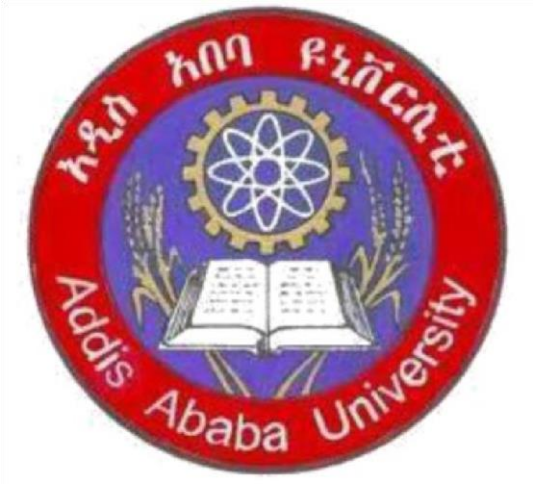

ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Vision Based Robot Control Using Machine Learning

By
Sabizer Birihanu

A Thesis Submitted to the School of Electrical and Computer Engineering of Addis Ababa Institute of Technology, School of Graduate Studies, Addis Ababa University in partial fulfillment of the Requirement for the Degree of Masters of Science in Control Engineering

Advisor: Dereje Shiferaw (Ph.D.)

June, 2019
Addis Ababa, Ethiopia

ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

VISION BASED ROBOT CONTROL USING MACHINE LEARNING

By Sabizer Birhanu G/abe

Approved By Board of Examiners

_____	_____
Dean,	**Signature**
School of Electrical and Computer Engineering	
Dr. Derje Sheferaw	_____
Advisor	**Signature**
_____	_____
External Examiner	**Signature**
_____	_____
Internal Examiner	**Signature**

ACKNOWLEDGMENT

I thank all who in one way or another contributed in the completion of this thesis. First, thanks to Almighty God for protection and ability to do this work.

I am so grateful to the Female scholarship given by Addis Ababa University for making it possible for me to study here. I give deep thanks to the Dr.'s and lecturers at Addis Ababa Information Technology Institute, Electrical Department, the librarians, and other workers of the faculty. My special and heartily thanks to my advisor, Dr. Derje Sheferaw who encouraged and directed me. His challenges brought this work towards a completion. It is with his supervision that this work came into existence.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

DECLARATION

I, the undersigned, declare that this thesis is my original work, and has not been presented for a degree in this or other universities, and all sources of materials used for this thesis work have been fully acknowledged.

Name: **SABIZER BIRIHANU**

Signature: _____

Place: Addis Ababa Institute of Technology, Addis Ababa University, Addis Ababa, Ethiopia

This thesis has been submitted for examination with my approval as a university advisor.

Dereje Shiferaw (Ph.D.) _____

Advisor's Name Signature

Abstract

A new simpler vision based robot control system is proposed characterized with position specific artificial neural network (ANN) and end-effecter integrated camera system. Position specific ANN avoids the difficulty of covering the whole joint space with changing parameters using one set of ANN, and end-effecter integrated camera system makes the image of an object consistent when the end-effecter approaches the object. The object coordinate can be directly used as feedback.

Most vision-based robot positioning techniques rely on analytical formulations of the relationship between the robot pose and the projected image coordinates of several geometric features of the observed scene. Feature matching algorithms, camera calibration, models of the camera geometry and object feature relationships are also necessary for pose determination. These steps are often computationally intensive and error-prone, and the complexity of the resulting formulations often limit the number of controllable degrees of freedom.

This thesis presents controlling mechanism of a parallel robot based on deep neural learning and position based visual servoing that overcomes many of these limitations. ROS/Gazebo simulator is used to model delta 3 parallel robot. From the model training data set is collected and a multi-layer feed forward deep neural network is used to learn the complex implicit relationship between the pose displacements of the delta 3 robot and joint angles. Three networks with three hidden layers but different number of neurons per hidden layer were trained and their performance is evaluated. Based on the simulation result it is shown that a network with higher number of neurons per hidden layer shows better performance.

The trained network may then be used to move the robot from arbitrary initial positions to a desired pose with respect to the observed scene with MSE less than 0.05. Simulation result shows that the system works smoothly, and converges in limited steps. The algorithm simplifies the model of vision based robot manipulator control system, and improves the control accuracy and response time**.**

key words: Artificial Intelligence; Machine Learning; Artificial Neural Networks; Deep Learning; Deep Neural Networks; Feed-forward neural Network; Rectified Linear Units; ROS/Gazebo; Supervised Learning; Visual Servoing

Table of Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

NN	Neural Networks
ANN	Artificial Neural Network
DANN	Deep Artificial Neural Network
FPK	Forward Position Kinematics
IPK	Inverse Position Kinematics
IMC	Internal Model Control
ROS	Robot Operating System
AI	Artificial Intelligence
ANNC	Artificial Neural Network Controller
MLP	Multi-Layer Perceptron
MSE	Mean Square Error
VS	Visual Servoing

Chapter One

1. Introduction

1.1. Background

A long standing goal in the field of robotics has been to endow robots with more sophisticated position control capabilities through the use of vision feedback. In theory, a robot with vision would be able to approach, track and grasp unknown objects in arbitrary locations, assemble parts and avoid unexpected obstacles, even despite deformations and mis-calibrations in its mechanical structure. The ideal system can be imagined as that which allows manipulating arbitrary objects in real environments, without having to model the object, the environment or the camera, which needs no calibration, and which can track moving objects in real time.

As the vision based control field is developing, a bottleneck problem is the confliction between high speed motion response and low bandwidth sensor data from camera became more and more apparent, because the vision data obtained from camera is a huge data flow, and it needs more time to process.To extract 3D object counter model from 2D image needs very complex processing, which adds more processing time to the system. Another factor is the complexity of manipulator kinematics. Fast algorithms have to be developed to relieve this burden. Machine learning is an ideal way to solve this problem. Visual servo is dynamical machine vision used to control the behavior of a robotic structure using visual information to execute a certain task.

In robotics, the class of parallel robots represents a constructive solution suitable for many industrial, medical or domestic applications, like flight simulators and entertainment rides, micro manufacturing, tool machine, pick and place operation, earthquake motion simulator, medical haptic devices, laparoscopic surgery, or even place docking technology [1].

Even if parallel robots have a reduced workspace, their benefits over their serial counters are various: stability and rigidity of contacts during haptic interaction, high accuracy, high stiffness, high payload capability, low moving inertia, good dexterity, compact size, large power to weight ratio [2]. Along with these advantages, there still exist many difficulties in the actual control process. The parallel robot used in this thesis is delta 3 parallel robots which has three degree of freedom.

Neural networks have shown great progress in identification of nonlinear systems. There are certain characteristics in ANN which assist them in identifying complex nonlinear systems. ANN are made up of many nonlinear elements and this gives them an advantage over linear techniques in modeling nonlinear systems. ANN are trained by adaptive learning, the network 'learns' how to do tasks, perform functions based on the data given for training. The knowledge learned during training is stored in the synaptic weights. The standard ANN structures feed forward ANN is used to train the delta 3 robot.

The main task of this project is to design a neural network controller where vision system is used as part of feedback to the system. After training, visual feedback guides the robot to the target from any arbitrary location in the workspace. This uses camera sensor as a feedback and controls the position of the end effecter.

There are three main types of neural control: supervised, direct inverse and unsupervised. Supervised learning uses an existing controller or human feedback in training the neural network. In order to train the neural network to imitate an existing controller a vector of inputs and control targets from the controller must be collected. With supervised control, a neural network could be trained to imitate a robust controller [3] .The robust controller can operate correctly, if the process operates around a certain point. The neuro-controller operates similarly to the robust controller but can also adapt if any disturbance occurs in the system.

In unsupervised learning set-up, no existing controllers can be imitated and the ANN doesn't have a target to compare to its output. The ANN must try different states and determine which state produces a good output.

Direct inverse control does not require an existing controller in training. A neural network is trained to model the inverse of the process [4]. The neural network is cascaded with the process. Theoretically if the inverse model is very accurate, the nonlinearities in the ANN will cancel out the nonlinearities in the process.

The proposed system mainly deals with applying vision sensing technology and machine learning techniques into delta 3 robot eye in hand configuration. Vision system is used as part of feed back to the system. Three DANN will be trained and the results will be presented accordingly. After training visual feedback guides the robot to the target from any arbitrary location in the work place with minimal computational time.

1.2. Problem Description

As the vision based control field is developing, the bottleneck problem which is the confliction between high speed motion response and low bandwidth sensor data from camera became more and more apparent, because the vision data obtained from camera is a huge data flow, and it needs more time to process. Secondly, to extract 3D object counter model from 2D image needs very complex processing, which adds more processing time to the system. Fast algorithms have to be developed to relieve this burden. Another factor is the complexity of manipulator kinematics. Neural networks have shown great progress in identification of nonlinear systems.

If it is possible to teach the robot arm to decide its next movements by using vision as a feedback from eye in hand configuration and resolve the delay from camera sensor data, then it solves the time delay encountered from image processing, reduce computational time, improve the robustness and increase the accuracy and speed of vision based robot manipulator.

1.3. Objective of the Study

1.3.1. General Objective

The general objective of this research is to design and model a vision based robot arm controller by using machine learning techniques.

1.3.2. Specific Objectives

Specific objectives of this research are

- Modeling of delta 3 robot on ROS\Gazebo
- Design machine learning based mechanism to model inter-related visual kinematics relations.
- To design AI controller for direct learning of the image Jacobian to adjust the target position.
- To stimulate Arm movements, dynamics, and kinematic with ROS (Robot operating System)
- To evaluate the performance of the proposed controller by using ROS\Gazebo

1.4. Methodology

In this thesis delta 3 robot with eye in hand configuration is modeled in ROS/Gazebo Environment. In order to find a robust controller for challenges faced by visual servoing, artificial intelligent

controllers are designed specifically neuro controller. From the robot model, training data set is collected. This data set is later used to train three DANN with three hidden layers but different number of neurons per hidden layer. Multi-layer feed forward deep neural network with back propagation learning algorithm is used to train the three networks. Finally the performance of each controllers is evaluated based on simulation result obtained from Gazebo simulator. The high level description of methods and methodologies used in doing this thesis is as shown in Figure 1.1.

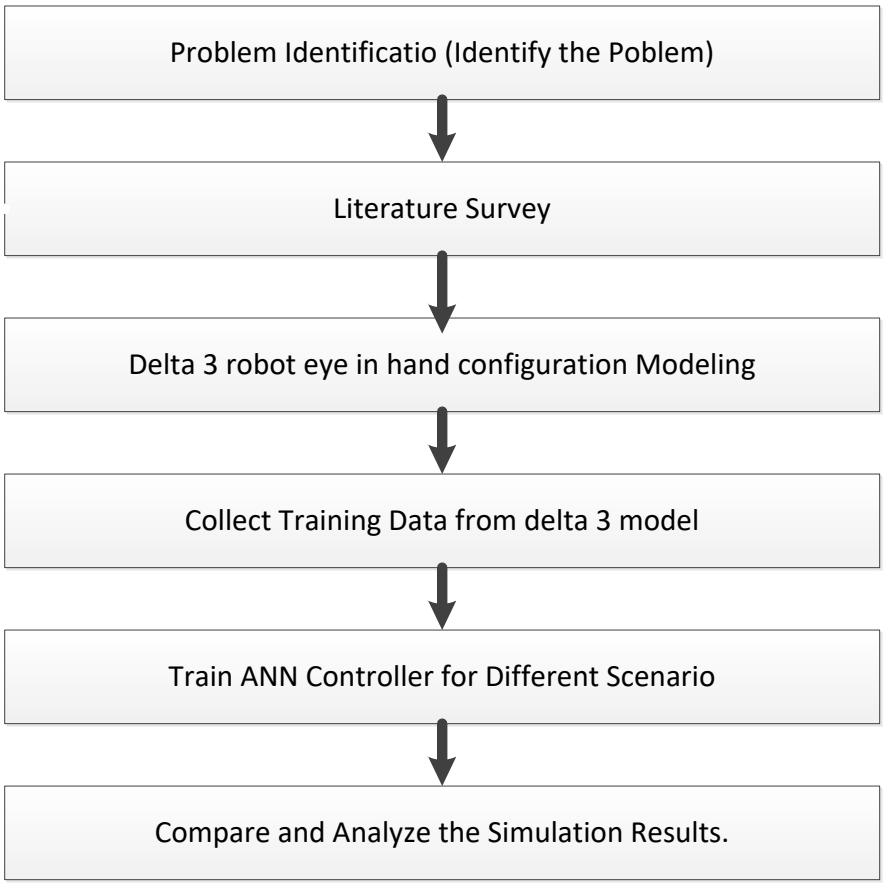

Figure 1-1: High Level Methodology Flow Chart

1.5. Scope and Limitation

The scope of this thesis is limited to modeling the delta 3 robot with eye in hand configuration and design three DANN controllers and comparing the performance of the controllers with the data gathered from the simulation environment, and each other. There is no further real time experimental setup to validate the simulation results.

1.6. Thesis Contribution

The significance of doing this thesis is to address the challenges of robot vision system data delay and image processing problem. Also it shows a new way of modeling Delta 3 robot in a new

Environment with eye in hand model and apply machine learning algorism based controller that can predict the motion of the robot, so that the vision system data delay and image processing problem can be improved.

This helps to improve accuracy and speed of the robot manipulator. still there is a gap in the works done previously to minimize computational time. So the unique aspect of this research is, by using machine learning techniques to develop a fast learning controller in order to minimize computational time so as to improve automation.

1.7. Outline of the Thesis

This thesis is organized in to 5 chapters. Chapter 2 details theoretical background on parallel robot, machine learning techniques and visual servoing. Dynamic system equations of delta 3 robot are expressed and also revises previous work done relates to this work. Chapter 3 covers methods and methodology used. Modeling of delta 3 robot with eye-in-hand configuration is done by gazebo simulation tool, the development of the Feed forward ANN controllers to stabilize the system is also discussed. Chapter 4 presents simulation result analysis and discussion. Finally Chapter 5 provides conclusion and recommendation for future work.

Chapter Two

2. Theoretical Background and Literature Review

2.1. Theoretical Back Ground

2.1.1. Parallel Robot

A generalized parallel robot is a closed-loop kinematic chain mechanism whose moving platform is linked to the base by several independent kinematic chains. A parallel robot consists of a fixed base platform connected to a moving platform by means of a number of limbs. These limbs often consists of an actuated prismatic joint connected to the platforms through passive spherical and\or universal joint. Hence, the links feel only traction or compression, not bending, which increases their position accuracy and allows a lighter construction. Moreover, in principle, parallel robots have high structural stiffness, since the moving platform is supported by several limbs at the same time [5]. Figure 2.1 shows the schematic of a general parallel robot.

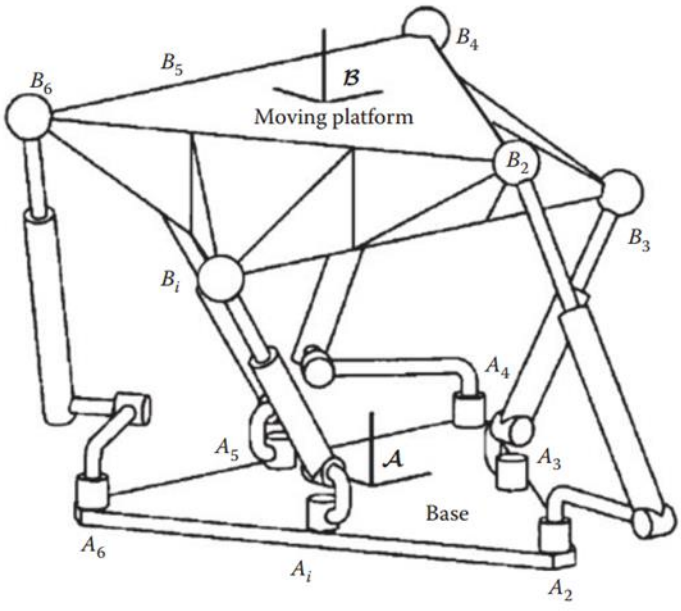

Figure 2-1: schematic of a general parallel robot [6]

All these features result in robots with a wide range of motion capability. Their major drawback is their limited workspace because the limbs can collide and, in addition each limb has a number of passive joints that have their own mechanical limits. Another drawback of parallel robots is that they may completely lose their stiffness at singular positions, and hence the robot gains extra

degrees-of-freedom, which are uncontrollable, and therefore, it becomes shaky or mobile at these configurations.

2.1.1.1. The Stewart-Gough platform

As historically reviewed by Bonev [7], the most celebrated parallel robot, which is paradoxically referred to in the literature as the Stewart platform was first invented by V.E. Gough in 1947. The idea behind this invention, which is called a universal rig, was borrowed from the design of an octahedral hexapod that was used to determine the properties of tires under combined loads. As shown in figure 2.1, in a Gough platform there are six independently actuated limbs, where the lengths of the legs are controlled to move the platform to a desired position and orientation.

As was reported in Gough's paper in 1956 [8], the octahedral hexapod was not invented from scratch, and at that time, systems with six struts (hexapods) were already in use. These hexapods had usually three vertical struts and three horizontal ones, and have been very popular then and are still used in different applications. The new idea of the Gough platform was the arrangement of the six struts. Since Gough needed relatively large ranges of motion, he naturally selected symmetrical arrangement forming an octahedron. The machine was built in the early 1950s and was fully operational in 1954. Although Gough was the first to invent and build the popular octahedral hexapod, Klaus Cappel later designed independently the very same hexapod, patented it [9], and licensed it to the first simulator companies. A picture of his first flight simulator is shown in Fig 2.2.

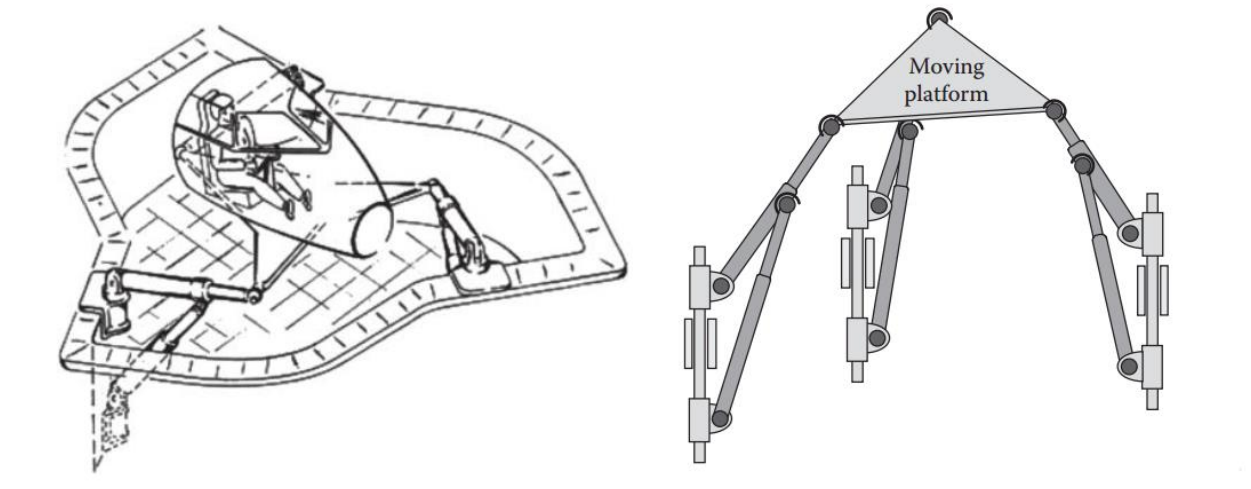

Figure 2-2: A schematics of the Stewart platform and its use as flight simulator [9]

The name of Stewart was attached to this architecture because Gough's earlier work and a photograph of his platform were mentioned in reviewers' remarks on a paper published by Stewart in 1965. In the paper, Stewart presents another hybrid design, with three legs having two actuators each. Stewart proposed this six degrees-of freedom motion platform for use as a flight simulator. The proposed parallel mechanism, however, is different from the octahedral invented by Gough that is paradoxically often referred to as the Stewart platform. There is no doubt that Stewart's remarkable paper had a great impact on the subsequent development in the field of parallel kinematics. Various suggestion for the use of a hexapod were made, many of which were accurate predictions of the future.

2.1.1.2. The delta Robot

A Delta robot is a type of parallel robot which consists of three arms connected to universal joints at the base (Figure 2.3). The key design feature is the use of parallelograms in the arms, which maintains the orientation of the end effecter. Delta robots have popular usage in the picking and packaging factories because they can be quite fast, some executing up to 200 picks per minute [5]. A delta robot was invented in the early 1980s by Reymond Clavel at the EPEL, Switzerland. The purpose of this new type of robot was to manipulate light and small objects at a very high speed. In 1987, Demaurex Company purchased a license for the Delta robot and started the production of Delta robots for packaging industry. In 1991, Reymond clavel presented his doctoral thesis and received the Golden Robot Award in1999 for his work and development of the Delta robot. In 1999, ABB Flexible Automation started selling its Delta robot, the Flex Picker [6].

A Delta robot is parallel robot, which can also be seen as a spatial generalization of a planar four-bar mechanism. It has four degrees-of-freedom: three translational and one rotational. The key concept of a Delta robot is the use of parallelograms. A parallelogram allows an output link to remain at a fixed orientation with respect to an input link. The use of three such parallelograms restrains completely the orientation of the mobile platform, which remains only with three purely translational degrees-of-freedom. The robot base is mounted above the work space. All the actuators are located on this base. From the base, three middle jointed arms are extended. The arms are usually made of lightweight composite material. The ends of the three arms are connected to a small triangular platform. Actuation of the input links will move the triangular platform in the x, y or z directions. Actuation can be done by linear or rotational actuators. From the base, a fourth

leg is used to transmit rotary motion from the base to an end effecter mounted on the mobile platform.

