta_4) & 0 & \cos(\theta_4) & 0.4\sin(\theta_4) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 4 to frame 5

$$\mathbf{A}_5^4 = \begin{bmatrix} \cos(\theta_5) & 0 & -\sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & -\cos(\theta_5) & 0 \\ 0 & 1 & 0 & 0.1547 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 5 to frame 6

$$\mathbf{A}_6^5 = \begin{bmatrix} \cos(\theta_6) & 0 & -\sin(\theta_6) & 0 \\ \sin(\theta_6) & 0 & \cos(\theta_6) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 6 to frame 7

$$\mathbf{A}_7^6 = \begin{bmatrix} \cos(\theta_7) & -\sin(\theta_7) & 0 & 0 \\ \sin(\theta_7) & \cos(\theta_7) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Appendix B: Transformation Matrix for Titan II

From frame 1 to frame 2

$$\mathbf{A}_2^1 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & d_2\sin(\theta_2)+a_2\cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & -d_2\cos(\theta_2)+a_2\sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 2 to frame 3

$$\mathbf{A}_3^2 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & a_3\cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) & 0 & a_3\sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 3 to frame 4

$$\mathbf{A}_4^3 = \begin{bmatrix} \cos(\theta_4) & 0 & -\sin(\theta_4) & a_4\cos(\theta_4) \\ \sin(\theta_4) & 0 & \cos(\theta_4) & a_4\sin(\theta_4) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 4 to frame 5

$$\mathbf{A}_5^4 = \begin{bmatrix} \cos(\theta_5) & 0 & \sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & -\cos(\theta_5) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From frame 5 to frame 6

$$\mathbf{A}_6^5 = \begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & 0 \\ \sin(\theta_6) & \cos(\theta_6) & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Appendix C: 3-DOF Planar Manipulator Simulation - RBF