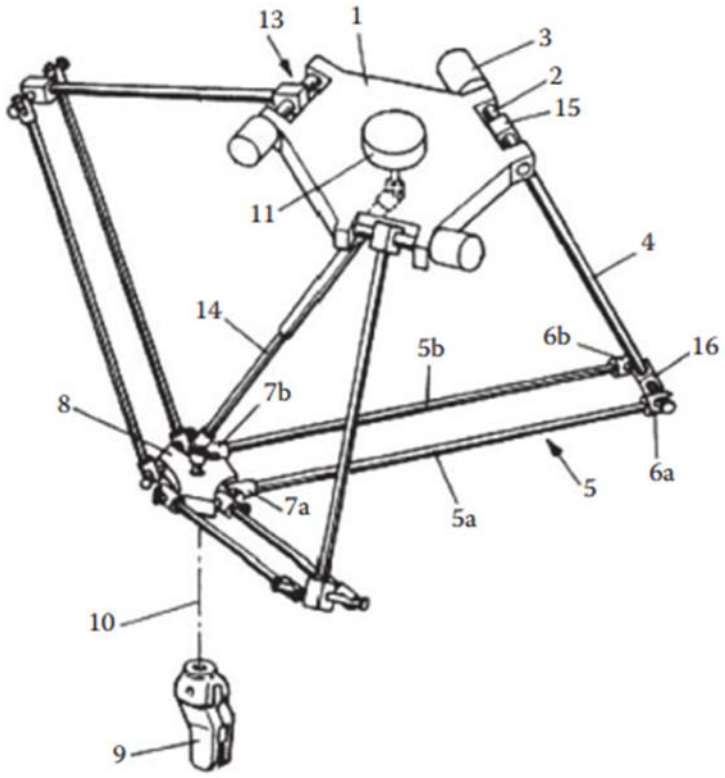

Figure 2-3: Schematics of delta robot [5]

Since the actuators are all located on the base, and the arms are made of a composite material, the moving parts of the Delta robot have a small inertia. This allows for very high accelerations; accelerations can be up to 30g and speeds of 10 m/s may be reached. This makes the Delta robot a perfect candidate for pick-and-place operations of light objects (from 10g to 1kg). Delta robots available on the market operate typically in a cylindrical workspace that is 1 m in diameter and 0.2 m high. The industries that take advantage of the high speed of the Delta robot are the packaging, medical, and pharmaceutical industries. Adept has built its delta-type parallel robot called Quattro for packaging industries. This manipulator has one degree of redundancy in actuation and consists of four parallelogram limbs. The structure of the Delta robot has also been used to create commercial haptic devices such as the Novint Falcon.

In this thesis delta 3 robot which has three DOF is used to demonstrate ANN based position control.

2.1.2. Delta Robot Kinematics Analysis

From the kinematic diagram below (figure 2.4), the following three vector-loop closure equations are written for the Delta Robot: [10]

$$\{^{B}B_i\}+\{^{B}L_j\}+\{^{B}I_j\}=\{^{B}P_p\}+[_{p}^{B}R]\{^{p}P_j\}=\{^{B}P_p\}+\{^{B}P_j\} \quad i=1,2,3 \tag{2.1}$$

Where $[p^{R}B]=[I_3]$,since no rotations are allowed by the Delta Robot.

The three applicable constraints state that the lower leg lengths must have the correct, constant length *l* (the virtual length through the center of each parallelogram):

$$l_i = \{^{B}I_j\} = \{^{B}P_j\} + \{^{p}P_j\} - \{^{B}B_i\} - \{^{B}L_i\} \qquad i=1,2, \tag{2.2}$$

It will be more convenient to square both sides of the constraint equations above to avoid the square-root in the Euclidean norms:

$$l_i^2 = \{^{B}I_j\} = l^2_{ix} + l^2_{iy} + l^2_{iz} \tag{2.3}$$

Again, the Cartesian variables are $^{B}P_P = \{x\ y\ z\}^{T}$. The constant vector values for points P_i and B_i *were* given previously. The vectors $\{^{B}L_i\}$ are dependent on the joint variables $\boldsymbol{\Theta} = \{\theta_1\ \theta_2\ \theta_3\}$:

$$^{B}L_1 = \begin{Bmatrix} 0 \\ -Lcos\theta 1 \\ -Lsin\theta 1 \end{Bmatrix} \qquad ^{B}L_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2}Lcos\theta 2 \\ \frac{1}{2}Lcos\theta 2 \\ -Lsin\theta 2 \end{Bmatrix} \qquad ^{B}L_3 = \begin{Bmatrix} \frac{-\sqrt{3}}{2}Lcos\theta 3 \\ \frac{1}{2}Lcos\theta 3 \\ -Lsin\theta 3 \end{Bmatrix} \tag{2.4}$$

Substituting all above values into the vector-loop closure equations yields:

$$\{^{B}l_1\} = \begin{Bmatrix} x \\ y + Lcos\theta 1 + a \\ z + Lsin\theta 1 \end{Bmatrix} \quad \{^{B}l_2\} = \begin{Bmatrix} x - \frac{\sqrt{3}}{2}Lcos\theta 2 + b \\ y - \frac{1}{2}Lcos\theta 2 + c \\ z + Lcos\theta 2 \end{Bmatrix} \quad ^{B}L_3 = \begin{Bmatrix} x + \frac{\sqrt{3}}{2}Lcos\theta 3 - b \\ y - \frac{1}{2}Lcos\theta 3 + c \\ z + Lcos\theta 3 \end{Bmatrix} \tag{2.5}$$

Where $a=w_b - u_p$

$b=s_{p/2} - \frac{-\sqrt{3}}{2}w_B$

$c=w_p - 1/2\ w_B$

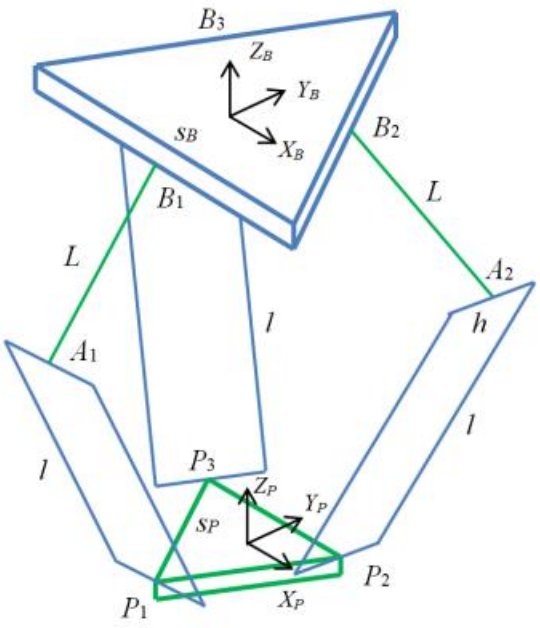

Figure 2-4 Delta Robot Kinematic Diagram [11]

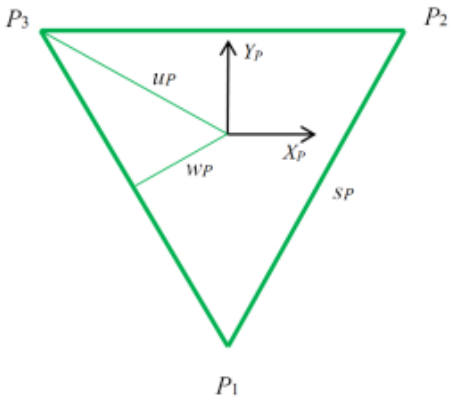

Figure 2-5 Delta Robot Moving Platform Details [11]

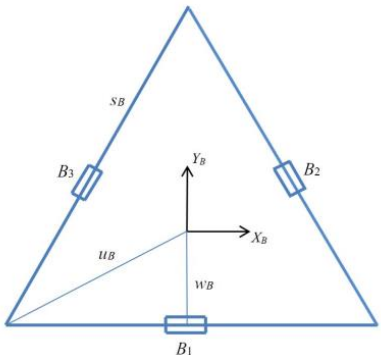

Figure 2-6 Delta Robot Fixed Base detail [11]

And the three constraint equations yield the kinematics equations for the Delta Robot:

$$2L(y+a)\cos\theta1 + 2zL\sin\theta1 + x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2 = 0$$

$$-L(\sqrt{3}\,(x+b) + y + c)\cos\theta2 + 2zL\sin\theta2 + x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2xb + 2yc - l^2 = 0 \quad (2.6)$$

$$L(\sqrt{3}\,(x-b) - y - c)\cos\theta3 + 2zL\sin\theta3 + x^2 + y^2 + z^2 + b^2 + c^2 + L^2 - 2xb + 2yc - l^2 = 0$$

The three absolute vector knee point are found using ${}^{B}\mathbf{A}_i = {}^{B}\mathbf{B}_i = {}^{B}\mathbf{L}_i$, $i = 1,2,3$

$${}^{B}\mathbf{A}_1 = \begin{Bmatrix} 0 \\ -w_B\ -L\cos\theta1 \\ -L\sin\theta1 \end{Bmatrix} \quad {}^{B}\mathbf{A}_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2}(w_B + L\cos\theta2) \\ \frac{1}{2}(w_B + L\cos\theta2) \\ -L\sin\theta2 \end{Bmatrix} \quad {}^{B}\mathbf{A}_3 = \begin{Bmatrix} \frac{-\sqrt{3}}{2}(w_B + L\cos\theta3) \\ \frac{1}{2}(w_B + L\cos\theta3) \\ -L\sin\theta3 \end{Bmatrix} \quad (2.7)$$

2.1.2.1. Inverse Position Kinematics (IPK) Solution

The 3-dof Delta Robot inverse position kinematics (IPK) problem is stated: Given the Cartesian position of the moving platform control point (the origin of $\{P\}$), $B\mathbf{P}P = \{x\ y\ z\}^T$, calculate the three required actuated revolute joint angles $\Theta = \{\theta_1\ \theta_2\ \theta_3\}^T$. The IPK solution for parallel robots is often straightforward; the IPK solution for the Delta Robot is not trivial, but can be found analytically. Referring to the Delta Robot kinematic diagram above, the IPK problem can be solved independently for each of the three RUU legs. Geometrically, each leg IPK solution is the intersection between a known circle (radius *L*, centered on the base triangle R joint point ${}^{B}B_i$) and a known sphere (radius l, centered on the moving plat form vertex ${}^{P}P_i$).

This solution may be done geometrically/trigonometrically. However, we will now accomplish this IPK solution analytically, using the three constraint equations applied to the vector loop-closure equations (derived previously). The three independent scalar IPK equations are of the form: [11]

$$E_i\cos\theta1 + F_i\sin\theta1 + G_i = 0 \qquad i = 1, 2, 3 \qquad (2.8)$$

Where

$$E_1 = 2L(y + a)$$

$$F_1 = 2zL$$

$$G_1 = x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2$$

$E_2 = -L(\sqrt{3}\,(x+b) + y + c)$ $\qquad$ $E_3 = L(\sqrt{3}\,(x+b) + y + c)$

$F_2=2zL$ $F_3=2zL$

$G_2= x^2 + y^2 +z^2 +b^2 + c^2+L^2+2(xb + yc)– l^2$ $G_3= G_2= x^2 + y^2 +z^2 +b^2 + c^2+L^2+2(-xb+yc)– l^2$ (2.10)

The equation $E_i\, cos\theta 1 + F_i\, sin\theta 1 + G_i=0$ appears a lot in robot and mechanism kinematics and is readily solved using the Tangent Half-Angle Substitution.

If we define $t_i=\tan\frac{\theta 1}{2}$ then $\cos\theta_1=\frac{1-t_i^2}{1+t_i^2}$ and $\sin\theta_1=\frac{2t_i}{1+t_i^2}$

Substitute the Tangent Half-Angle Substitution into the *EFG* equation:

$E_i\left(\frac{1-t_i^2}{1+t_i^2}\right)+F_i\left(\frac{2t_i}{1+t_i^2}\right)+G_i=0$ (2.11)

$E_i(1-t_i^2)+F_i(2t_i)+G_i(1+t_i^2)=0$ (2.12)

$(G_i- E_i)\, t_i^2+(2F_i)t_i+(G_i+E_i)=0$ (2.13)

quadratic formula $t_{i1,2}=x=\frac{-F_i\pm\sqrt{E_i^2+F_i^2-G_i^2}}{G_i-E_i}$ (2.14)

Solve for θ_i by inverting the original Tangent Half-Angle Substitution definition:

$\theta_i=2tan^{-1}(t_i)$ (2.15)

Two θi solutions result from the ± in the quadratic formula. Both are correct since there are two valid solutions: knee left and knee right. This yields two IPK branch solutions for each leg of the Delta Robot, for a total of 8 possible valid solutions. Generally the one solution with all knees kinked out instead of in will be chosen.

2.1.2.2. Forward Position Kinematics (FPK) Solutions

The 3-dof Delta Robot forward position kinematics (FPK) problem is stated: [11] given the three actuated joint angles $\mathbf{\Theta}=\{\theta 1 \;\; \theta 2 \;\; \theta 3\}^T$, calculate the resulting Cartesian position of the moving platform control point (the origin of $\{P\}$), ${}^B\mathbf{P}_P=\{x\ y\ z\}^T$. The FPK solution for parallel robots is generally very difficult. It requires the solution of multiple coupled nonlinear algebraic equations, from the three constraint equations applied to the vector loop-closure equations (derived previously). Multiple valid solutions generally result.

Thanks to the translation-only motion of the 3-dof Delta Robot, there is a straightforward analytical solution for which the correct solution set is easily chosen.

Since $\Theta=\{\theta1 \quad \theta2 \quad \theta3\}^T$ are given, we calculate the three absolute vector knee points using ${}^{B}A_i = {}^{B}B_i + {}^{B}L_i$, i =1,2,3.Referring to the Delta Robot FPK diagram below, since we know that the moving platform orientation is constant, always horizontal with $[{}^{B}_{p}R] = [I_3]$ we define three virtual sphere centers:

${}^{B}A_{1v} = {}^{B}A_i + {}^{P}P_i$, i =1,2,3: (2.16)

$$ {}^{B}A_{1v} = \begin{Bmatrix} 0 \\ -w_B \\ -Lsin\theta1 \end{Bmatrix} \begin{matrix} \\ -Lcos\theta1 + u_p \\ \\ \end{matrix} $$

$$ {}^{B}A_{2v} = \begin{Bmatrix} \frac{\sqrt{3}}{2}(w_B + Lcos\theta2) - s^p/2 \\ \frac{1}{2}(w_B + Lcos\theta2) - w_p \\ -Lsin\theta2 \end{Bmatrix} \qquad (2.17) $$

$$ {}^{B}A_{3v} = \begin{Bmatrix} \frac{-\sqrt{3}}{2}(w_B + Lcos\theta3 + s^p/2) \\ \frac{1}{2}(w_B + Lcos\theta3) - w_p \\ -Lsin\theta3 \end{Bmatrix} $$

And then the Delta Robot FPK solution is the intersection point of three known spheres. Let a sphere be referred as a vector center point {**c**} and scalar radius *r*, ({**c**}, *r*). Therefore, the FPK unknown point $\{{}^{B}\mathbf{P}_P\}$ is the intersection of the three known spheres:

$(\{{}^{B}\mathbf{A}_{1v}, l\})$ $(\{{}^{B}\mathbf{A}_{2v}\}, l)$ $(\{{}^{B}\mathbf{A}_{3v}\}, l)$

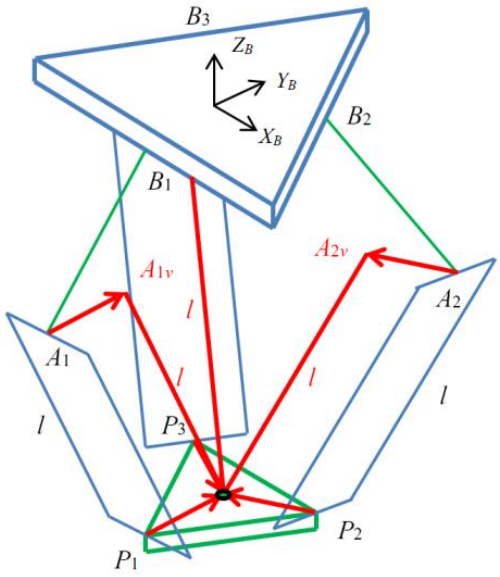

Figure 2-7 Delta Robot FPK Diagram [11]

2.1.3. Machine Learning

Machine learning is the application of artificial intelligence (AI) that provide systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observation or data, such as examples, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly [12].

In logic and statistical inference, transduction is reasoning from observed, specific (training) cases to general rules, which are then applied to the test cases. Machine learning falls into two broad classes: inductive learning or transductive learning [13].

Inductive learning pursues the standard goal in machine learning, which is to accurately classify the entire input space. In contrast, transductive learning focuses on a predefined target set of unlabeled data, the goal being to label the specific target set.

Multi task learning improves the generalization performance of learns by leveraging the domain-specific information contained in the related tasks. Multiple related tasks are learned simultaneously using a shared representation. In fact, the training signals for extra tasks serve as an inductive bias [14].

In order to learn accurate models for rare cases, it is desirable to use data and knowledge from similar cases; this is known as transfer learning. Transfer learning is a general method for speeding up learning. It exploits the insight that generalization may occur not only within tasks, but also across tasks. The core idea of transfer is that experience gained in learning to perform one source task can help improve learning performance in a related, but different, target task. Transfer learning is related in sprit to case-based and analogical learning. A theoretical analysis based on an empirical Bayes perspective exhibits that the number of labeled examples required for learning with transfer is often significantly smaller than that required for learning each target independently [14].

2.1.4. Visual servoing

Vision is a useful robotic sensor since it mimics the human sense of vision and allows for non-Contact measurement of the environment. Since the early work of Shirai and Inoue [15] who describe how a visual feedback loop can be used to correct the position of a robot to increase task accuracy , considerable effort has been devoted to the visual control of robot manipulators. Robot controllers with fully integrated vision systems are now available from a number of vendors. Typically visual sensing and manipulation are combined in an open-loop fashion, `looking' then `moving'. The accuracy of the resulting operation depends directly on the accuracy of the visual sensor and the robot end-effector.

An alternative to increasing the accuracy of these subsystems is to use a visual-feedback control loop that will increase the overall accuracy of the system. A principal concern in most applications. Taken to the extreme, machine vision can provide closed-loop position control for a robot end-effector this is referred to as visual servoing. This term appears to have been first introduced by Hill and Park in 1979 to distinguish their approach from earlier `blocks world' experiments where the system alternated between picture taking and moving. Prior to the introduction of this term, the less specific term visual feedback was generally used.

Visual servoing is the fusion of results from many elemental areas including high-speed image Processing, kinematics, dynamics, control theory, and real-time computing. It has much in common with research into active vision and structure from motion, but is quite different from the often described use of vision in hierarchical task-level robot control systems. Many of the control and vision problems are similar to those encountered by active vision researchers who are building `robotic heads'. However the task in visual servoing is to control a robot to manipulate its environment using vision as opposed to just observing the environment.

Visual perception is important for humans and robots alike, it provides rich and detailed information about the environment the agent is moving in. The goal of visual servoing techniques is to control a dynamic system, such as a robot, by using the information provided by one or multiple cameras [15]. Classical approaches to visual servoing rely on the extraction, tracking and matching of a set of visual features. These features, generally points, lines, or moments, are used as inputs to a control law that positions (or navigates) the robot in a desired pose. Many control

strategies have been proposed over the years, in particular neural networks have been considered when designing control schemes early on.

The tracking and matching of such features, especially given the rich and detailed information stemming from cameras, is a difficult task. While there has been progress in extracting the relevant features, a technique called direct visual servoing was introduced recently for exploiting the full image, requiring no feature extraction [16].The main drawback of this direct approach is its small convergence domain compared to classical techniques. This is due to high nonlinearities between the image information and the 3D motion. To remedy this issue this thesis herein propose the use of a trained neural network to perform the extraction of features and estimation of the current Cartesian coordinate pose relative to the desired.

Visual Servoing (VS) techniques have been applied to a variety of robotic tasks, including reaching, docking and navigation. While most of these approaches require a feature extractor or model tracker, direct VS was introduced to make use of the information available in full images [15]. The principle is to directly compare the current image with the desired image as a whole, avoiding the classical but error-prone feature detection, tracking and matching steps. Various control laws have been proposed in order to improve the robustness of direct VS approaches by varying descriptors or the cost functions, such as mutual information, histogram distances or mixture of Gaussians.

The main drawback of these dense approaches is their small convergence domain, due to high non-linearity between the image information and the 3D motion. To remedy this issue, combining direct VS with other approaches has been proposed recently, such as the use of particle filtering in the control scheme or to consider photometric moments to retain geometric information. Over the last years neural networks, have progressed the state-of-the art in a number of computer vision tasks: image classification for object recognition, inferring a depth map from a single RGB image, computing displacement through homograph estimation, and performing camera relocalization. Learning has also started to become more prominent in robotics. For example, NNs have also been trained for predicting grasp locations. Recent progress in deep learning reaching tasks has achieved promising results: learning complex positioning tasks facilitated by vision coupled with supervised learning and purely from vision [17] .

2.2. Literature Review

ANN has been introduced into vision based robot control at the end of the last century. Zhao and Cheah [18] have developed an ANN for vision based control of multi-fingered robot hands. The robot uses a multi-fingered hand to grasp an object, a vision system is used to sense the position and orientation of the object to provide feedback signal. Because the kinematics of the multiple fingers is difficult to obtain, an ANN is employed to obtain the control model. They developed a vision-based ANN controller for robots with uncertain kinematics, dynamics and constraint surface. The controller achieves stability with uncertain constraints without knowing the exact model.

Kuhn et al. [17] have proposed an ANN system to teach a robot 3 DOF manipulator to reach an object. The Jacobean of the manipulator is identified by the ANN. Rathbone and Sharkey [ ] have developed another ANN system combined with generic algorithm (GA). They used GA to selectively set the initial status of the ANN, and then the ANN can learn to improve its performance.

Gonzalez-Olvera et al. [19] have developed Black-Box model of a 2-DOF manipulator using recurrent neuro-fuzzy networks. This thesis treats the manipulator as a black box with actuating voltages as input and joint angles in image as output. A Neuro-fuzzy network is implemented to represent the dynamics of the manipulator, and it is trained by simulating and experimental data. This work simplified the identification of the manipulator model.

Widrow-Hoff uses NN, with adaptive learning algorithm derived using Lyapunov stability theory, was used [20] in to obtain kinematics inversion of redundant manipulator. As this inverse kinematics has infinite number of solutions, additional fuzzy neural network was used as hint generator to limit searching space and to improve quality of the solution. Some physical constraints, like joint velocity limits and joint angle limits, are incorporated into this approach.

Pei and mittal in [21] proposed position specific ANN to control a robot arm which has end-effector integrated camera system. Position specific ANN was chosen in order to avoid the difficulty of covering the whole joint space with changing parameter using one set of ANN and end effector integrated camera makes the image of an object consistence when the end effector approaches the object.

Hessa Al-Junaid [22] developed an ANN-based mechanism to model inter-related visual kinematics relations, which are an important fragment of visual servo closed loop system. The goal was is to visually servo a 6-DOF robot arm, from an initial location toward a desired posture using only image data from a scene provided during the arm maneuver. Arm movements, dynamics, and kinematics were simulated with Matlab RoboticsTool Box. The arm is equipped with a CCD camera, where the visual CCD part is modeled and simulated with Matlab Epipolar GeometryToolbox. The main issue that hinders visual servo system is related to the variant feature Jacobian matrix. The methodology used is based on the concept of integrating a multilayer ANN with an Image Based Visual Servoing system. ANN was used to learn, approximate, and map the highly complex relations that relate a scene movements to an arm movement through a visual servo.

In [23] describes configuration space control of a Delta robot with a neural network based kinematics. Since Mathematical model of the kinematics for parallel Delta robot used for manipulation purposes in micro factory was validated, and experiments showed that this model is not describing "real" kinematics properly they proposed a new solution for kinematics mapping. The Solution was found in neural network utilization, and it was used to model robot's inverse kinematics. Neural network is then used as a part of the control system. It showed significantly better mapping between task space coordinates and configuration (joint) space coordinates than the mathematical model, for the workspace of interest. Consequently positioning accuracy was improved.

What makes this thesis unique from the aforementioned literatures is it incorporates visual servoing with AI control mechanism for a parallel robot specifically delta 3 robot. In this thesis multi-layer feed forward DNN is used to model inter-related visual kinematics relations of the delta robot. Delta robot model was modeled by using ROS (Gazebo) simulator to provide necessary set of data for network training and also to uses as delta 3 environment in order to collect simulation result. Back propagation algorithm is then used to design the controller. Its performance is analyzed by using mean square error. Also by varying training number of hidden layer neurons, Comparison and analysis will be made on the performance of the controller.

Chapter Three

3. System Design

3.1. Artificial Neural Networks

Artificial neural networks are circuits, computer algorithms, or mathematical representations loosely inspired by the massively connected set of neurons that form biological neural networks. Artificial neural networks are an alternative computing technology that have proven useful in a variety of pattern recognition, signal processing, estimation, and control problems.

3.2.1. The feed forward Multilayer Perceptron

The feed forward multilayer perceptron is the most popular neural network in control system applications that this thesis is used. The second most popular one is the radial basis function neural network.

The multilayer perceptron is composed of an interconnected set of neurons, each of which has the form shown in Figure 3.1.

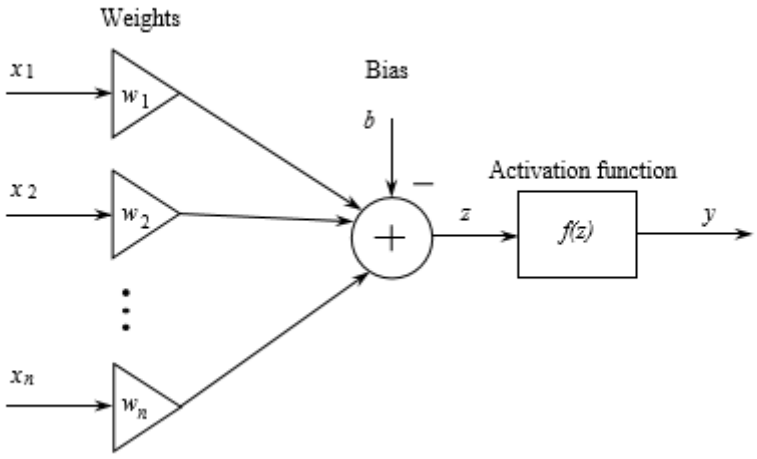

Figure 3-1: Single neuron model

Here,

$$z = \sum_{i=1}^{n} w_i X_i - b \qquad (3.1)$$

And the w_i are the interconnection "weights" and b is the "bias" for the neuron (these parameters model the interconnections between the cell bodies in the neurons of a biological neural network). The signal z represents a signal in the biological neuron and the processing that the neuron performs on this signal is represented with an "activation function" f where

$$y = f(z) = f(\sum_{i=1}^{n} w_i X_i - b) \qquad (3.2)$$

The neuron model represents the biological neuron that "fires" (turns on) when its inputs are significantly excited (i.e., z is big enough). "Firing" is defined by an "activation function" f where two (of many) possibilities for its definition are:

- Threshold function: $f(x) = \begin{cases} 1, & if \ z \geq 0 \\ 0, & if \ z < 0 \end{cases}$ (3.3)
- Sigmoid (logistic) function: $f(z) = \frac{1}{1+e^{-z}}$ (3.4)
- Rectified Linear Unit (ReLU) function: $f(x) = \max(0, x) = \begin{cases} x_i, & if \ x_i \geq 0 \\ 0, & if \ x_i < 0 \end{cases}$ (3.5)
- Linear function: f(z)=z (3.6)

With one of the above activation functions, represents the computations made by one neuron. Next, we interconnect them. Let circles represent the neurons (weights, bias, and activation function), and lines represent the connections between the inputs and neurons and the neurons in one layer and the next layer.

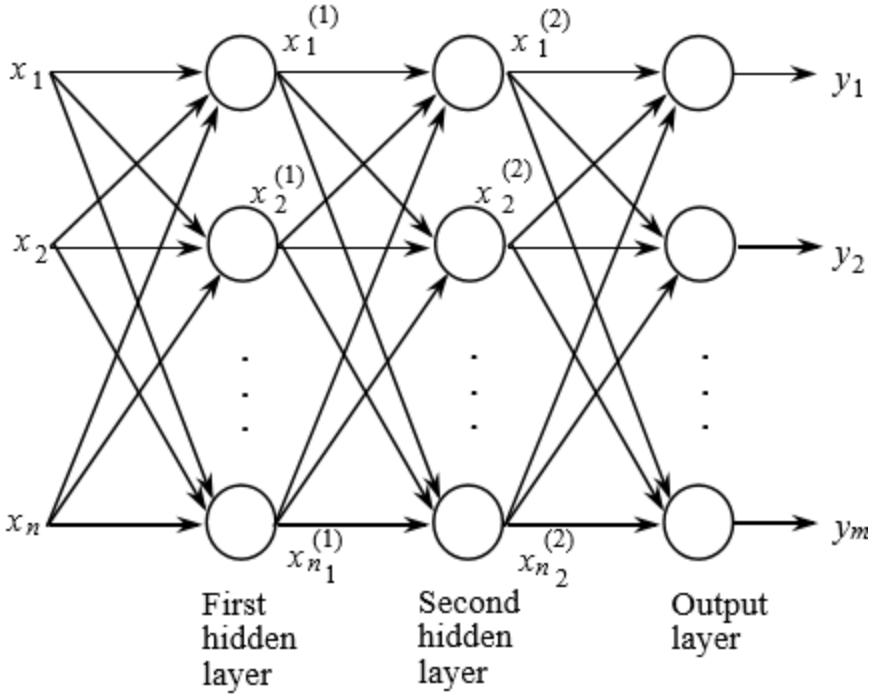

Figure 3-2: Multilayer perceptron model.

Figure 3.2 is a three "layer" perceptron since there are three stages of neural processing between the inputs and outputs. Here, we have

• Inputs: x_i, i =1, 2... n

• Outputs: y_i , j =1, 2,...,m

• Number of neurons in the first "hidden layer," n_1, in the second hidden layer n_2, and in the output layer, m

• In an N layer perceptron there are n_i neurons in the i^{th} hidden layer, i =1, 2... N−1. We have

$$x_j^1 = f_j^1(\sum_{i=1}^{n} w_{ij}^1 X_i - b_j^1) \qquad (3.7)$$

With j =1,2,...,n1. We have

$$x_j^2 = f_j^2\left(\sum_{i=1}^{n} w_{ij}^2 X_i^1 - b_j^2\right) \qquad (3.8)$$

with j =1 ,2,...,n2. We have

$$y_j = f_j\,(\sum_{i=1}^{n2} w_{ij}\, X_i^2 - b_j\ ) \qquad (3.9)$$

with j =1 ,2,...,m. Here, we have

• w^1_{ij} (w^2_{ij}) are the weights of the first (second) hidden layer

• w_{ij} are the weights of the output layer

• b^1_j are the biases of the first hidden layer.

• b^2_j are the biases of the second hidden layer

• b_j are the biases of the output layer

• f_j (for the output layer), f^2_j (for the second hidden layer), and f^1_j (for the first hidden layer) are the activation functions (all can be different).

3.2.2. Deep Neural Networks

Neural networks which consist of more than three layers of neurons (including the input and output layer) are called as deep neural networks. And Training them is called as Deep learning. Deep learning represents the very cutting edge of artificial intelligence. Instead of teaching computers to process and learn from data which is how machine learning works, with deep learning, the computer trains itself to process and learn from data. A deep learning system is self-teaching, learning as it goes by filtering information through multiple hidden layers, in a similar way to humans.

To represent more complex features and to "learn" increasingly complex models for prediction and classification of information that depends on thousands or even millions of features, we need ANNs which are a little more complex. This is accomplished by simply increasing the number of hidden

layers and/or the number of neurons per hidden layer. More layers and more neurons can represent increasingly complex models but they also come at the cost of increasing time and power-consuming computations. Since this thesis used camera sensor as a feedback and it should cover all the possible points in the work space, it needs thousands of data to train the network so it uses deep neural networks with three hidden layers and train it by using deep learning.

3.2.3. Types of learning

Neural networks have three main modes of operation: supervised, reinforced and unsupervised learning. In supervised learning the output from the neural network is compared with a set of targets, the error signal is used to update the weights in the neural network. Reinforced learning is similar to supervised learning however there are no targets given, the algorithm is given a grade of the ANN performance. Unsupervised learning updates the weights based on the input data only. The ANN learns to cluster different input patterns into different classes.

In this thesis supervised learning is used. Supervised learning adjusts network parameters by a direct comparison between the actual network output and the desired output. Supervised learning is a closed-loop feedback system, where the error is the feedback signal. The error measure, which shows the deference between the network output and the output from the training samples, is used to guide the learning process. The error measure is usually defined by the mean squared error (MSE).

$$E = \frac{1}{N} \sum_{P=1}^{N} \left\| y_p - \hat{y_p} \right\| \qquad (3.10)$$

Where N is the number of pattern pairs in the sample set, y_p is the output part of the p^{th} pattern pair, and $y_p\hat{}$ is the network output corresponding to the pattern pair p.

The error E is calculated repeatedly after each epoch. The learning process is terminated when E is sufficiently small or a failure criterion is met.

To decrease E toward zero, a gradient-descent procedure is usually applied. The gradient-descent method always converges to a local minimum in a neighborhood of the initial solution of network parameters. The LMs and BP algorithms are two most popular gradient-descent based algorithms. Second-order methods are based on the computation of the Hessian matrix.

Multiple-instance learning is a variation of supervised learning. In multiple instance learning, the examples are bags of instances, and the bag label is a function of the labels of its instances. Typically, this function is Boolean OR.

Deductive reasoning starts from a cause to deduce the consequence or effects. Inductive reasoning allows us to deduce possible causes from the consequence. The inductive learning is a special class of the supervised learning techniques, where given a setoff $\{x_i, f(x_i)\}$ pairs, we determine a hypothesis $h(x_i)$ such that $h(x_i)=(x_i)$. In inductive learning, given many positive and negative instances of a problem the learner has to form a concept that supports most of the positive but no negative instances. This requires a number of training instances to form a concept in inductive learning. Unlike this, analogical learning can be accomplished from a single example.

3.2.4. Back Propagation Algorithm

The back propagation algorithm [24] is used in layered feed-forward ANNs. This means that the artificial neurons are organized in layers, and send their signals “forward”, and then the errors are propagated backwards. The network receives inputs by neurons in the input layer, and the output of the network is given by the neurons on an output layer. There may be one or more intermediate hidden layers.

The back propagation algorithm uses supervised learning, which means that we provide the algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. The idea of the back propagation algorithm is to reduce this error, until the ANN learns the training data. The training begins with random weights, and the goal is to adjust them so that the error will be minimal.

For practical reasons, ANNs implementing the back propagation algorithm do not have too many layers, since the time for training the networks grows exponentially. Also, there are refinements to the back propagation algorithm which allow a faster learning.

3.2.5. Advantages of Artificial Neural Networks

The main advantage of neural networks is that it is possible to train a neural network to perform a particular function by adjusting the values of connections (weights) between elements. For example, if we wanted to train a neuron model to approximate a specific function, the weights that

multiply each input signal will be updated until the output from the neuron is similar to the function.

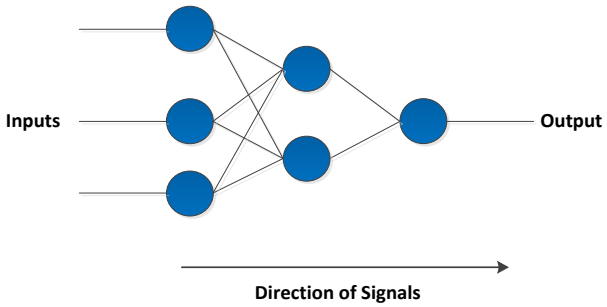

Figure 3-3: Diagram shows the parallelism of neural network

Neural networks are composed of elements operating in parallel. Parallel processing allows increased speed of calculation compared to slower sequential processing.

Artificial neural networks (ANN) have memory. The memory in neural networks corresponds to the weights in the neurons. Neural networks can be trained offline and then transferred into a process where adaptive learning takes place. In this thesis, a neural network controller is designed to control a delta 3 parallel robot offline in Gazebo environment. After training, the network weights are set. The ANN is placed in a feedback loop with the actual process. The network will adapt the weights to improve performance as it controls the delta 3 robot position.

The main disadvantage of ANN is they operate as black boxes. The rules of operation in neural networks are completely unknown. It is not possible to convert the neural structure into known model structures such as ARMAX, etc. Another disadvantage is the amount of time taken to train networks. It can take considerable time to train an ANN for certain functions.

3.2. Position Based Visual Servoing

In robotics, neural networks are used in many tasks such as to solve the inverse kinematic problem of robots, and to map sensory information for robot control. Neural networks are used for direct learning of the image Jacobian, also they are capable of avoiding the costly matching of image features in current and reference images.

The Emergence of ROS (Robot Operating System) goes along the way of improving the current state of the art software development in robotics by aiming at reducing integration costs through

standardization [25].ROS is world-class middleware which allows robot programmers and industrials to quickly and easily make good solutions reusable [26].

Visual servoing consists of using feedback provided by one or several vision sensors to influence the motion of a dynamic system [27], [28], [29].Generally, the task in visual servoing consists of controlling the pose (position and orientation) of a rigid body relative to an object of interest as shown by Figure 3.4. This figure shows the positioning in the world space of the end-effector of delta 3 robot, using the vector of image coordinates of four points in the image space to generate control signals in the robot motor space.

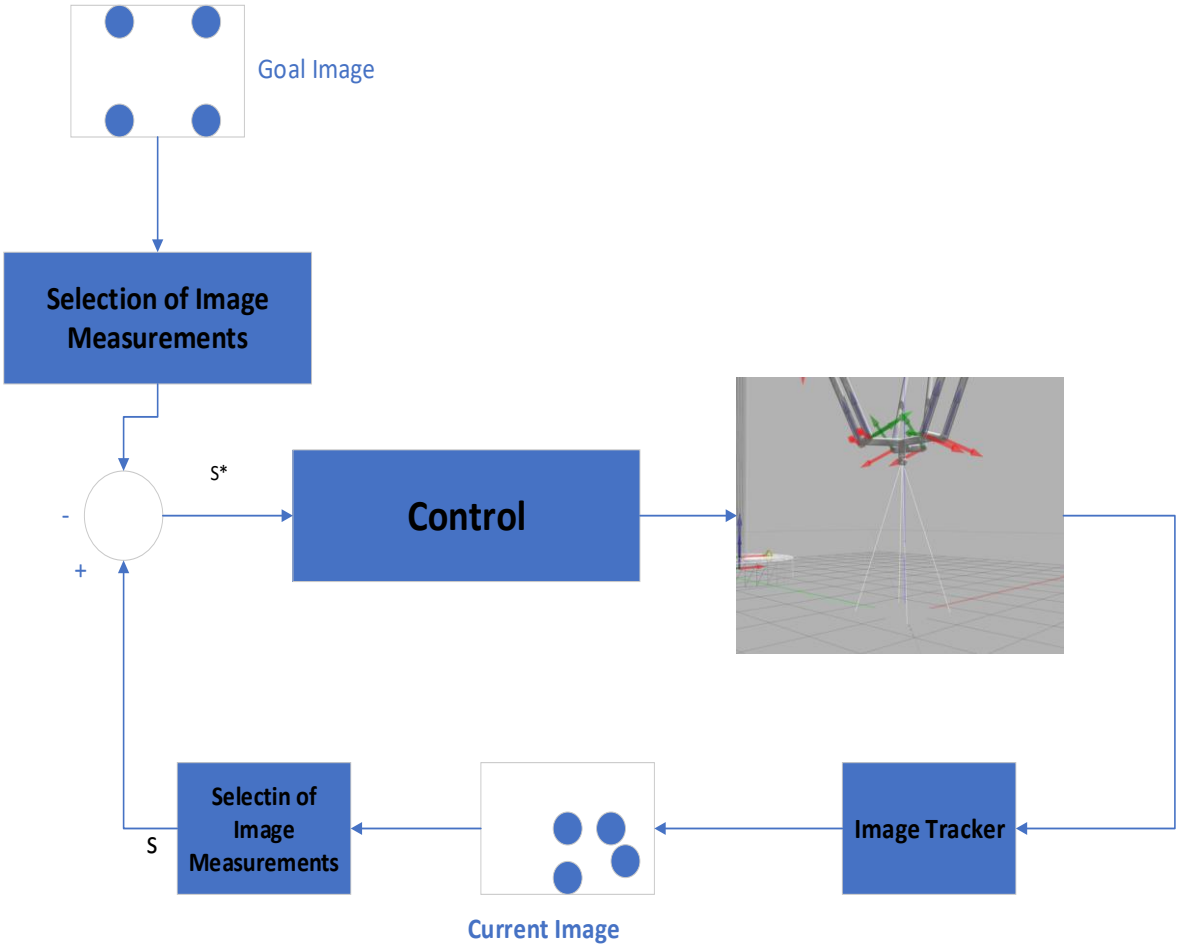

Figure 3-4: Task in Visual Servoing

To achieve a visual servoing positioning task, a parameterization s (q, t) ∈ R^m of a set of visual features, where q represents the rigid body state, is selected from the image of the object of interest. To be more precise, the visual measurements vector s(q, t) represents a parameterization of a configuration of a set of visual features. The primary control principle is to regulate the error

$e(q(t), t) = s(q(t), t) - s^*(q^*(t), t)$, where the goal values $s^*(q^*, t)$ corresponds to the goal state $q^*$of the system. That is what we want to achieve

$$\lim_{t\to\infty} \|e(t)\| = 0. \qquad (3.11)$$

One approach to achieving this objective is to compute the error dynamics differential equation

$$De/dt = J(t) \cdot q(t) + \partial e(t)\partial t \qquad (3.12)$$

Where q(t) is the system control inputs, $J(t) = \partial e(t)/\partial q(t)$ is the feature Jacobian matrix and $\partial e(t)/\partial t$ represents the contribution of a possible autonomous motion of the object of interest.

The control law is built from equation 3.10 using the knowledge of J(t) and $\partial e(t)/\partial t$ which in general are not determined exactly.