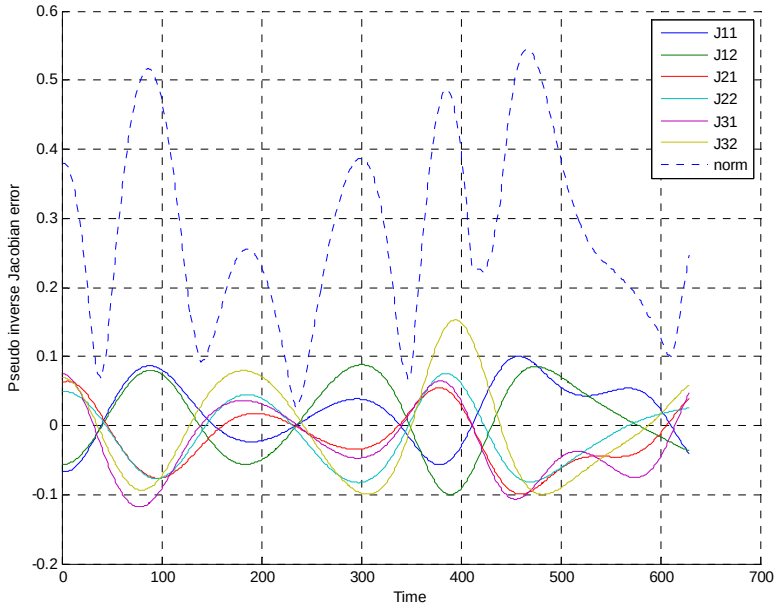

Figure C-1 RBF Pseudo-inverse Jacobian error

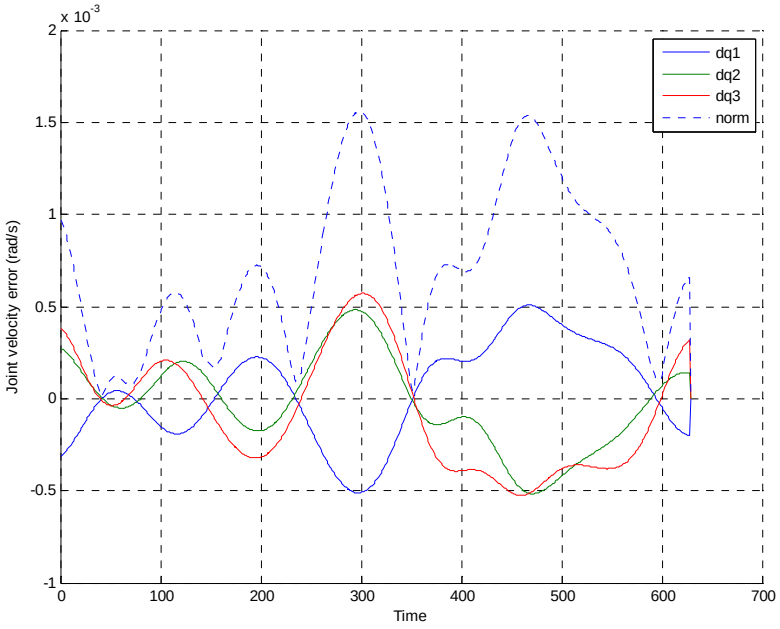

Figure C-2 RBF Joint velocity error

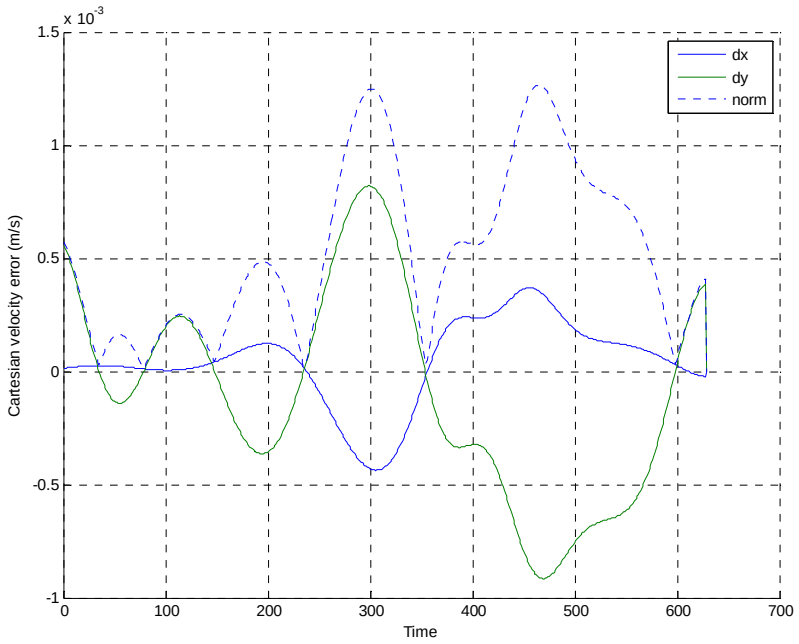

Figure C-3 RBF Cartesian velocity error

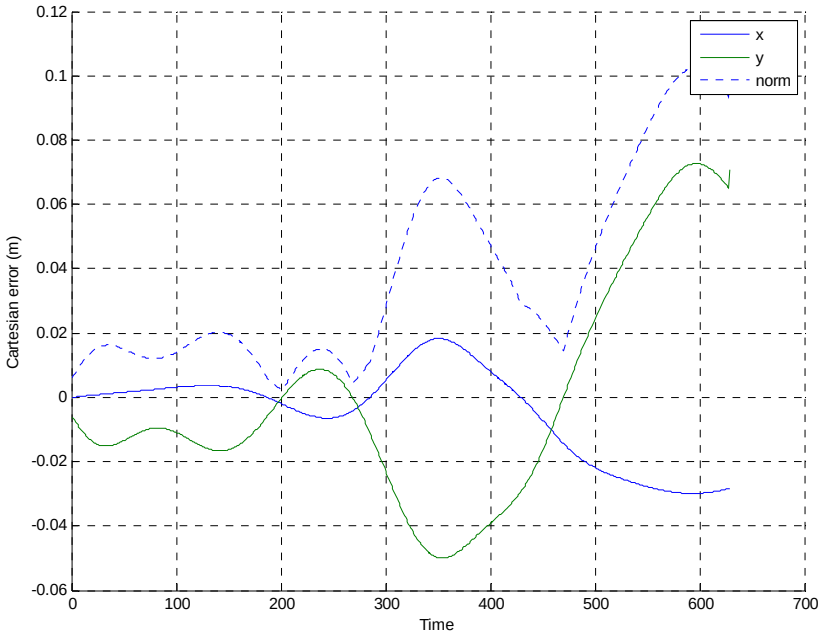

Figure C-4 RBF Cartesian error

Appendix D: 3-DOF Planar Manipulator Simulation - GRNN

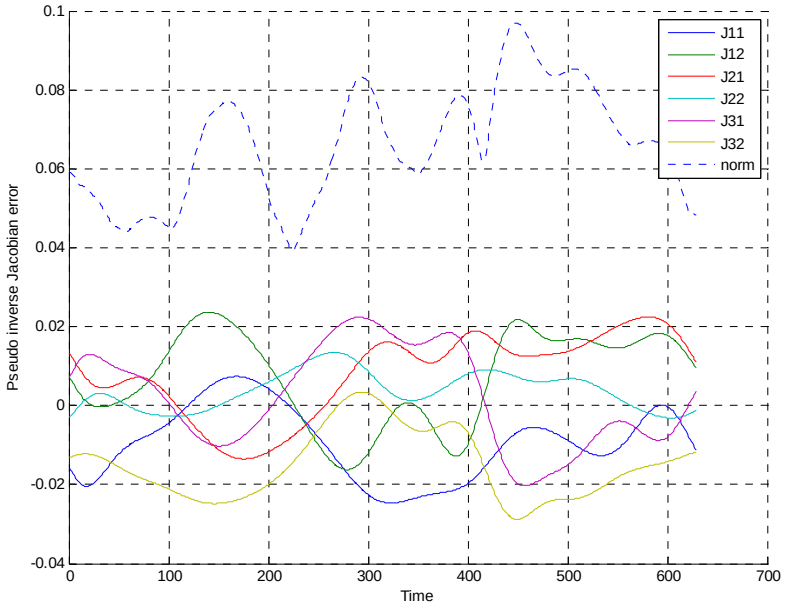

Figure D-1 GRNN Pseudo-inverse Jacobian error

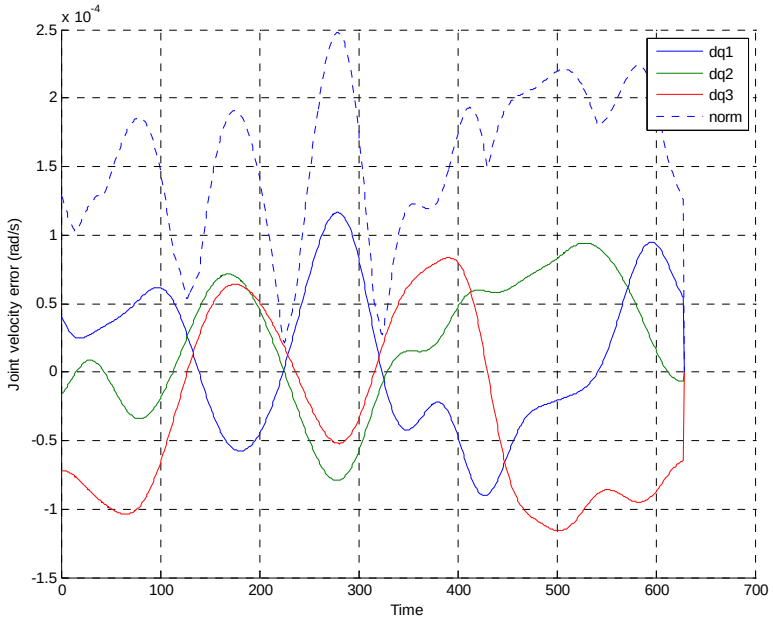

Figure D-2 GRNN Joint velocity error

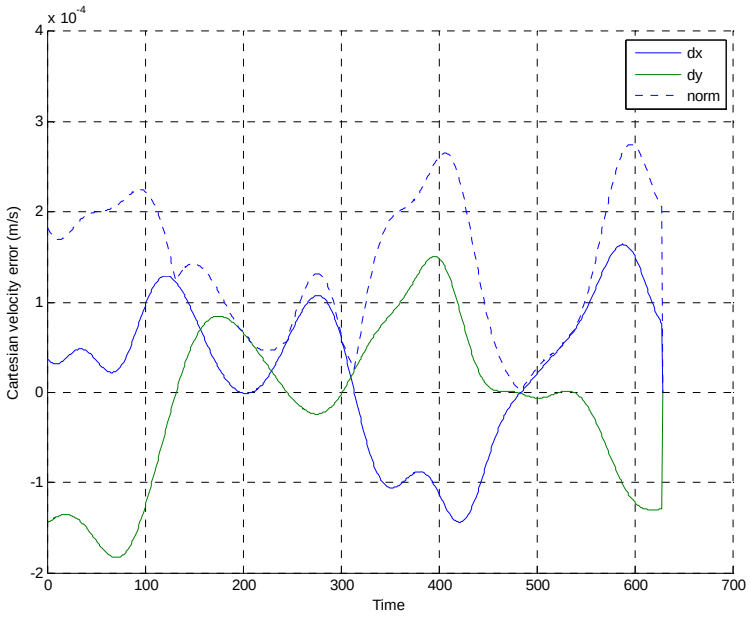

Figure D-3 GRNN Cartesian velocity error

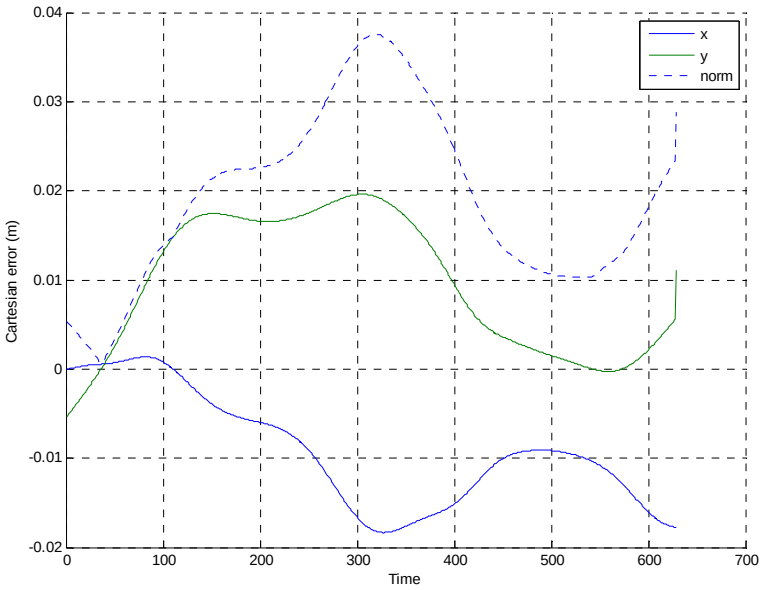

Figure D-4 GRNN Cartesian error

Appendix E: Programming Code of MLP for Titan II

```
/* File name: ANN_math.h                    */
/* Header file for ANN_math.c               */


#ifndef _ANN_MATH_
#define _ANN_MATH_

double ***ANNnew3dMatrix(int num, int nor, int noc);
void free_3dmatrix(double ***pMatrix, int num, int nor);
void ANNprintMatrix(double **a, int rows, int cols,int flag);
void ANNprintVector(double *a, int length,int flag);
void ANNmvDotProduct(double **a,double *b, int row, int col,int length, double *c);
void ANNtansig(double *pResult, double *pMatrix, int length);
void ANNint_matrix(double **pMatrix, int nor, int noc);
void ANNint_vector(double *pVector, int length);
void ANNvectorAddition(double *a,double *b,int length1,int length2,double *c);
void ANNvectorCopy(double *a, double *b, int length);
void ANNload_wb_files(double ***pmW,double **pmB);
void ANNbp_simul(double *p, double ***w, double **b,int non, int nout, int nlayer, double *ans, int z);
void ANNmain_simul(double *p, double ***pmW, double **pmB, double **pPseudo);

#endif
```

```
/* File name: ANN_math.c                        */



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "ANN_math.h"
#include "titanmath.h"

double ***ANNnew3dMatrix(int num, int nor, int noc)
{

        double ***p3dMatrix;
        int i, j, k;

        if ((p3dMatrix=(double ***)malloc(num*sizeof(int **)))==NULL) {
                printf("malloc error\n");
                exit(-1);
        }
        for (i=0;i<num;i++)
                if ((p3dMatrix[i]=(double **)malloc(nor*sizeof(int *)))==NULL) {
                        printf("malloc error\n");
                        exit(-1);
                }
        for (i=0;i<num;i++)
                for (j=0;j<nor;j++)
                        if ((p3dMatrix[i][j]=(double *)malloc(noc*sizeof(double)))==NULL) {
                                fprintf(stderr, "out of memory\n");
                                exit(-1);
                        }

        for (i=0;i<num;i++)
                for (j=0;j<nor;j++)
                        for (k=0;k<noc;k++)
                                p3dMatrix[i][j][k]=0;

        return p3dMatrix;
}

void free_3dmatrix(double ***pMatrix, int num, int nor)
{

        int i, j;

        for(i=0;i<num;i++){
                for(j=0;j<nor;j++){
                        if(pMatrix[i][j]!= NULL)
                                free(pMatrix[i][j]);
                }
        }

        for(i=0;i<num;i++){
                 if(pMatrix[i]!= NULL)
```

```
                free(pMatrix[i]);
        }
        if(pMatrix!=NULL)
                free(pMatrix);

}

void ANNprintMatrix(double **a, int rows, int cols,int flag)
{

        int i,j;

        if(flag) {
                printf("[row column\n");
                for(i=0;i<rows;i++) {
                        for(j=0;j<cols;j++)
                                printf("%3d %3d %12.8lf\n",i+1,j+1,a[i][j]);
                        getchar();

                }
                printf("]\n");
        }

        else{
                printf("[\n");
                for(i=0;i<rows;i++) {
                        for(j=0;j<cols;j++)
                                printf("%7.4lf, ",a[i][j]);
                        printf("\n");
                }
                printf("]\n");
        }
}

void ANNprintVector(double *a, int length,int flag)
{

        int i;

        if(flag) {
                printf("[\n");
                for(i=0;i<length;i++)
                        printf("%lf",a[i]);
                printf("]\n");
        }

        else{
                printf("[\n");
                for(i=0;i<length;i++)
                        printf("%d %12.8lf\n",i+1,a[i]);