There are three main approaches in visual servoing: 3D, hybrid and image-based visual servoing. In 3D visual servoing, visual features in the 3D Cartesian space, represented by $SE(3) = R^3 \times SO(3)$, are used as control inputs [30]. These features are obtained from the relative pose of the camera with respect to the object of interest. This pose can be recovered knowing the model of the object (for example the Euclidean distances between salient points of the object). In hybrid (or so-called 2 1/2 D) visual servoing which combines image and 3D data, the orientation in SO(3) (which is the 3D data) of the camera between the current and the goal poses, are generally obtained via the estimation of an Euclidean homography transformation (mapping between points in two Euclidean planes) between the current and goal images. This method has the advantage over 3D visual servoing to not need the model of the object. However there is a drawback to the 2 1/2 D visual servoing method: it is more sensitive to image noise (like 3D visual servoing) than image-based visual servoing, which uses directly features extracted in the image as control inputs. The image-based visual servoing approach, as opposed to 3D and hybrid visual servoings, uses direct image feedback, and is thus very suitable for unstructured environment, for instance without camera and object model. In this thesis 3D position based visual servoing is used to extract 3d Cartesian space position information which later used to find the corresponding joint angles for the deep neural network.

3.3. Neural network control Methods

The main task of this thesis is to design a controller which guides the delta 3 robot end effector to the desired position only using vision as a feedback. There are a few important points to remember

when designing a controller for a parallel robot. Delta 3 robot is open-loop unstable, non-linear and a multiple input multiple output system.

Nonlinear system: ANN's have shown that they are capable of identifying complex nonlinear systems. They should be well suited for generating the complex internal mapping from inputs to control actions.

Multi-output system: Delta 3 robot has three input and three outputs, in order to have full state feedback control neural networks have a big advantage here due to their parallel nature. One ANN could be used to control the three outputs.

Before the actual neuro-controller is developed in ROS, the main types of neuro - control are discussed. The five types of neural network control methods that have been researched. This are supervised, model reference control, direct inverse, internal model control and unsupervised control.

3.3.1. Supervised Learning

It is possible to teach a neural network the correct actions by using an existing controller or human feedback. This type of control is called supervised learning. But why would we want to copy an existing controller that already does the job is a good question. Most traditional controllers (feedback linearization, rule-based control) are based around an operating point. This means that the controller can operate correctly if the plant/process operates around a certain point. These controllers will fail if there is any sort of uncertainty or change in the unknown plant. The advantages of Neuro-control is if an uncertainty in the plant occurs the ANN will be able to adapt its parameters and maintain controlling the plant when other robust controllers would fail. In supervised control, a teacher provides correct actions for the neural network to learn. (Fig. 3.5) In offline training the targets are provided by an existing controller, the neural network adjusts its weights until the output from the ANN is similar to the controller [3].

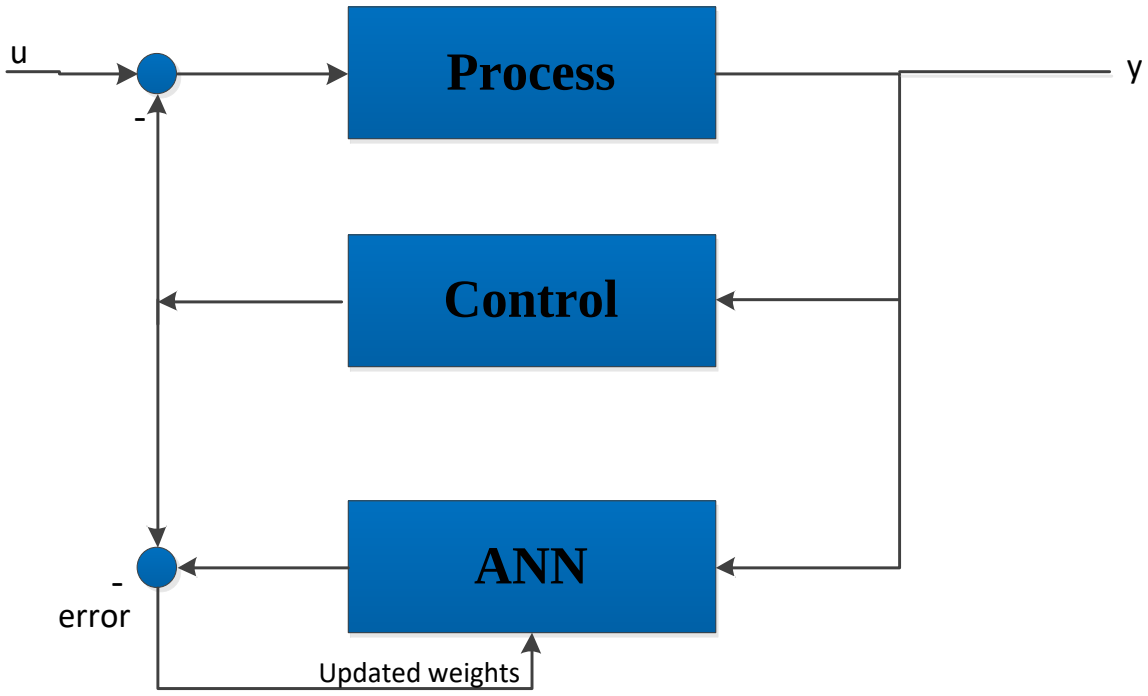

Figure 3-5: Supervised learning using an existing controller

When the neural network is trained, it is placed in the feedback loop. Because the ANN is trained using the existing controller targets, it should be able to control the process.

At this stage, there is ANN which controls the process similar to the existing controller. The real advantage of Neuro-control is the ability to be adaptive online. (Fig. 3.6) An error signal (desired signal – real output signal) is calculated and used to adjust the weights online.

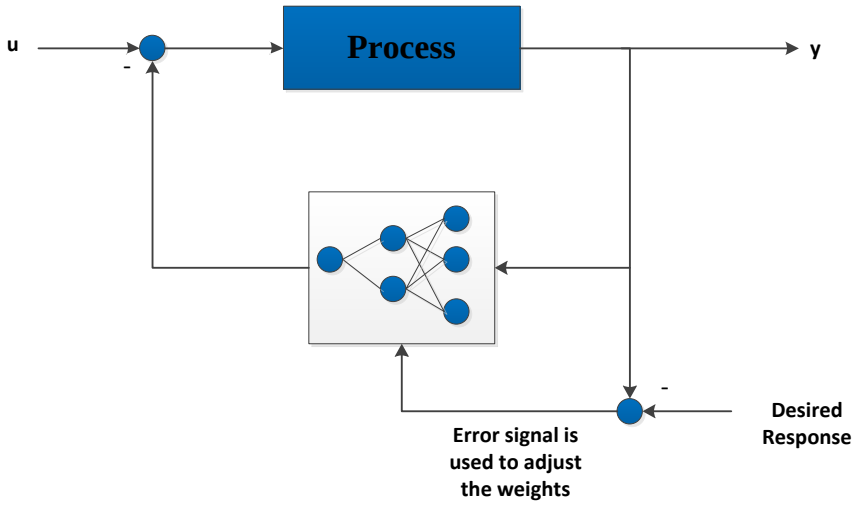

Figure 3-6: Adaptive neural control

If a large disturbance/uncertainty occurs in the process-the large error signal is feedback into the ANN and this adjusts the weights so the system remains stable.

3.3.2. Model Reference Control

In the diagram above (Fig.3.6) the error signal is generated by subtracting the output signal from the desired system response. In model reference control the desired closed loop response is specified through a stable reference model (Fig.3.7).The control system attempts to make the process output similar to the reference model output [31].

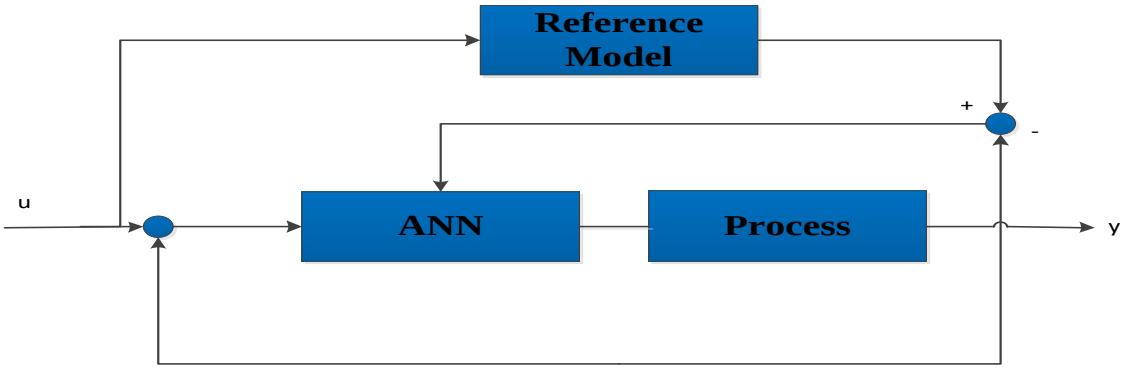

Figure 3-7: Model reference control

3.3.3. Direct inverse control

The next type of control technique researched is direct inverse control. The advantage of using inverse control over supervised control is that inverse control does not require an existing controller in training. Inverse control utilizes the inverse of the system model. The diagram below (Fig.3.8) is a simple example of direct inverse control. A neural network is trained to model the inverse of the process. When the inverse controller is cascaded with the process the output of the combined system will be equal to the set point. The inverse nonlinearities in the controller cancel out the nonlinearities in the process. For the nonlinearities to be effectively cancelled, the inverse model must be very accurate [32].

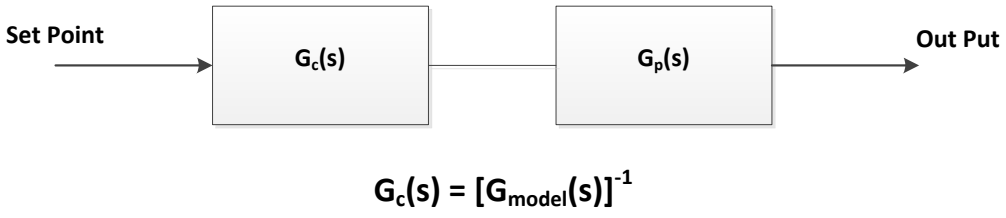

Figure 3-8: Direct inverse control

Inverse modelling is used to generate the inverse of the process. The system output is used as an input to the network. The ANN output is compared with the training signal (the system input) and this error signal is used to train the network. (Fig. 3.9) This training method will force the neural network to represent the inverse of the system.

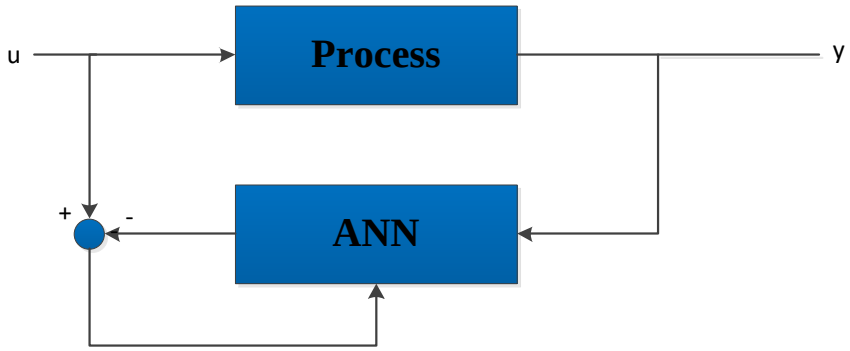

Figure 3-9: Inverse modelling of a process

There are certain problems associated with direct inverse control. There can be process-model mismatches. The training of the ANN for an inverse model might lead to the model being not strictly proper. This will lead to unknown disturbances in the system.

3.3.4. Internal model control

Internal model control is based on direct inverse control. The problems associated with direct inverse control such as process-model mismatch, etc are reduced using IMC. The diagram below shows the set-up of IMC (Fig. 3.10).

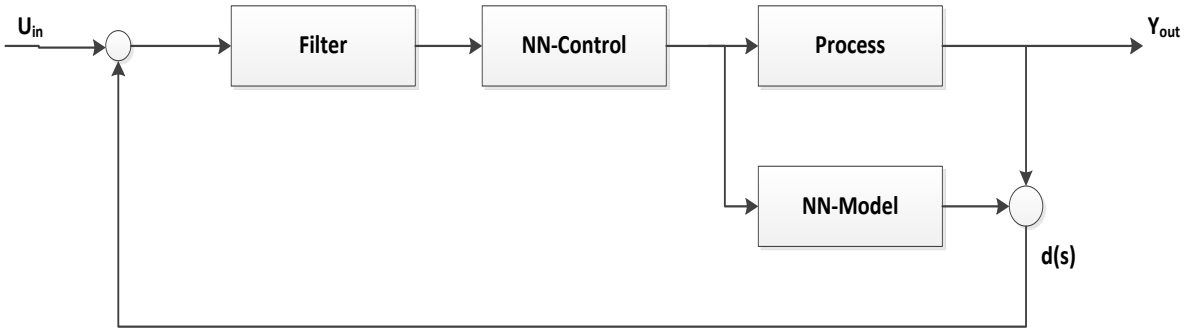

Figure 3-10: Diagram of Internal model control [30]

A neural network model is placed in parallel with the real system. The controller is an inverse model of the process. The filter makes the system robust to process-model mismatch. With the IMC scheme, the aim is to eliminate the unknown disturbance affecting the system. The difference between the process and the model d(s) is determined. If the ANN model is a good approximate

of the process then the d(s) is equal to the unknown disturbance. The signal d(s) is the information that is missing from the NN-model and can be used to improve the control. The d(s) signal is subtracted from the input set point U_{in}. In theory, using this method it is possible to achieve perfect control [33].

3.3.5. Unsupervised Learning

The previous neural control methods are all trained using a priori knowledge such as an explicit teacher providing correct actions. In unsupervised learning set-up, no existing controllers can be imitated and the ANN doesn't have a target to compare to its output. The ANN must try different states and determine which state produces a good output. Learning from experience during periods of no performance feedback is difficult.

To determine the type of neural control to be used, the pros and cons of each of the control methods was determined specifically relating to the delta 3 robot and the availability of data. To develop a supervised neural controller for the delta 3 robot an existing controller or actual data of the robot position with its corresponding joint angles is required. Model of the delta 3 robot in 3D simulator tool was developed and using vision as a feedback, actual data (recorded Cartesian position , recorded joint angle) was collected, that was available for training .This training data could be used as a teacher. Inverse control and internal model control are both based on developing an accurate inverse model of the plant (Delta 3 robot). The problem with this method is the dynamics of the robot is very sophisticated. The unsupervised control of the Delta 3 robot is too complex for the project time frame. Therefor supervised neuro-controller is developed for position control of the Delta 3 robot. The overall system architecture block diagram is as shown in figure 3.11. As shown in the block diagram visual processing has for stages which are visual feedback sensor of 3d features, epipolar geometry, feature extraction and pose estimation. ROS will combine this steps and gives end effector pose estimation. This is the main reason for ROS to be widely used in vision systems.

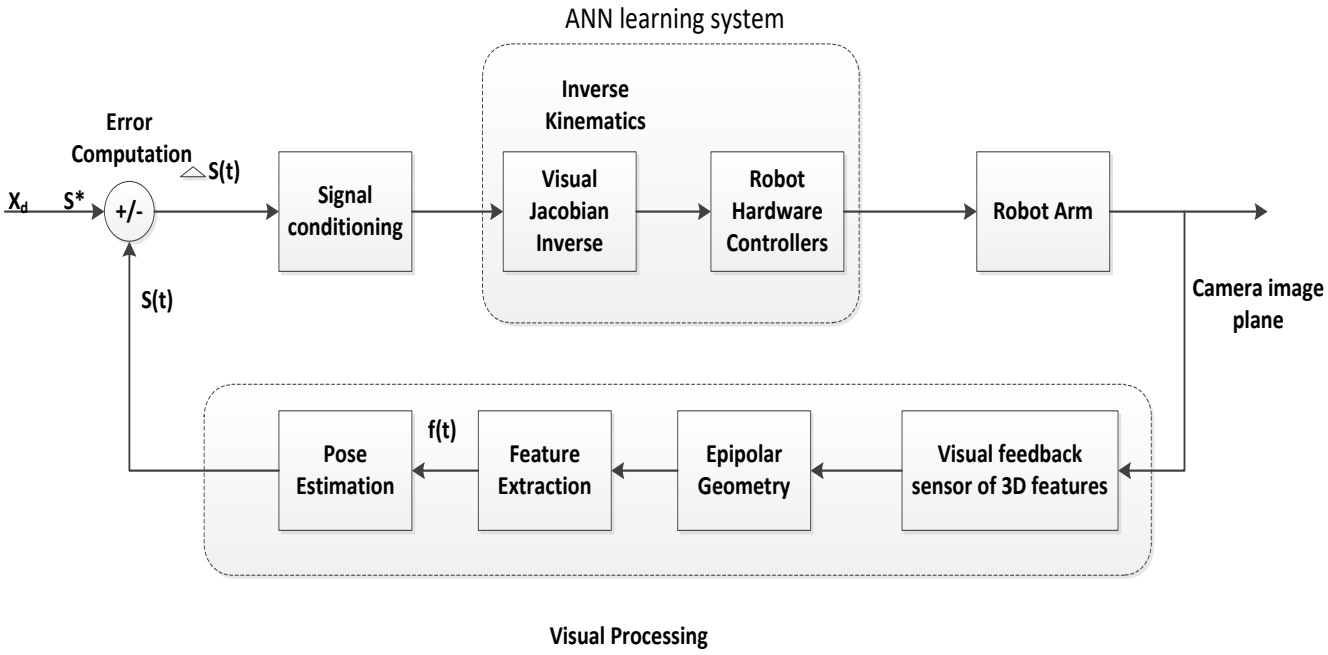

Figure 3-11: Closed-loop ANN based visual servo system

3.4. ROS (Robot Operating system)

Originally developed in 2007 by the Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot project, ROS is an open source operating system or robots. It offers hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It has enormous possibilities and allows simulation of gravity, creating a dynamic environment, different types of robots with many degrees of freedom, the creation of sensors of various kinds. The first step in modeling in ROS requires creating a three dimensional model of the environment [34].

Currently, ROS is predominantly used for mobile robots and intelligent environments. However, when the developments in the field of sensors, actuators, mobile robots and Immobotic systems are integrated and merged together with architectural design the buildings will become more complex and would require stimulation in an environment like ROS and Gazebo before it is executed. Furthermore, it would bring about a change in the modeling process where it demands not only the modeling of the three dimensional images but also the kinematics of the structure. So for all these purposes the open source system ROS provides a virtual testing platform and the capabilities are limited only by the imagination of the engineer.

3.4.1. Delta 3 robot simulation model

The first step in modeling of a robot in ROS requires creating a three dimensional model of the environment. It is important to remember that all parts of the environment having a different degree of freedom of movement must be created separately. After having created the 3D models of individual components they are combined in ROS to whole environments. This is possible with help of the package robot model, which contains the parser SDF (Simulation description format) files. The SDF format is itself described using XML. Also the SDF descriptions link different meshes/links through the concept of links and joints. Two links connected through a joint are referred as a parent link and child link and the type of movement of each element in the model is determined by the type of joint between the two links. The joint type used in modeling delta 3 Robot is Revolute joint. This type of Joint rotates around an axis and the movement is restricted within the specified upper and lower limits. The sdf file Contains all the elements in a simulation, including robot, lights, sensors (for the purpose of this thesis camera sensor), and static objects description of models to import into the ROS. Such description includes the name of the object,

its mesh-structure, the shell for the calculation of the collision, additional information about the visual components of an object, such as color, texture, and others. Consider a more detailed composition in sdf:

- **<*robot*>** tag contains the name of the robot (delta3), which will be displayed in all subsystems of ROS. This tag contains a description of the whole environment or part thereof (for example, a robot and camera sensor).
- **<*link*>** tag also contains the name and represents a visual component of the simulated environment, or an object that is connected to other objects with help of joints (<joint> tags). Inside tag <link> you can see a description of the visual part of the model (<visual> tag), which contains information about the geometry of the object <geometry>. Possible values - <cylinder>, <box>, <sphere> and <mesh>.
- Tag **<*origin*>** adjusts the initial position of an object in space.
- Tag **<*collision*>** initiates the parameters of collision. This is possible due to the fact that ROS creates an invisible layer on top of the visual object that is a shell which can collide with other objects.
- Tag **<*joint*>** is a connection between two parts of an object <link>. Tag <joint> has description of a parent object and a child-object (their names), connection type, that is type of movement (revolute – rotation around some axis with upper and lower limit); also it has an information about location of the joint; information about controllers.

Sdf file is usually created in a text editor, for this thesis it was created in visual studio text editor. Of course in this case it is impossible to monitor in real time the changes in 3D space. However, ROS has a special package Rviz, which allows you to import sdf files and shows the model in three dimensions. In addition, Rviz gives an opportunity to manipulate any objects that are included in the environment.

Launch file contains the path to sdf file as a parameter (a description of the robot robot_description) and the parameter for using the GUI for manual control of the environment with help of the sliders. Furthermore, the launch file runs the nodes "robot_state_publisher" and "joint_state_publisher", which necessary for the processing of information on the joints, as well as "package_Rviz", which shows us a three dimensional representation of the model. Actually, you can add any information, such as view from the camera, even motion planning if you already programmed the robot.

Unfortunately, Rviz can only show us the three dimensional representation of the robot or an objects environment, but not able to create the emulation of the world. This is a complex problem is solved by a package Gazebo, simulating gravity, which allows you to import many models and gives them an opportunity to communicate with each other.

It is necessary to ensure that delta parallel robot has a reasonable workspace volume. Workspace analysis is also required in the design of the parallel robot. For this thesis the desired work volume is 400x300x200mm$^{3.}$ Input kinematic parameters for the desired work space volume is as shown as in table 3.1 below.