                printf("]\n");
        }
}
```

```
void ANNmvDotProduct(double **a,double *b, int row, int col,int length, double *c)
{
	int i,j;
	if(col!=length){
		printf("check the matrix and the vector length!\n");
		exit(1);
	}
	for(i=0;i<row;i++){
		c[i]=0.0;
		for(j=0;j<col;j++){
			c[i]=c[i]+a[i][j]*b[j];
		}
	}
}


void ANNtansig(double *pResult, double *pMatrix, int length)
{
	int i;

	for(i=0;i<length;i++)
		pResult[i]=2/(1+exp(-2*pMatrix[i]))-1;

}

void ANNint_matrix(double **pMatrix, int nor, int noc)
{
	int i,j;

	for(i=0;i<nor;i++)
		for(j=0;j<noc;j++)
			pMatrix[i][j]=0;

}

void ANNint_vector(double *pVector, int length)
{
	int i;

	for(i=0;i<length;i++)
		pVector[i]=0;

}

void ANNvectorAddition(double *a,double *b,int length1,int length2,double *c)
{

	int i;
	if(length1!=length2){
		printf("check vector length!\n");
		exit(1);
	}
	for(i=0;i<length1;i++){
		c[i]=a[i]+b[i];
```

```
		}
}

void ANNvectorCopy(double *a, double *b, int length)
{
 int i;
 for(i=0;i<length;i++)
		b[i]=a[i];

}

void ANNload_wb_files(double ***pmW,double **pmB)
{
		FILE *fp1, *fp2;
		double *pvW, *pvB;

		int lenw=10260;
		int lenb=450;
		int tw=0, tb=0;

		pvW=new_vector(lenw);
		pvB=new_vector(lenb);

		ANNint_vector(pvW,lenw);
		ANNint_vector(pvB,lenb);

		int i=0, j=0, k=0, h=0, l=0;

		fp1=fopen("weights.txt","r");
		if (fp1==NULL) {
			printf("I couldn't open a txt file for reading.\n");
			getchar();
			exit(0);
		}

		fp2=fopen("biases.txt","r");
		if (fp2==NULL) {
			printf("I couldn't open a txt file for reading.\n");
			getchar();
			exit(0);
		}


		while(fscanf(fp1, "%lf\n", &pvW[i]) == 1) {
			i=i+1;
		}

		while(fscanf(fp2, "%lf\n", &pvB[j]) == 1) {
			j=j+1;
		}


		fclose(fp1);
		fclose(fp2);
```

```
for(j=0;j<18;j++){
        for(k=0;k<6;k++){
                pmW[0][j][k]=pvW[tw];
                tw=tw+1;
        }
}
for (i=1;i<4;i++){
        for(j=0;j<18;j++){
                for(k=0;k<18;k++){
                        pmW[i][j][k]=pvW[tw];
                        tw=tw+1;
                }
        }
}
i=4;
for (h=0;h>3;h++){
        for(j=0;j<30;j++){
                for(k=0;k<6;k++){