Table 3-1: Delta 3 Robot 3D model Kinematic Parameters

Kinematic Parameters of Delta 3 Robot	Length (mm)
Bicep Length	75
Forearm Length	400
Base Triangle Side Length	500
End Effector Triangle Side Length	100
Ball Joint Pivot Angle Range	+/- 60 degrees
Motor Axis Revolute Joint Restriction	-60 to +130 degrees

Sdf file used for modeling of the delta 3 robot is in Appendix A. By following the steps descried below the delta 3 robot with eye in hand configuration was simulated.

- load a world
- load the description of the robot
- spawn the robot in the world
- publish joints states
- publish robot states
- run rviz

The simulation result is presented in chapter four.

3.5. Design and Training of NN for Visual Servoing Controller design (Structure)

3.5.1. ANN structure

3.5.1.1. Inputs and Outputs

This thesis used simple yet powerful structure to construct the DANNC for the delta three parallel robot In order to simplify the problem of inverse kinematics for every joint, one ANN is constructed for each input. In other words ANN controller consists of three separate networks. The input of every network is position in Cartesian coordinate (x,y,z) from the camera .The output is joint angle of the delta three robot (θ_1, θ_2, θ_3). Robot dynamics is assumed to be constant.

3.5.1.2. Network Structure

Multi-layer perceptron model which consists of three hidden layers each with Rectified Linear Unit (ReLU) activation Function and output layer with linear activation function network structure is used as shown in the fig. 3.15.

The rectified linear unit (ReLU) activation function was proposed by Nair and Hinton 2010, and ever since, has been the most widely used activation function for deep learning applications with state-of-the-art results to date. The ReLU is a faster learning activation function, which has proved to be the most successful and widely used function. It offers the better performance and generalization in deep learning compared to the Sigmoid and tanh activation functions. The ReLU represents a nearly linear function and therefore preserves the properties of linear models that made them easy to optimize, with gradient-descent methods. The ReLU activation function performs a threshold operation to each input element where values less than zero are set to zero thus the ReLU is given by

$$f(x) = \max(0, x) = \begin{cases} x_i, & if \ \ x_i \geq 0 \\ 0, & if \ \ x_i < 0 \end{cases}$$

This function rectifies the values of the inputs less than zero thereby forcing them to zero and eliminating the vanishing gradient problem observed in the earlier types of activation function. The ReLU function has been used within the hidden units of the deep neural networks with another activation functiom, used in the output layers of the network with typical examples found in image processing and speech recognition applications. The main advantage of using the rectified linear

units in computation is that, they guarantee faster computation since it does not compute exponentials and divisions, with overall speed of computation enhanced. Which is the main objective of this thesis. Another property of the ReLU is that it introduces sparsity in the hidden units as it squishes the values between zero to maximum. However, the ReLU has a limitation that it easily over fits compared to the sigmoid function although the dropout technique has been adopted to reduce the effect of over fitting of ReLUs and the rectified networks improved performances of the deep neural networks. The ReLU has a significant limitation that it is sometimes fragile during training thereby causing some of the gradients to die.

It is proven that a network with this structure and enough amounts of neurons in its hidden layers can produce any function with limited number of discontinuity.

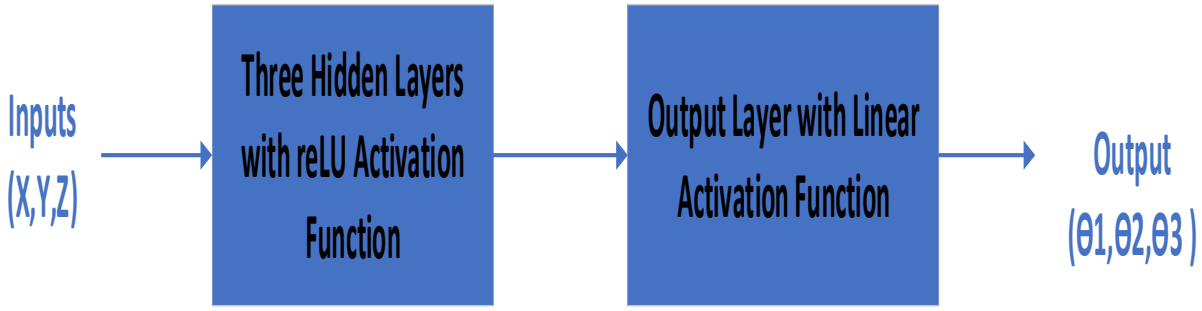

Figure 3-12: Deep Neural Network Architecture

3.5.2. Training Method

3.5.2.1. Designing a Training Data Set

The design of the training dataset is the most critical step in the process of training a NN, as it affects the ability of the training process to converge, as well as the precision and robustness of the end performances. Gathering real-world data is often cumbersome and sometimes unsuitable depending of the environment where the robot is expected to operate in. Furthermore, it can be difficult to re-create all possible conditions within the real-world environment. In order to overcome this problem this thesis uses the delta 3 model described above to collect the necessary data set for training. It contains all the necessary information for the NN to learn how to regress from an image to position input to joint angle output for the three DOF delta three robot .A code was written that moves the robot continuously in the working volume until it covers all the possible points in the work volume and at each camera pose (Io) the corresponding joint angles where recorded. The code used to implement this action is shown in appendix B. since the camera is

integrated at the end effector there was no image processing done to find the corresponding coordinates. Sixty six thousand three hundred thirty seven (66,337) data was collected within forty-eight hours that will be later used for training of the ANN. This data is presented in tabular format Appendix C.

3.5.2.2. Learning Function

The training function is the "fit" function from ROS library. “fit” is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization. It is often the fastest back propagation algorithm in ROS, and is highly recommended as a first-choice for supervised algorithm, although it does require more memory than other algorithms. Back propagation is used to calculate the network performance with respect to the weight and bias.

Chapter Four

4. Simulation Result and Discussion

4.1. Simulation Model of delta 3 robot in Gazebo

The delta 3 eye in hand configuration simulation model on Gazebo Environment is a as shown in the figure 4.1. As shown in the model the camera is integrated at the end effector.

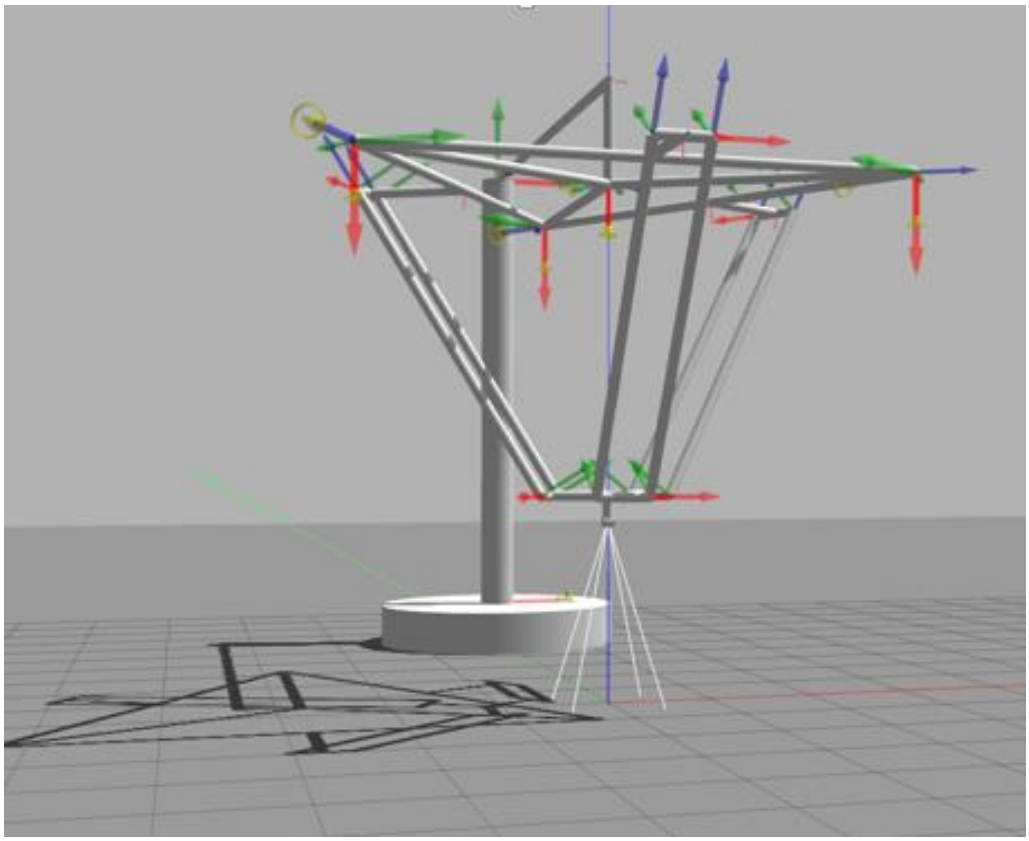

Figure 4-1: Delta 3 Eye in Hand Configuration simulation Model

The delta 3 eye in hand configuration model top view, side view, rare view and end effector integrated camera is as shown in the figure 4.2, figure 4.3, figure 4.4, and figure 4.5 respectively. The Sdf file used for modeling of the delta 3 robot is in Appendix A.

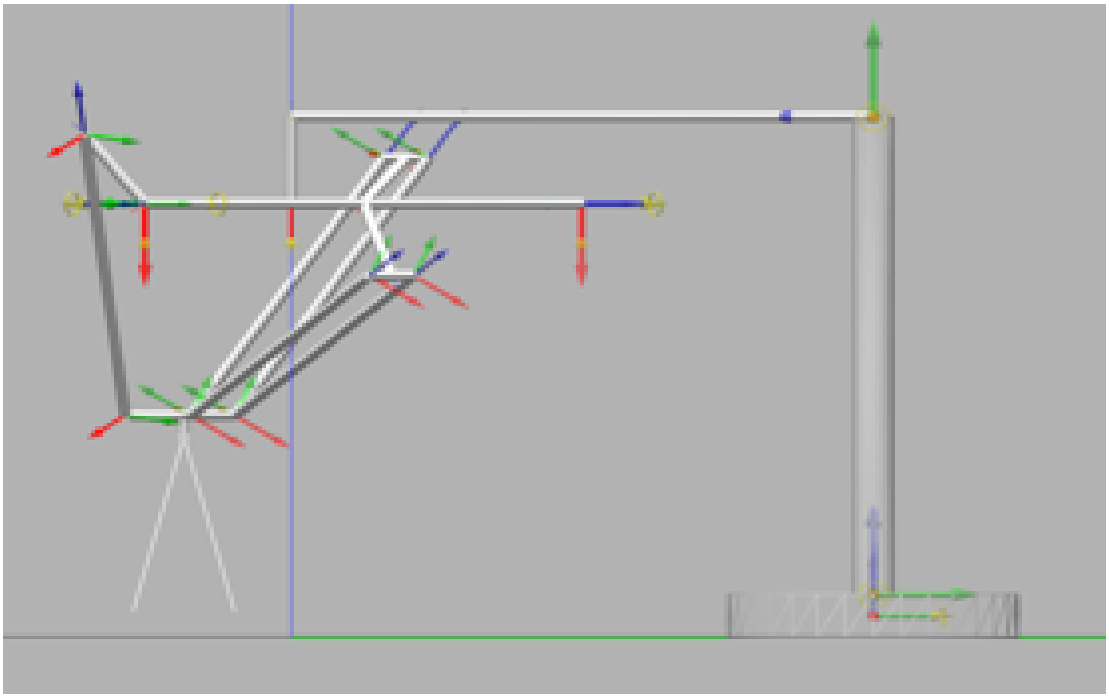

Figure 4-2: Side View of Delta 3 Robot Eye in Hand Configuration

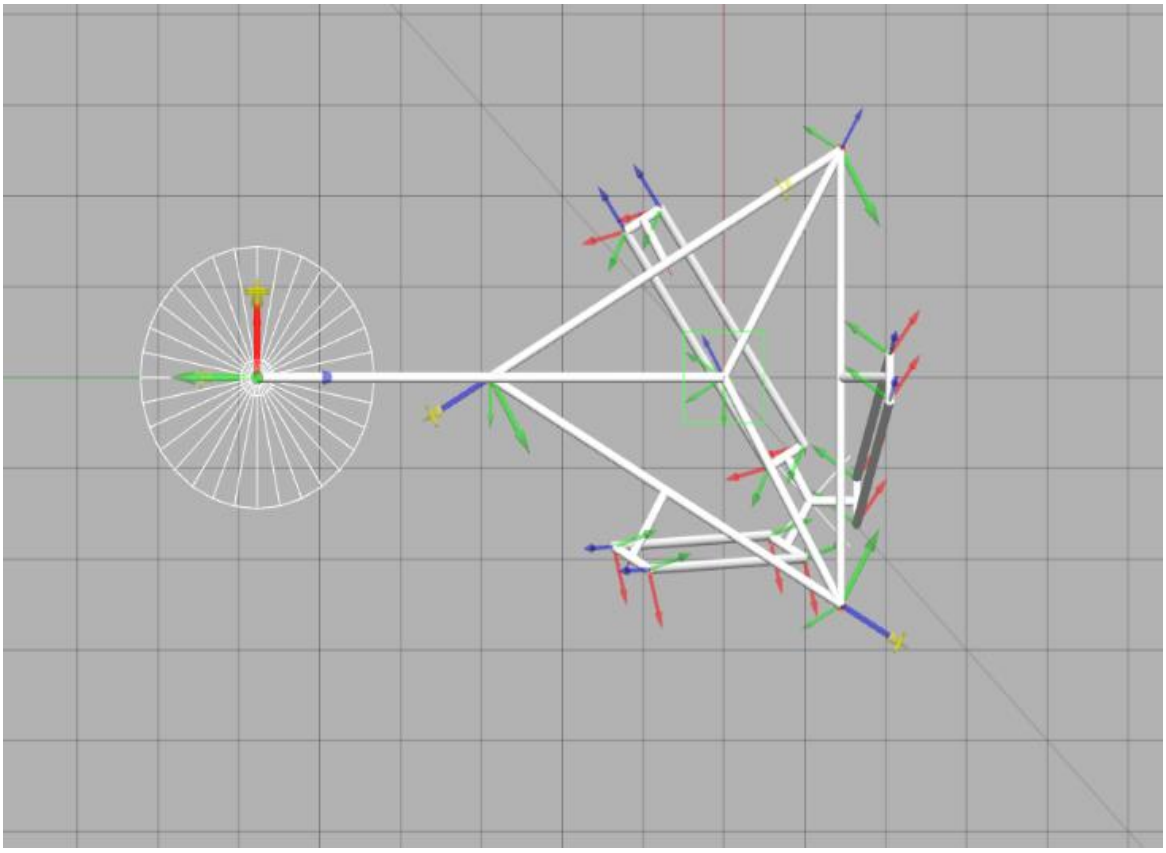

Figure 4-3: Top View of Delta 3 Robot Eye in Hand Configuration

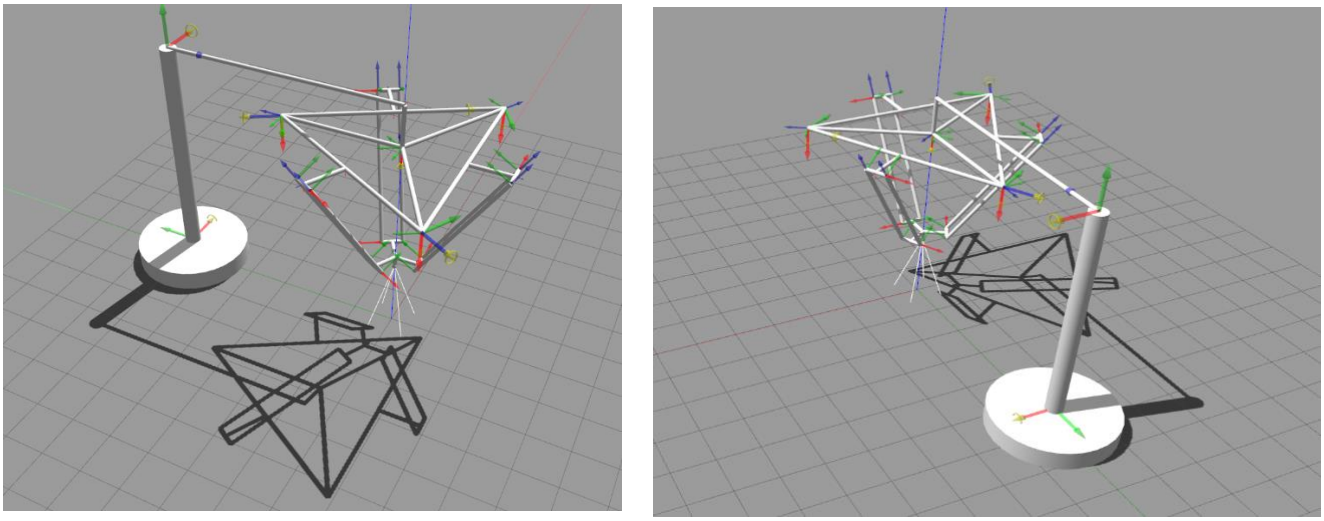

Figure 4-4: Rare View of Delta 3 Robot Eye in Hand Configuration

The partial source code used to model the delta three robot is presented in appendix A.

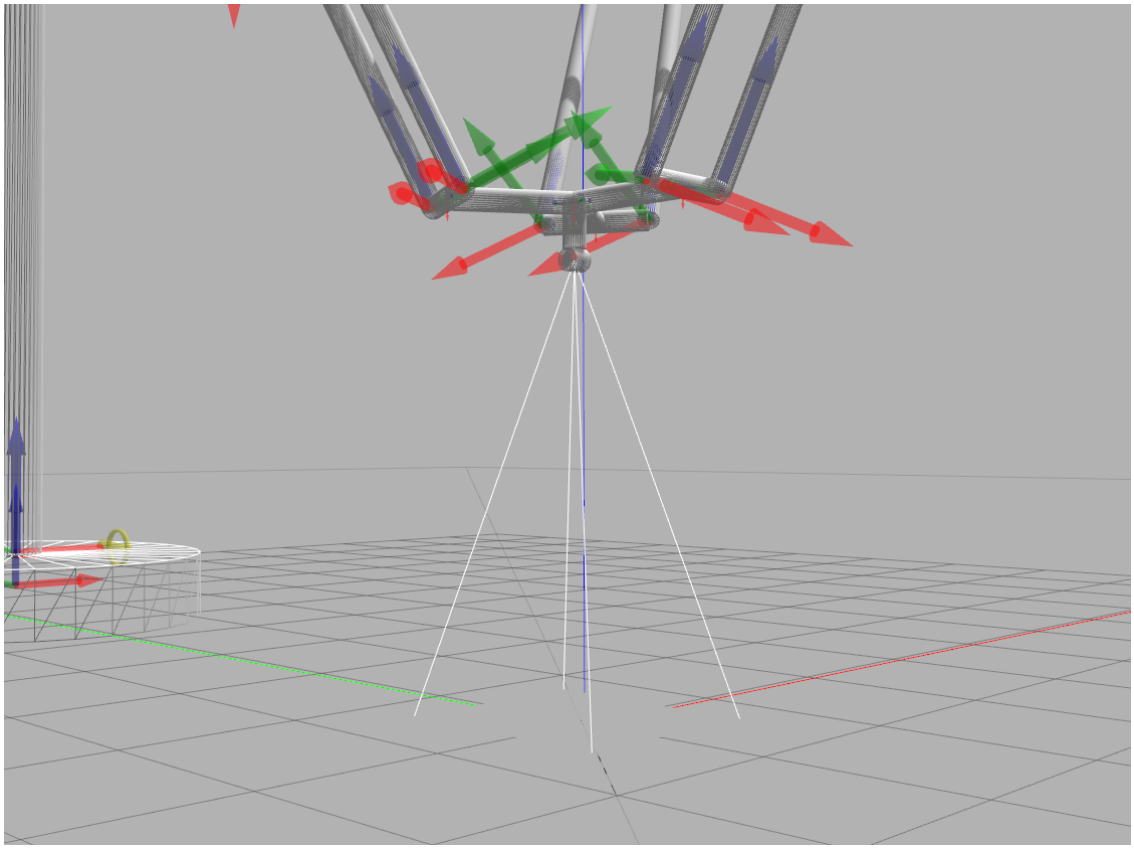

Figure 4-5: End effector integrated camera

4.2. Simulation of the Training Data Set

An algorithm was developed that moves the robot continuously in the working volume until it covers all the possible points in the work volume and at each camera pose(Io) the corresponding joint angles for training Data set. Figure 4.7 shows position of the end effector at a given time while figure 4.8 shows its corresponding joint angles that corresponds to the end effector position at a given time. Figure 4.8 integrates end effector position with the delta 3 robot joint angles at a given time. Sixty six thousand three hundred thirty seven (66,337) data was collected within forty-eight hours that will be later used for training of the ANN. This data is presented in tabular format Appendix C.

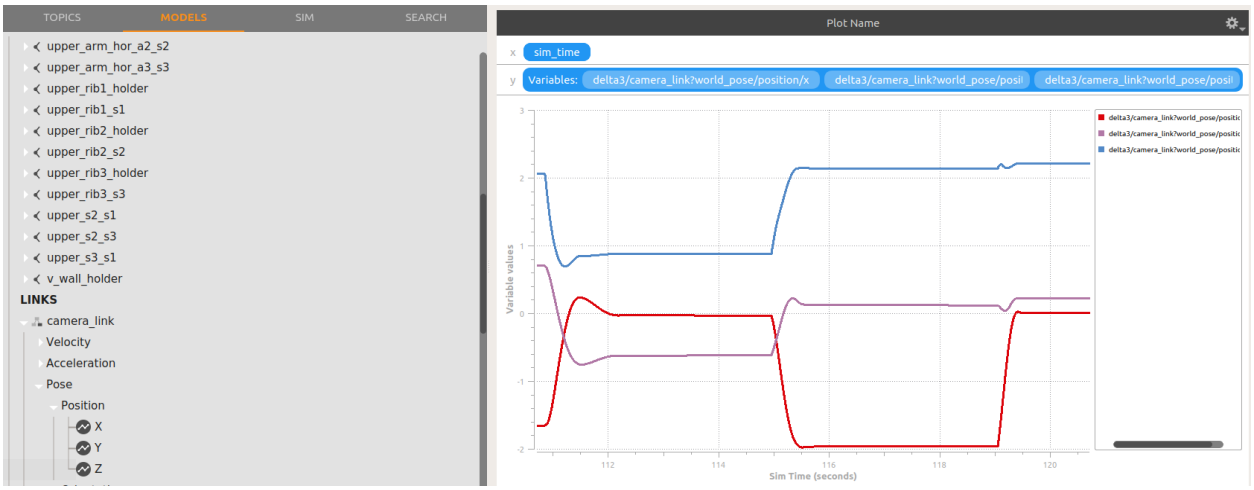

Figure 4-6: Delta 3 robot end effector position at a given time interval

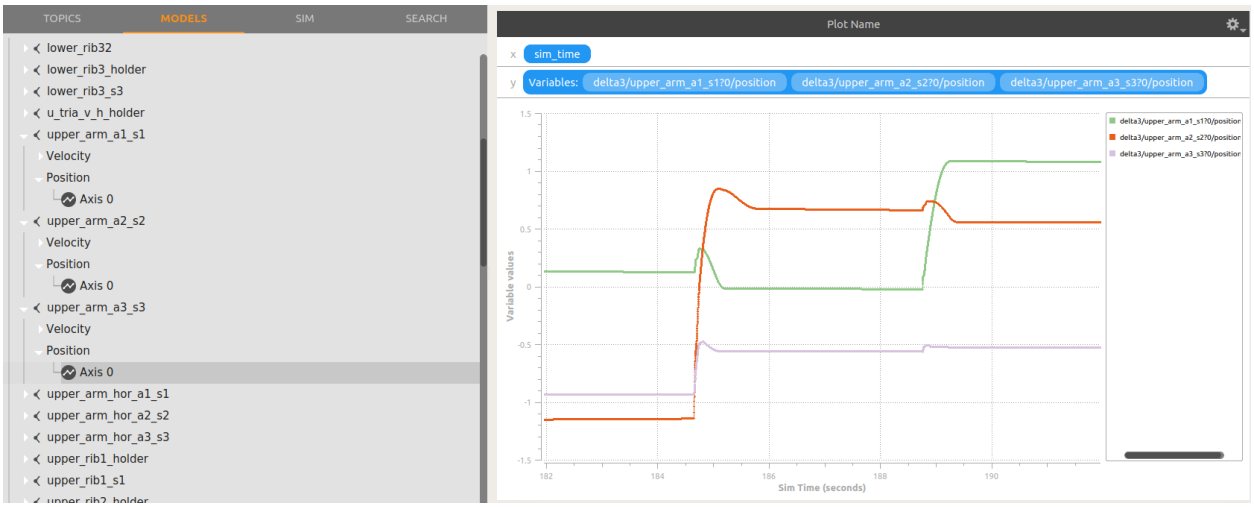

Figure 4-7: Delta 3 robot joint angle values at a given time interval

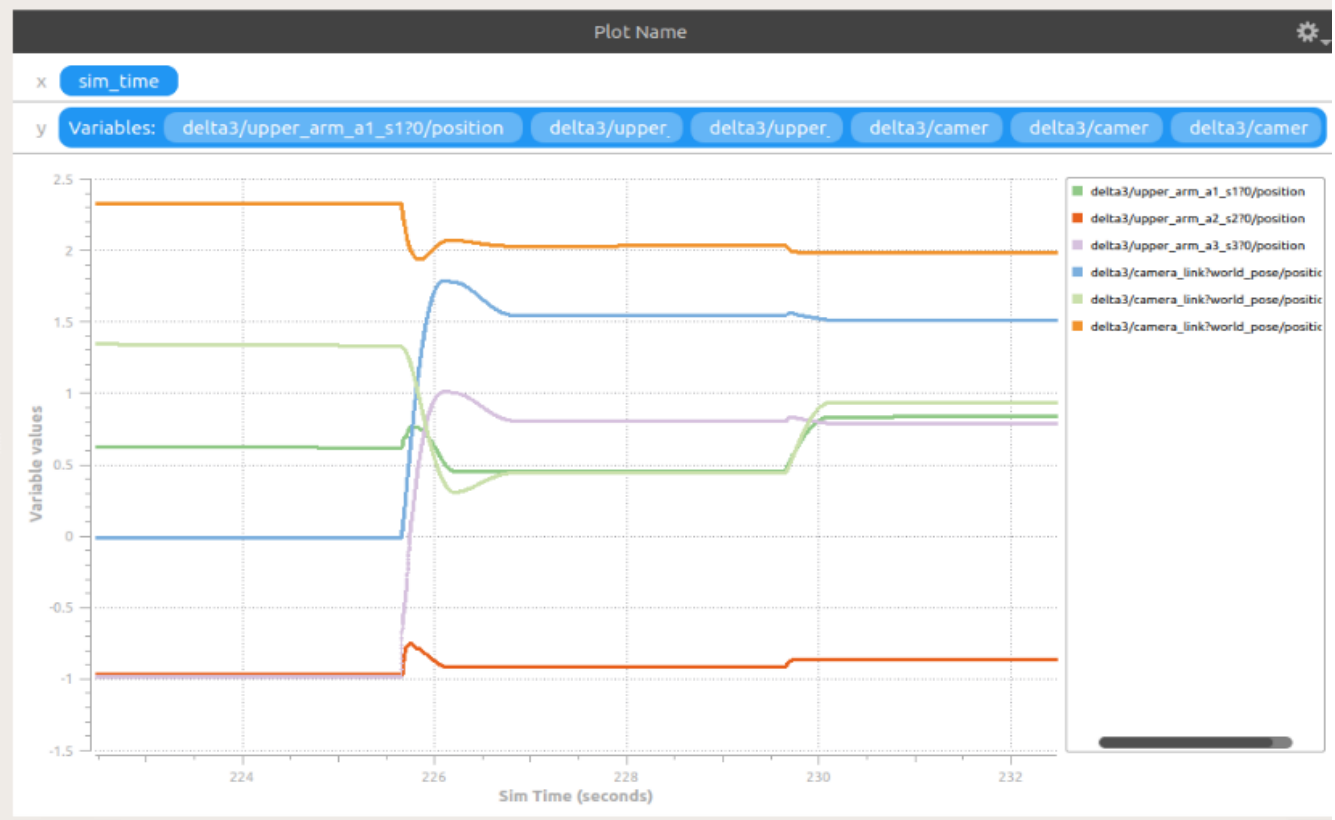

Figure 4-8: Delta 3 robot end effector position with its corresponding Joint angles at a given time interval

4.3. Performance of Trained Deep ANN

This thesis trains three networks and compare and contrast their performance based on robustness and response time. This three networks have three hidden layers but different number of neurons per hidden layer. The first network has 5 neurons per hidden layer. The second network to be trained has 20 neurons per hidden layer and the third network has 40 neurons per hidden layer. All three network has ReLU activation function in there hidden layer and a linear function in the output layer as described in chapter three. These networks use the back-propagation learning rule to update the weights. The number of epochs for each networks training is set to 200. During training, the input vector will be passed through the neural network and the weights will be adjusted 200 times. The learning rate of the network is also set to be 0.001 for all the three networks. The 'fit' function adjusts the weights of the network so that output of the network will be similar to the training data that was collected from delta 3 robot model. From the Sixty six thousand three hundred thirty seven (66,337) training data sets that was collected earlier, randomly 70 % of the data is used for training and 30 % of the data is used for validation.

The following diagram shows the training of the MLP. (Fig. 4.9) for the first network which is A Feed-Forward Network, 3 Hidden layer with 5 hidden neurons per hidden layer. At the start of the training, the error between the network and the validation data is high. As the number of epochs increases the mean squared error decreases. As the curve converges very fast and very little learning is taking place. Looking at the training diagram below it is possible to determine the correct amount of epochs for training.

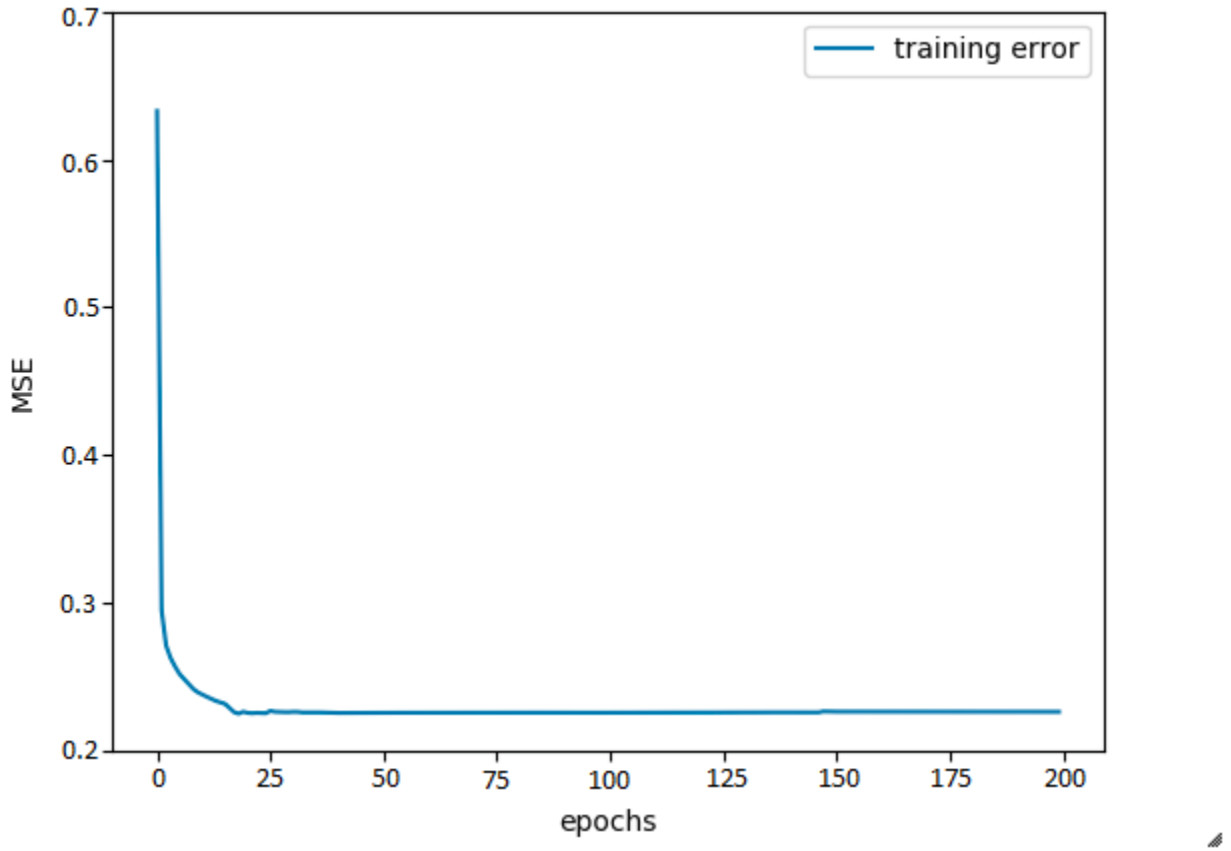

Figure 4-9: Feed-Forward Network, 3 Hidden layer, 5 hidden neurons per hidden layer

The second trained network is a Feed-Forward Network, 3 Hidden layer with 20 hidden neurons per hidden layer. Figure 4.10 shows the training of MLP. At the start of the training, the error between the network and the validation data is smaller compared to the above network. But As the number of epochs increases the mean squared error decreases. As the curve begins to converge, very little learning is taking place. Looking at the training diagram below it is possible to determine the correct amount of epochs for training.

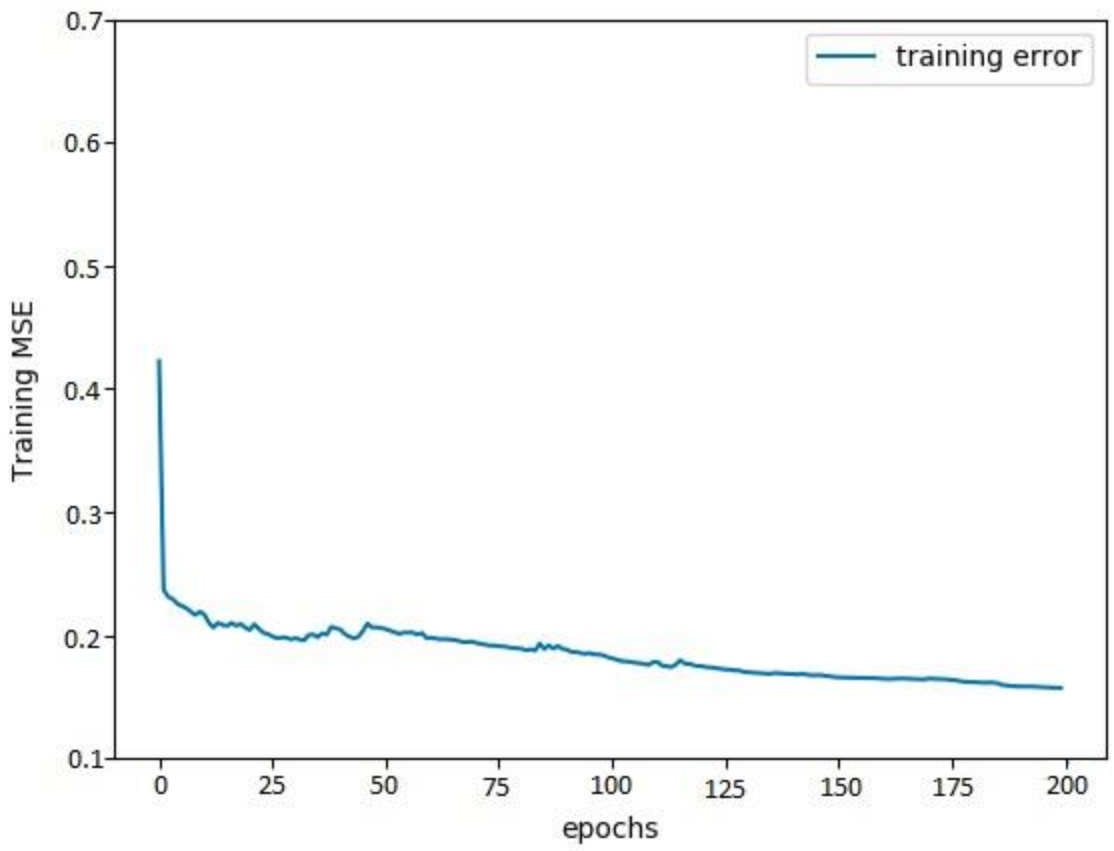

Figure 4-10: Feed-Forward Network, 3 Hidden layer, 20 hidden neurons per hidden layer

The third trained network is a Feed-Forward Network, 3 Hidden layer with 40 hidden neurons per hidden layer. Figure 4.11 shows the training of MLP. At the start of the training, the error between the network and the validation data is much smaller compared to the above two networks. As the number of epochs increases the mean squared error decreases significantly. As the curve begins to converge, very little learning is taking place. Looking at the training diagram below it is possible to determine the correct amount of epochs for training.

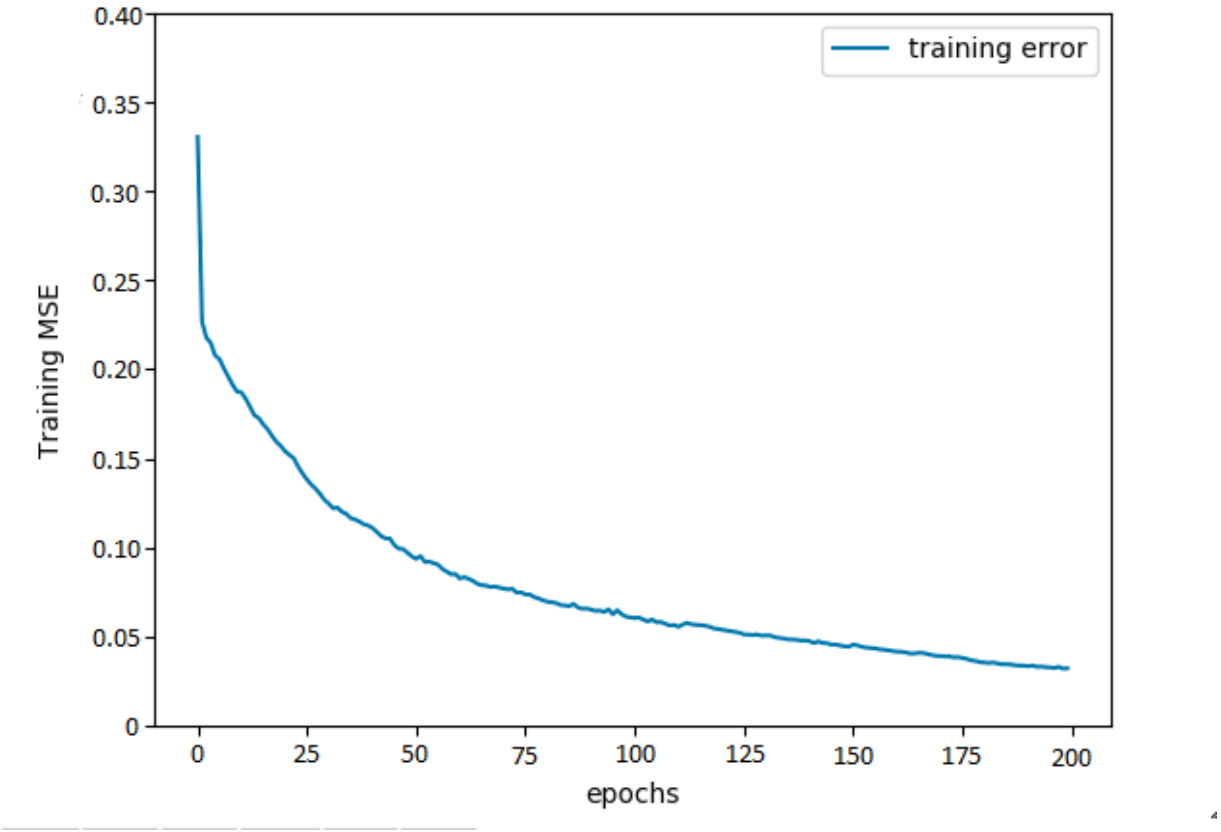

Figure 4-11: Feed-Forward Network, 3 Hidden layer, 20 hidden neurons per hidden layer

The quality of the neural model is tested by calculating the MSE (mean squared error). The MSE gives a good indication of the accuracy of the model. The MSE between the model and the process should be low. During testing, the three neural networks were simulated. It was expected that as the number of hidden neurons increased the more accurate the model would become and more computational time will take. Table 4.1shows the MSE of each joint angles. Keeping the learning rate, training epochs and training data constant, increasing the number of neurons per hidden layer decreases mean square error per joint angle as shown in the table.

Table 4-1: Simulation result of the Feed forward Deep Neural Networks

Type of ANN	Number of Hidden Layers	Number of Neurons In each Layer	Training Epochs	Learning Rates	MSE in degree		
					Θ_1	Θ_2	Θ_3
FF	3	5	200	0.001	0.159	0.108	0.0613
FF	3	20	200	0.001	0.1186	0.081	0.0489
FF	3	40	200	0.001	0.1015	0.0697	0.0384

The feed forward networks model the process well. The MSE is low and the neural model predicts the robot joint angle. The results indicate that increasing the number of hidden neurons does improve the MSE between the model and the process and the computational time is insignificantly the same among the three networks.

Most of the research in system identification uses open loop identification. Increasing the number of hidden layer neurons will have a direct influence on the accuracy of the model. This thesis also proved the same theory.

4.4. Performance of the controller

In order to evaluate the performance of the controller, a ball was placed in the simulation environment as shown in the figure 4.12. The goal is for the robot to locate the object, calculate the joint angles that will lead to the ball position and the robot end effector reach to the ball. The three networks that were trained above tested online.