                        pmW[i][j][k]=pvW[tw];
                        tw=tw+1;
                }
        }
        i=i+1;
        for (l=0;l<3;l++){
                for(j=0;j<30;j++){
                        for(k=0;k<30;k++){
                                pmW[i][j][k]=pvW[tw];
                                tw=tw+1;
                        }
                }
                i=i+1;
        }
        for(j=0;j<6;j++){
                for(k=0;k<30;k++){
                        pmW[i][j][k]=pvW[tw];
                        tw=tw+1;
                }
        }
        i=i+1;
}

for(i=0;i<4;i++){
        for(j=0;j<18;j++){
                pmB[i][j]=pvB[tb];
                tb=tb+1;
        }
}
i=4;
for(h=0;h<3;h++){
        for(l=0;l<4;l++){
                for(j=0;j<30;j++){
                        pmB[i][j]=pvB[tb];
                        tb=tb+1;
```

```
                        }
                        i=i+1;
                }
                for(j=0;j<6;j++){
                        pmB[i][j]=pvB[tb];
                        tb=tb+1;
                }
                i=i+1;
        }

        free(pvW);
        free(pvB);

}

void ANNbp_simul(double *p, double ***w, double **b,int non, int nout, int nlayer, double *ans, int z)
{

        int i=0, j=0, k=0, h=0;

        double *c, *d, *e, *e2;
        c=new_vector(non);
        d=new_vector(non);
        e=new_vector(non);
        e2=new_vector(non);
        ANNint_vector(c,non);
        ANNint_vector(d,non);
        ANNint_vector(e,non);
        ANNint_vector(e2,non);


        for(i=0;i<non;i++){
                c[i]=0.0;
                for(j=0;j<6;j++){
                        c[i]=c[i]+w[z][i][j]*p[j];
                }
                d[i]=c[i]+b[z][i];
                e[i]=2/(1+exp(-2*d[i]))-1;
        }
        z=z+1;
        for(h=0;h<nlayer-1;h++){
                for(i=0;i<non;i++){
                        c[i]=0.0;
                        for(j=0;j<non;j++){
                                c[i]=c[i]+w[z][i][j]*e[j];
                        }
                        d[i]=c[i]+b[z][i];
                        e2[i]=2/(1+exp(-2*d[i]))-1;
                }
                z=z+1;