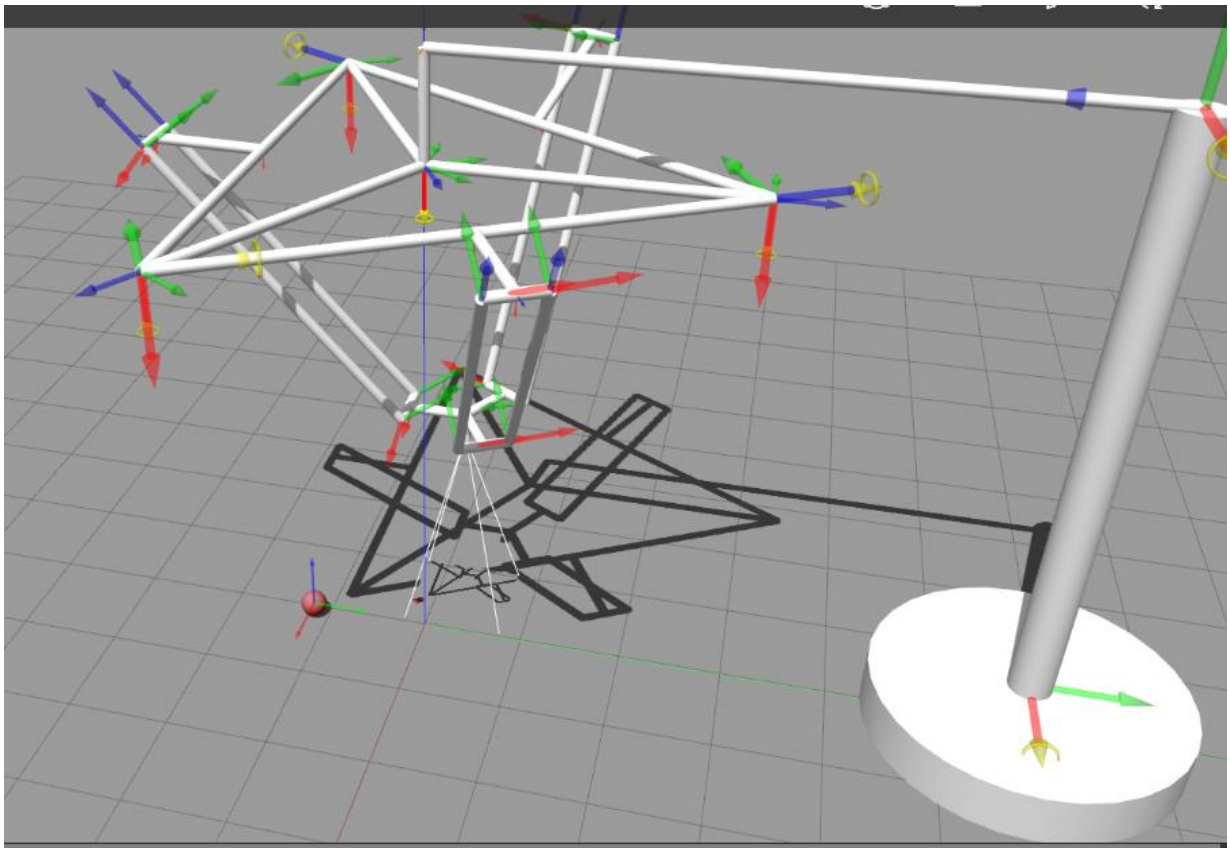

Figure 4-12: Initial position of the robot (The circle is target object)

Position of the ball is set to (0.101, -1.16, 1.168) ^{T}. The first network with 5 hidden neurons per hidden layer find the joint angles to be (-21, 10, 13) as shown in the figure 4.13 with MSE of 0.2283.

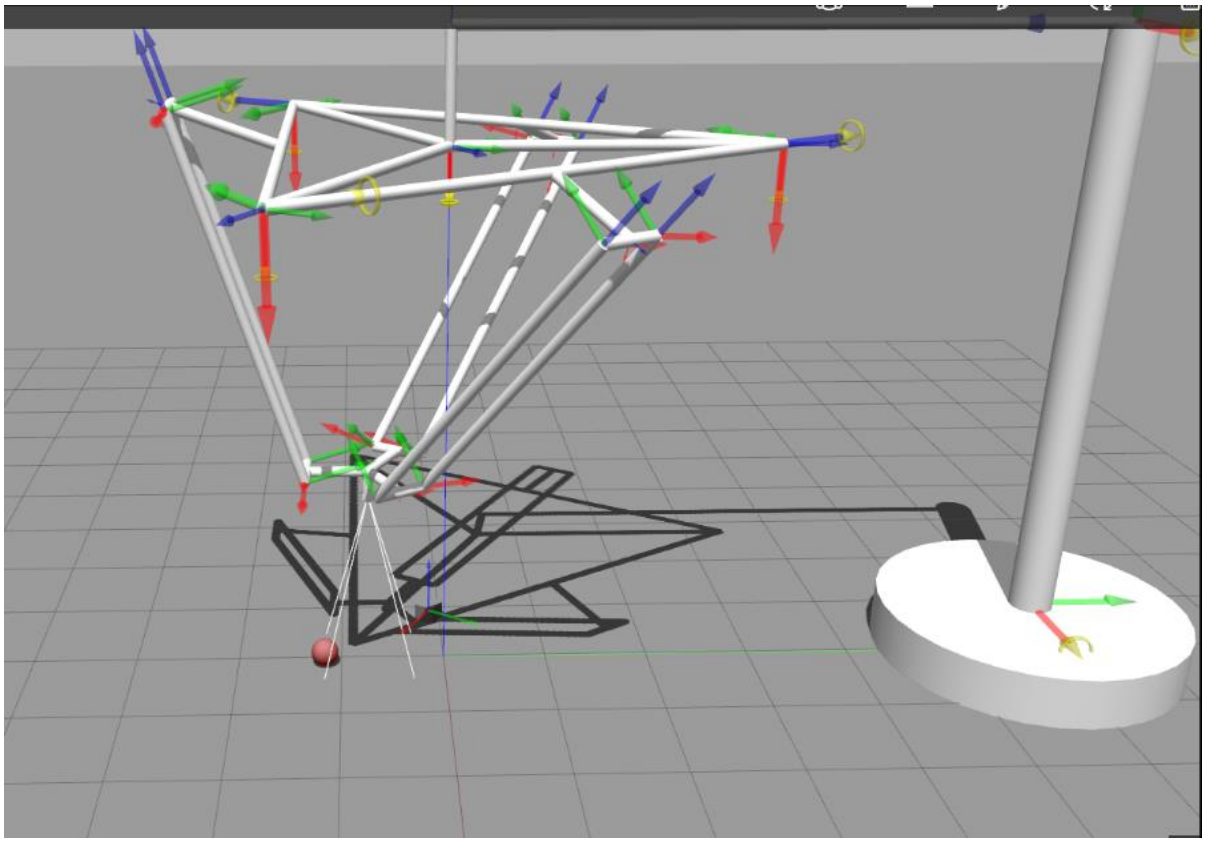

Figure 4-13: Final position for Feed-Forward Network, 3 Hidden layer, 5 hidden neurons per hidden layer

Position of the ball is set to (0.101, -1.16, 1.168) ^{T}. The second network with 20 hidden neurons per hidden layer adjust the joint angles to be (-21, 35, 35) as shown in the figure 4.14 with MSE of 0.0613.

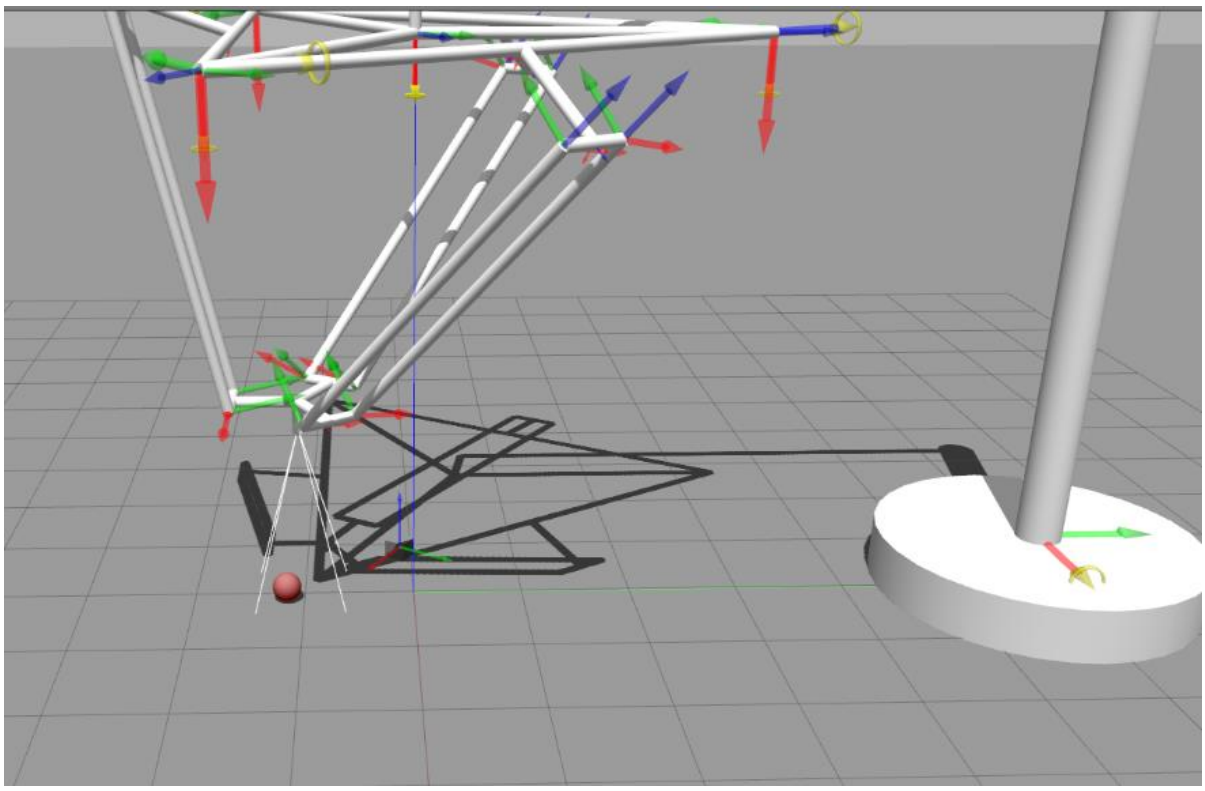

Figure 4-14: Final position for Feed-Forward Network, 3 Hidden layer, 20 hidden neurons per hidden layer

Position of the ball is set to $(0.101, -1.16, 1.168)^T$. The third network with 40 hidden neurons per hidden layer find the joint angles to be (-21, 20, 35) as shown in the figure 4.15 with MSE of 0.0028.

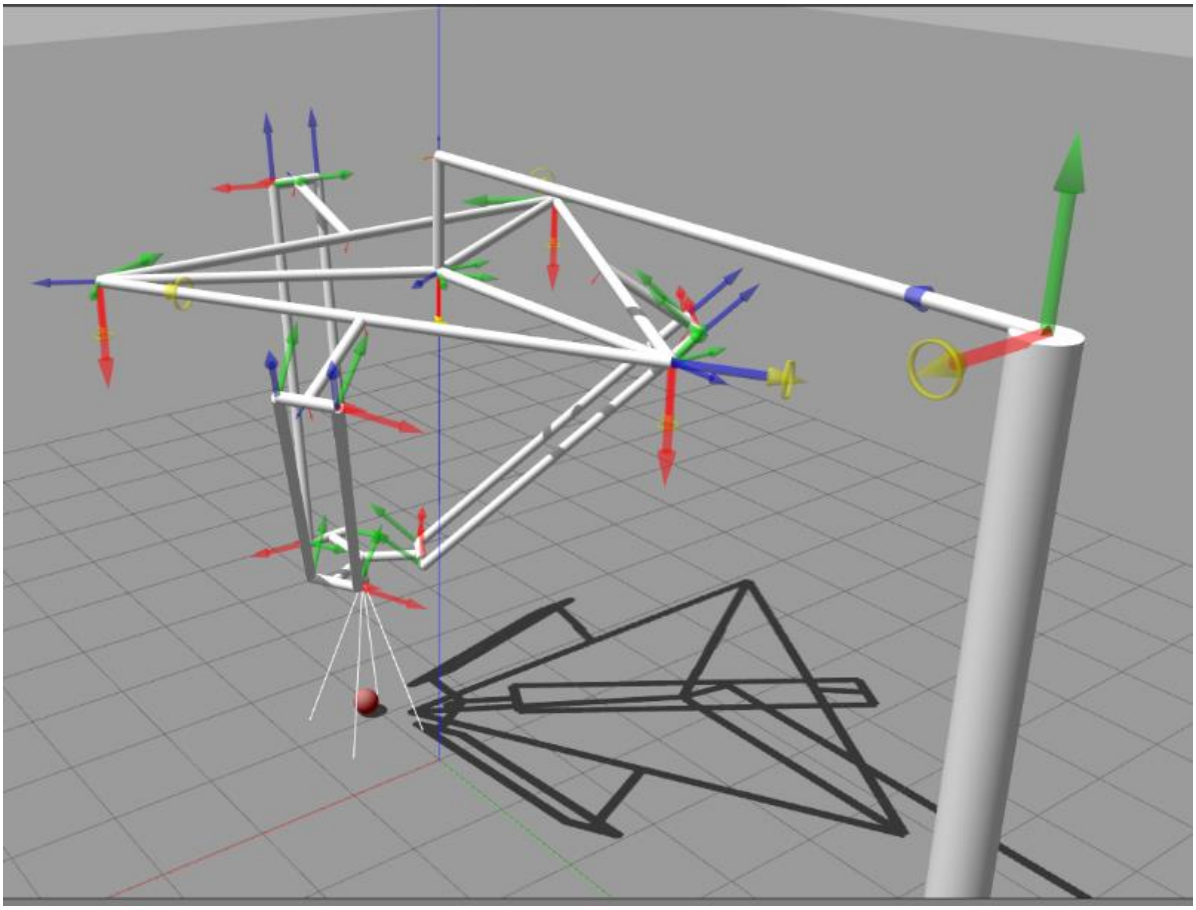

Figure 4-15: Final position for Feed-Forward Network, 3 Hidden layer, 40 hidden neurons per hidden layer

Chapter Five

5. Conclusion and Recommendation for Future Work

5.1. Conclusion

In this thesis, an ANN robot manipulator control system is developed for delta 3 parallel robot. One creative point of the project is position specific ANN. Another point is ANN combined with end-effecter integrated camera system. The advantage of position specific ANN is that it can avoid the difficulty of covering the whole joint space with one set of ANN, concerning the dramatically changing kinematics and dynamics of a manipulator. The advantage of end-effecter integrated camera is that, when the end-effecter is approaching the target, the target image caught by the camera is always the same, no matter where the target is in Cartesian coordinate. The output can be directly used as feedback, and no extra transformation is needed. In thesis presents a new generic method for robust visual servoing from deep neural networks.

Delta 3 robot with eye in hand configuration was modeled and simulated in ROS/Gazebo environment. This model is then later used to collect training data for the ANN supervised controller and for simulation and analysis of the result. An ANN control system is developed with five layer feed-forward network and a deep learning is performed by using back propagation algorithm.

Three ANN controllers where simulated and the performance of each controller was measured by using MSE. Since the main objective of this thesis is to design a controller that improves control accuracy, robustness and response time. By varying the number of neurons in each hidden layer, results shows that ANN controller with more number of neurons in hidden layer shows a better performance as a controller. Its MSE is find to be less than 0.05. Response time difference to execute task between the three networks is almost insignificant because of the learning function used in the hidden layer.

Generally this thesis established simpler and more feasible vision based robot control system for parallel robot by using deep neural network.

5.2. Recommendation

Based on the findings and conclusion of the study recommendations to be considered are using world class middleware robot like ROS makes vision based Robot modeling and simulation, quick and easily make good solutions reusable, training a network by using deep neural network architecture gives a great advantage in order to increase its accuracy and robustness with enough amount training data sets.

The current framework allows delta 3 parallel robot to visual servo with respect to a single scene, which forms the basis of the training set. Changing the application scene only requires synthesis of a new, comparatively small, training dataset and fine tuning of the network to generate the desired pose estimates. Future research will focus on extending the proposed method to generalize to multiple scenes, eventually training a network that provides scene-agnostic relative camera pose estimations.

References

[1] V. D. Zoran PANDILOV, "COMPARISON OF THE CHARACTERISTICS BETWEEN SERIAL AND PARALLEL ROBOTS," *ACTA TEHNICA CORVINIENSIS – Bulletin of Engineering,* vol. VII, pp. 15-16, 2014.

[2] P. M. G. Y. D. Patel, "Parallel Manipulators Applications," pp. 58-58, February 2012.

[3] M. a. D. Hagan, *H- Neural Network Design,* Boston, 1996 .

[4] G. R. J. D.B.Anuradha, "Direct Inverse Neural Network Control of A Continuous Stirred Tank Reactor (CSTR)," in *Proceedings of the International MultiConference of Engineers and Computer Scientists* , Hong Kong, 2019.

[5] T. HAMID D., "Parallel Robots," in *Parallel Robots Mechanics and ControlMechanics and Control*, London, CRC press Taylor and Fransis Group, pp. 13-19.

[6] H. D. T. Hirad, Parallel Robots Mechanics and Control, London: Taylor & Francis Group, LLC, 2013.

[7] I. Bonev., "," January 12 2017. [Online]. Available: http://www.parallemic.org/Reviews/Review007.html.

[8] V. E. Gough, "Contribution to discussion to papers on research in automobile stability on control and in tyre performance," in *Proceedings of the Automotive Division Instrument Engineering*.

[9] K. Cappel, "Motion simulator," January 3,1967.

[10] 1. January 3, "The Delta Parallel Robot: Kinematics Solutions," www.ohio.edu/people/williar4/html/pdf/DeltaKin.pdf, 2016.

[11] P. Robert L. Williams II, The Delta Parallel Robot: Kinematics Solutions, ohio: Mechanical Engineering, Ohio University, October 2016.

[12] N. J. Nilsson, Introducton to Machine Learning, Stanford : Stanford University, November 3, 1998.

[13] ---, Fundamentals of Machine Learning.

[14] T. O. Ayodele, "Types of Machine Learning Algorithms," United Kingdom, University of Portsmouth, pp. 189-210.

[15] F. C. a. S. Hutchinson, "Visual servo control, part i: Basic approaches," *IEEE Robotics and Automation Magazine,* vol. 13, pp. no. 4, pp. 82–90,, December 2006.

[16] O. F. a. R. N. M. Jagersand, "Experimental evaluation of uncalibrated visual servoing for precision manipulation," in *IEEE Int. Conference on Robotics and Automation*, 1997.

[17] J. B. a. J. U. D. Kuhn, "Neural approach to visual servoing for robotic hand eye," in *IEEE International Conference on, vol. 5, nov/dec*, 1995.

[18] Y. Z. a. C. Cheah, "Vision-based neural network control for constrained robots with constraint uncertainty," in *Control Theory Applications*, october 2008, p. pp. 906 –916.

[19] A. R.-M. a. Y. T. M. Gonzalez-Olvera, "Black-box modeling of a 2-dof manipulator in the image plane using recurrent neurofuzzy networks," in *Proceedings of the 48th IEEE Conference*, , 2009.

[20] K. W. a. K. I. S. F. M. Assal, "“Neural network-based kinematic inversion of industrial"," *IEEE/ASME Trans. on,* pp. pp. 593-603, Oct., 2006.

[21] S. X. Y. a. G. S. M. Jin Pei, "Vision Based Robot Control Using Position Specific Artificial Neural Network," in *2010 International Conference on Computational Intelligence and Communication networks*, canada, 2010.

[22] H. Al-Junaid, "ANN Based Robotic Arm Visual Servoing Nonlinear System," in *The 2015 International Conferenceon Soft Computing and Software Engineering*, 2015.

[23] E. G. E. A. B. a. A. S. Tarik Uzunovic, "Configuration Space Control of a Parallel Delta Robot with a Neural Network Based Inverse Kinematics," Sabanci University, Istanbul, Turkey.

[24] R. a. McClelland, "A General Framework for Parallel Distributed Processing," 1986 , Sabanci University, Istanbul, Turkey.

[25] S. E. a. C. Lewis, "Ros-industrial:applying the robot operating system (ros) to industrial," in *IEEE Int. Conference on Robotics and Automation, ECHORD Workshop*, St Paul, Minnesota, 2012.

[26] B. G. K. C. J. F. T. F. J. L. E. B. R. W. a. A. N. M. Quigley, "Ros: an open-source robot operating system," in *EEE Int. Conference on Robotics and Automation, Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[27] G. D. H. a. P. C. Seth Hutchinson," A Tutorial on Visual Servo Control", New Haven, April 14, 2012.

[28] F. C. a. S. Hutchinson, "Visual servo control, part i: Basic approaches," *IEEE Robotics and Automation Magazine,* p. 82–90, December 2006.

[29] "Visual servo control, part ii: Advanced approaches," *IEEE Robotics and Automation Magazine,* p. 109–118, march 2007.

[30] C. H. a. G. B. W. Wilson, "Relative end-effector control using cartesian position-based visual servoing," *IEEE Transactions on Robotics and Automation,* vol. 12, p. pp. 684–696, Oct. 1996.

[31] P. a. R. Marco, "Application of several neurocontrolschemes to a 2 DOF manipulator".

[32] N. N. D. T. Magnus Norgaard, "http://www.iau.dtu.dk/research/control/nnlib/manual.pdf," [Online].

[33] H. a. Sbarbaro, "Neural Networks for Control System -A Survey," 1992.

[34] S. E. a. C. Lewis, "Ros-industrial:applying the robot operating system (ros) to industrial applications," in *IEEE Int. Conference on Robotics and Automation, ECHORD Workshop*, St Paul, Minnesota, May 2012.

[35] S. X. Y. ,. a. G. S. M. Jin Pei, "Vision Based Robot Control Using Position Specific Artificial Neural Network," in *2010 International Conference on Computational Intelligence and Communication Networks*, 2010.

[36] E. G. ,. E. A. B. a. A. S. Tarik Uzunovic, *Configuration Space Control of a Parallel Delta Robot with a Neural Network Based Inverse Kinematics,* Sabanci University, Istanbul,Turkey.

[37] P. a. R. Marco, *Application of several neurocontrolschemes to a 2 DOF manipulator*.

[38] H. a. Sbarbaro, "Neural Networks for Control System -A Survey," 1992, pp. 1083 -1112.