                for(k=0;k<non;k++)
                        e[k]=e2[k];
        }
        for(i=0;i<nout;i++){
```

```
                c[i]=0.0;
                for(j=0;j<non;j++){
                        c[i]=c[i]+w[z][i][j]*e[j];
                }
                d[i]=c[i]+b[z][i];
                ans[i]=2/(1+exp(-2*d[i]))-1;

        }
        z=z+1;

        free(c);
        free(d);
        free(e);
        free(e2);

}

void ANNmain_simul(double *p, double ***pmW, double **pmB, double **pPseudo)
{
        int i=0, j=0, k=0, l=0;

        double *ans1, *ans2, *ans3, *ans4;
        ans1=new_vector(18);
        ans2=new_vector(6);
        ans3=new_vector(6);
        ans4=new_vector(6);

        ANNint_vector(ans1,18);
        ANNint_vector(ans2,6);
        ANNint_vector(ans3,6);
        ANNint_vector(ans4,6);


        ANNbp_simul(p, pmW, pmB, 18, 18, 3, ans1, 0);
        ANNbp_simul(p, pmW, pmB, 30, 6, 4, ans2, 4);
        ANNbp_simul(p, pmW, pmB, 30, 6, 4, ans3, 9);
        ANNbp_simul(p, pmW, pmB, 30, 6, 4, ans4, 14);


        for (j=0;j<6;j++){
                for (k=0;k<3;k++){
                        pPseudo[j][k]=ans1[i];
                        i=i+1;
                }
        }

        for (j=0;j<6;j++){
                pPseudo[j][3]=ans2[l];
                pPseudo[j][4]=ans3[l];
                pPseudo[j][5]=ans4[l];
                l=l+1;
        }
```

```
        free(ans1);
        free(ans2);
        free(ans3);
        free(ans4);

}
```

Vita

Joong-kyoo Park was born in South Korea in 1971, and had lived in Seoul. He entered Soonchunhyang University in 1991, and received his Bachelor of Science degree in Physics 1995. In the same year he was commissioned as a second lieutenant in Korean army. He entered the undergraduate school in Mechanical Engineering at Virginia Polytechnic Institute and State University in 1999, and received another Bachelor of Science degree in Mechanical Engineering. After graduation in 2002, he joined the Daesan Metal Corporation, and had work for two years. He entered the graduate school in Mechanical Engineering at the University of Tennessee in 2005. He is glad to receive the master degree in 2007.