[39] G. H. a. P. C. S. Hutchinson, "A tutorial on visual servo control," in *IEEE Transactions on Robotics and Automation*, Oct. 1996.

[40] "Visual servo control, part ii: Advanced approaches," *IEEE Robotics and Automation Magazine,* vol. 14, pp. pp. 109–118,, March 2007.

[41] A. M. &. C. S. S. G. M. Pranav, "A NOVEL DESIGN OF DELTA ROBOT," *International Journal of Multidisciplinary Research and Modern Education (IJMRME),* vol. II, no. II, 2016.

[42] A. Olsson, "Modeling and control of a Delta-3 robot," 2009.

[43] F. C. P. Q. a. M. G. Romeo Tatsambon, "Towards Practical Visual Servoing in Robotics," *Computer Vision and Robotics Research Group Department of Computing Science University of Alberta*.

[44] E. M. J. L. ,. F. C. P. C. Quentin Bateux, "Visual Servoing from Deep Neural Networks," 20`17.

[45] A. S. M. V. B. E. a. T. B. Thomas Linner, "MODELING AND OPERATING ROBOTIC ENVIRONMENTS USING GAZEBO/ROS," Technische Universität München, Germany, 2016.

[46] H. Al-Junaid, "ANN Based Robotic Arm Visual Servoing Nonlinear System," in *The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)*, 2015.

Appendix

Appendix A: Source Code for Modeling of Delta 3 Robot

```
<?xml version="1.0" ?>
<sdf version="1.5">
<model name="delta3">
  <static>false</static>

  <joint name="fixed_to_ground" type="revolute">
    <parent>world</parent>
    <child>wall</child>
    <axis>
                <xyz>0 1 0</xyz>
                <limit>
                  <upper>0</upper>
                  <lower>0</lower>
                </limit>
              </axis>
 </joint>
  <link name="upper_side1">
    <pose>0.0 -1.4433756729740643 5.0 0.0 1.5707963267948966 0.0</pose>
    <gravity>0</gravity>
    <inertial>
      <mass>0.1</mass>
      <inertia>
        <ixx>0.0005</ixx>
        <iyy>0.0005</iyy>
```

```
            <izz>0.0005</izz>
          </inertia>
        </inertial>
        <collision name="col">
          <geometry>
            <cylinder>
              <radius>0.05</radius>
              <length>5</length>
             </cylinder>
          </geometry>
        </collision>
        <visual name="visual">
          <geometry>
            <cylinder>
              <radius>0.05</radius>
              <length>5</length>
            </cylinder>
          </geometry>
        </visual>
    </link>

    <link name="upper_side2">
      <pose>1.25 0.7216878364870318 5.0 0.0 1.5707963267948966
2.0943951023931953</pose>
      <gravity>0</gravity>
      <inertial>
        <mass>0.1</mass>
```

```
        <inertia>
          <ixx>0.0005</ixx>
          <iyy>0.0005</iyy>
          <izz>0.0005</izz>
        </inertia>
.
.
.
.

  <link name="camera_link">
    <inertial>
        <mass>0.01</mass>
        <inertia>
          <ixx>0.0005</ixx>
          <iyy>0.0005</iyy>
          <izz>0.0005</izz>
        </inertia>
    </inertial>
    <pose>0 0 1.7499999999999998 0 1.5707963267948966 0</pose>
    <visual name="visual">
      <geometry>
        <cylinder>
          <radius>0.05</radius>
          <length>0.1</length>
        </cylinder>
      </geometry>
```

```
      </visual>
      <collision name="col">
        <geometry>
          <cylinder>
            <radius>0.05</radius>
            <length>0.1</length>
          </cylinder>
        </geometry>
      </collision>
      <sensor type="camera" name.
.....
<child>horizontal_holder</child>
            <parent>main_holder</parent>
              <axis>
                <dynamics>
                  <friction>1.0</friction>
                  <damping>.1</damping>
                </dynamics>
                <xyz>1 0 0</xyz>
                <limit>
                  <upper>0</upper>
                  <lower>0</lower>
                </limit>
              </axis>
                          </joint>
        <joint type="revolute" name="v_wall_holder">
            <pose>0 0 -2.75 0 0 0</pose>
```

```
        <child>main_holder</child>
        <parent>wall</parent>
          <axis>
            <dynamics>
              <friction>1.0</friction>
              <damping>.1</damping>
            </dynamics>
            <xyz>1 0 0</xyz>
            <limit>
              <upper>0</upper>
              <lower>0</lower>
            </limit>
          </axis>
                    </joint>
    <joint type="revolute" name="camera_joint">
        <pose>0 0 0 0 0 0</pose>
        <child>camera_link</child>
        <parent>lower_triangle_holder</parent>
          <axis>
            <dynamics>
              <friction>1.0</friction>
              <damping>.1</damping>
            </dynamics>
            <xyz>1 0 0</xyz>
            <limit>
              <upper>0</upper>
              <lower>0</lower>
```

```
                </limit>
            </axis>
                    </joint>

    <plugin name="arm_control" filename="libdelta3_gazebo.so" />

</model>
</sdf>
```

Appendix B: Source Code for Training data Collection

```
#include <boost/bind.hpp>
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/common/common.hh>
#include <gazebo/common/Plugin.hh>
#include "ros/ros.h"
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include <ignition/math/Vector3.hh>
#include <ignition/math/Pose3.hh>
#include <stdio.h>
#include <gazebo/transport/transport.hh>
#include <gazebo/msgs/msgs.hh>
#include "std_msgs/Float32.h"
#include "std_msgs/String.h"
#include <thread>
#include <delta3_lib/Angles.h>
#include <cstdlib>
#include <math.h>
namespace gazebo
{
class Delta3ModelPlugin : public ModelPlugin
{
public:
```

```
 void Load(physics::ModelPtr _parent, sdf::ElementPtr _sdf)
 {
  this->model = _parent;
  this->jointController = this->model->GetJointController();
  if (!ros::isInitialized())
  {
   int argc = 0;
   char **argv = NULL;
   ros::init(argc, argv, "gazebo_client", ros::init_options::NoSigintHandler);
   ROS_FATAL_STREAM("Ros is not initialized."
                    << "Load the .. in gazebo_ros");
  }
  else
  {
   ROS_INFO("Starting plugin");
  }
  this->pid = common::PID(5, 1, 0.5);
  std::cout << "\n\n"
            << this->model->GetName() << "\n\n";
  this->rosNode.reset(new ros::NodeHandle("gazebo_client"));
  ros::SubscribeOptions so = ros::SubscribeOptions::create<delta3_lib::Angles>(
     "/" + this->model->GetName() + "/pos_cmd",1,
     boost::bind(&Delta3ModelPlugin::OnRosMsg, this, _1),
     ros::VoidPtr(), &this->rosQueue);
  this->data_pub = this->rosNode->advertise<std_msgs::String>("/data", 1000);
  this->rosSub = this->rosNode->subscribe(so);
  this->rosQueueThread = std::thread(std::bind(&Delta3ModelPlugin::QueueThread, this));
```

```
  this->rosDataPublishThread = std::thread(std::bind(&Delta3ModelPlugin::Publish, this));
  this->updateConnection = event::Events::ConnectWorldUpdateBegin(
    std::bind(&Delta3ModelPlugin::OnUpdate, this));
  ROS_INFO("Finished setting up");
 }
private:
 void Publish()
 {
  ros::Rate loop_rate(10);
  while (ros::ok())
  {
   if (this->update_num == -1)
   {
    std_msgs::String msg;
    std::stringstream ss;
    physics::LinkPtr gripper = this->model->GetLink("lower_triangle_holder");
    physics::LinkState state = physics::LinkState(gripper);
    ignition::math::Vector3d pos = state.Pose().Pos();
    double degrees[3] = {0, 0, 0};
    this->GetJointPoses(degrees);
    // std::cout << "Actual Arm Joint Pose: " << degrees[0] << " " << degrees[1] << " " <<
degrees[2] << " \n";
    // std::cout << "Expected Arm Joint Pose: " << angle[0] << " " << angle[1] << " " <<
angle[2] << "\n";
    // std::cout << "Gripper Pose: " << pos.X() << " " << pos.Y() << " " << pos.Z() << "\n\n";
    physics::ModelState modelState = physics::ModelState(this->model);
    ignition::math::Vector3d modelPose = modelState.Pose().Pos();
```

```
      ss << this->choose << " ";
      ss << modelPose.X() << " " << modelPose.Y() << " " << modelPose.Z() << " ";
      // ss << angle[0] << " " << angle[1] << " " << angle[2] << " ";
      // ss << degrees[0] << " " << degrees[1] << " " << degrees[2] << " ";
      // ss << pos.X() << " " << pos.Y() << " " << pos.Z();
      msg.data = ss.str();
      ROS_INFO("%s", msg.data.c_str());
      this->data_pub.publish(msg);
      GenerateAngle();
      this->GenerateChoose();
      update_num = 0;
    }
    ros::spinOnce();
    loop_rate.sleep();
  }
 }
public:
 float Random(float a, float b)
 {
  float random = ((float)rand()) / (float)RAND_MAX;
  float diff = b - a;
  float r = random * diff;
  return a + r;
 }
public:
 void OnUpdate()
 {
```

```
  if (update_num == 0)
  {
   if (choose != 1)
   {
    this->SetAngle("upper_arm_a1_s1", angle[0]);
   }
   if (choose != 2)
   {
    this->SetAngle("upper_arm_a2_s2", angle[1]);
   }
   if (choose != 3)
   {
    this->SetAngle("upper_arm_a3_s3", angle[2]);
   }
  }
  else if (update_num < 4000)
  {
   this->jointController->Update();
  }
  else if (update_num == 4000)
  {
   this->model->GetJoint("upper_arm_a1_s1")->SetParam("fmax", 0, 0);
   this->model->GetJoint("upper_arm_a2_s2")->SetParam("fmax", 0, 0);
   this->model->GetJoint("upper_arm_a3_s3")->SetParam("fmax", 0, 0);
   update_num = -1;
  }
  if (update_num >= 0)
```

```
  {
    update_num++;
  }
 }
private:
 void GenerateChoose()
 {
  switch (choose)
  {
  case 1:
    choose = 2;
    break;
  case 2:
    choose = 3;
    break;
  case 3:
    choose = 1;
    break;
  }
 }
private:
 void GenerateAngle()
 {
  angle[0] = this->Random(-60, 60);
  angle[1] = this->Random(-60, 60);
  angle[2] = this->Random(-60, 60);
 }
```

```
private:
 void GetJointPoses(double degrees[])
 {
  double p1 = physics::JointState(this->model->GetJoint("upper_arm_a1_s1")).Position(0);
  double p2 = physics::JointState(this->model->GetJoint("upper_arm_a2_s2")).Position(0);
  double p3 = physics::JointState(this->model->GetJoint("upper_arm_a3_s3")).Position(0);
  degrees[0] = p1 * 180 / M_PI;
  degrees[1] = p2 * 180 / M_PI;
  degrees[2] = p3 * 180 / M_PI;
 }


private:
 void SetAngle(std::string joint_name, float degree)
 {
  if (degree >= -60 && degree <= 60)
  {
   // std::cout << joint_name << std::endl;
   float rad = M_PI * degree / 180;
   std::string name = this->model->GetJoint(joint_name)->GetScopedName();
   this->jointController->SetPositionPID(name, pid);
   this->jointController->SetPositionTarget(name, rad);
  }
 }
public:
 void OnRosMsg(const delta3_lib::Angles::ConstPtr &msg)
 {
  angle[0] = msg->upper_arm1;
```

```
    angle[1] = msg->upper_arm2;
    angle[2] = msg->upper_arm3;
    update_num = 0;
  }
private:
  void SetJointAngle(const std::string &name, float degree)
  {
    float angle = (M_PI * degree / 180);
    this->jointController->SetPositionPID(name, this->pid);
    this->jointController->SetPositionTarget(name, angle);
    this->jointController->Update();
  }
public:
  void QueueThread()
  {
    static const double timeout = 0.01;
    while (this->rosNode->ok())
    {
      this->rosQueue.callAvailable(ros::WallDuration(timeout));
    }
  }
private:
  float angle[3] = {0, 0, 0};
private:
  float gripper_pos[3] = {0, 0, 0};
private:
  int update_num = 0;
```

```
private:
  int choose = 1;
private:
  physics::ModelPtr model;
private:
  physics::JointControllerPtr jointController;
private:
  std::unique_ptr<ros::NodeHandle> rosNode;
private:
  ros::Subscriber rosSub;
private:
  ros::Subscriber linkStateSub;
private:
  std::thread rosQueueThread, rosDataPublishThread;
private:
  ros ::CallbackQueue rosQueue;
private:
  ros::Publisher data_pub;
private:
  event::ConnectionPtr updateConnection;
private:
  common::PID pid;
};
GZ_REGISTER_MODEL_PLUGIN(Delta3ModelPlugin);
} // namespace gazebo
```

Appendix C: Tabular Representation of Collected Training Data Set

Θ1	Θ2	Θ3	X	Y	Z
3.32E-01	1.41E-01	1.39E-01	-1.17E+00	7.14E-01	1.95E+00
6.27E-01	8.72E-01	8.71E-01	-1.61E+00	6.07E-01	2.00E+00
6.28E-01	4.99E-01	-6.22E-01	7.30E-01	9.07E-01	1.87E+00
6.24E-01	-6.13E-01	-6.11E-01	7.22E-01	9.10E-01	1.87E+00
5.49E-01	-5.49E-01	-5.47E-01	8.67E-03	1.08E+00	2.07E+00
5.51E-01	-5.42E-01	3.87E-01	-7.92E-01	7.75E-01	1.73E+00
6.33E-01	3.81E-01	3.80E-01	-7.85E-01	7.74E-01	1.74E+00
6.43E-01	1.01E+00	1.01E+00	-2.63E-01	-4.85E-01	9.18E-01
5.60E-01	1.01E+00	9.51E-01	-4.41E-01	-1.57E+00	1.78E+00
-5.61E-01	9.97E-01	9.97E-01	-4.42E-01	-1.57E+00	1.78E+00
-5.52E-01	-3.48E-01	-3.45E-01	8.59E-01	-6.48E-01	1.99E+00
-5.12E-01	-3.36E-01	-3.34E-01	-5.00E-01	-3.22E-01	2.05E+00
-5.30E-01	2.70E-01	2.70E-01	-7.02E-01	-3.22E-01	2.09E+00
-5.29E-01	4.51E-01	1.55E-01	5.11E-01	2.22E-02	1.36E+00
4.07E-01	1.57E-01	1.57E-01	5.08E-01	1.94E-02	1.36E+00
7.68E-01	-5.88E-01	-5.87E-01	1.11E+00	8.89E-01	1.79E+00
7.68E-01	-5.86E-01	-8.28E-01	6.21E-01	1.21E+00	2.08E+00
7.07E-01	-8.78E-01	-8.78E-01	6.21E-01	1.21E+00	2.08E+00
6.41E-01	-8.75E-01	-8.75E-01	1.13E+00	4.64E-01	2.00E+00
3.05E-01	-8.76E-01	-8.77E-01	6.22E-02	-9.05E-01	1.44E+00
-8.85E-01	9.06E-01	9.05E-01	-4.48E-01	-1.64E+00	2.02E+00
-8.80E-01	5.37E-01	4.46E-01	-2.83E-01	-1.06E+00	1.99E+00
-8.21E-01	4.43E-01	4.42E-01	-2.79E-01	-1.06E+00	1.99E+00
8.64E-01	3.96E-01	3.96E-01	-2.94E-01	6.92E-01	1.36E+00
7.76E-01	3.96E-01	3.78E-01	-3.08E-01	6.68E-02	1.49E+00
2.97E-01	3.89E-01	3.90E-01	-3.16E-01	6.32E-02	1.49E+00
3.56E-01	1.01E+00	1.01E+00	-9.36E-01	-3.76E-01	1.31E+00
3.51E-01	1.01E+00	1.01E+00	-1.48E+00	5.51E-01	1.60E+00
1.09E+00	1.02E+00	1.02E+00	-1.62E+00	1.05E+00	1.81E+00
1.09E+00	-9.71E-01	-9.90E-01	2.76E-01	-1.94E-02	2.26E+00
-6.97E-01	-9.89E-01	-9.89E-01	2.76E-01	-1.78E-02	2.26E+00
-5.36E-01	9.62E-01	9.60E-01	-1.60E+00	-6.13E-01	2.31E+00
-5.38E-01	9.58E-01	9.96E-01	-1.68E+00	-2.40E-01	2.16E+00
-5.90E-02	9.78E-01	9.77E-01	-1.68E+00	-2.37E-01	2.16E+00
1.20E+00	-1.93E-01	-1.91E-01	-5.03E-01	1.78E+00	2.23E+00
1.10E+00	-2.32E-02	1.46E-01	-7.90E-01	1.59E+00	2.09E+00
1.06E+00	1.52E-01	1.53E-01	-7.97E-01	1.58E+00	2.09E+00

3.56E-01	2.77E-01	2.76E-01	5.54E-01	-6.59E-01	1.49E+00
-9.65E-03	2.72E-01	5.11E-01	1.59E-01	-1.51E+00	1.95E+00
-8.20E-01	5.66E-01	5.66E-01	1.58E-01	-1.51E+00	1.95E+00
-8.43E-01	-2.10E-01	-2.09E-01	1.10E+00	-1.18E+00	2.16E+00
-8.41E-01	-2.07E-01	-2.07E-01	1.37E+00	1.02E-01	1.33E+00
9.24E-01	-1.20E-01	-1.20E-01	1.09E+00	6.93E-01	1.38E+00
9.15E-01	4.01E-01	7.57E-01	1.18E-01	-1.18E+00	1.42E+00
-1.33E-01	7.49E-01	7.47E-01	-1.07E+00	-6.29E-01	1.72E+00
-8.81E-02	1.04E+00	1.04E+00	-1.35E+00	-6.33E-01	1.81E+00
-8.49E-02	1.04E+00	1.04E+00	-1.35E+00	-6.01E-01	1.80E+00
-5.36E-02	1.04E+00	1.04E+00	-1.36E+00	-5.97E-01	1.79E+00
9.09E-02	-2.63E-01	-2.61E-01	-3.89E-02	3.69E-01	1.92E+00
9.49E-02	-2.51E-01	-1.87E-01	5.85E-01	4.31E-02	1.67E+00
1.57E-01	-1.89E-01	-1.89E-01	5.86E-01	4.11E-02	1.67E+00
5.02E-01	-2.01E-01	-2.01E-01	-2.04E-01	1.24E+00	1.93E+00
7.73E-01	-2.00E-01	-2.00E-01	-4.64E-01	1.24E+00	1.80E+00
8.84E-01	2.44E-01	2.46E-01	-6.03E-01	1.18E+00	1.75E+00
8.86E-01	3.01E-01	4.45E-01	-1.11E+00	1.20E+00	1.99E+00
8.84E-01	4.50E-01	4.51E-01	-1.11E+00	1.20E+00	1.99E+00
1.03E+00	5.53E-01	5.53E-01	-6.25E-01	1.05E+00	1.40E+00
1.03E+00	-6.10E-01	-6.16E-01	4.25E-01	1.46E+00	2.01E+00
9.36E-01	-6.09E-01	-6.07E-01	3.67E-01	1.46E+00	1.99E+00
9.79E-01	6.45E-01	6.43E-01	-1.35E+00	1.23E+00	2.03E+00
9.75E-01	6.14E-01	4.67E-01	-1.04E+00	-3.80E-01	2.21E+00
-6.21E-01	4.68E-01	4.68E-01	-1.04E+00	-3.77E-01	2.21E+00
5.39E-01	8.65E-01	8.66E-01	-1.60E+00	4.93E-01	2.00E+00
5.36E-01	8.66E-01	8.66E-01	1.06E-01	6.45E-01	1.68E+00
4.90E-01	-1.24E-01	-1.22E-01	1.12E-01	6.34E-01	1.69E+00
-4.74E-01	-1.05E-01	-1.04E-01	9.41E-01	-9.03E-01	1.91E+00
-4.71E-01	1.56E-01	9.18E-01	-4.75E-01	5.59E-01	8.91E-01
1.11E+00	9.25E-01	9.27E-01	-4.85E-01	5.47E-01	8.88E-01
1.06E+00	8.71E-01	8.70E-01	-7.77E-01	7.23E-01	1.13E+00
1.05E+00	6.84E-01	8.52E-01	-1.24E+00	-2.47E-01	1.78E+00
8.26E-02	8.50E-01	8.49E-01	-1.23E+00	-2.44E-01	1.78E+00
1.04E-01	3.36E-01	3.36E-01	-6.67E-01	1.24E-01	1.80E+00
1.08E-01	3.36E-01	4.25E-01	9.97E-01	-5.49E-01	1.33E+00
2.81E-01	2.58E-01	2.57E-01	9.61E-01	-5.25E-01	1.33E+00
-2.59E-01	2.06E-01	2.06E-01	1.05E-01	-5.66E-01	1.71E+00
-2.59E-01	2.08E-01	2.09E-01	-2.08E-01	-7.44E-01	1.61E+00
-2.10E-01	5.82E-01	5.83E-01	-2.76E-01	-7.81E-01	1.60E+00
-2.06E-01	2.44E-01	5.26E-02	1.18E+00	-9.32E-01	1.73E+00

-1.82E-01	5.14E-02	5.11E-02	4.86E-01	-4.51E-01	1.76E+00
-7.93E-02	-8.45E-03	-6.50E-03	-1.73E-01	2.54E-02	1.85E+00
-7.62E-02	4.28E-01	8.18E-01	-9.62E-01	6.69E-01	1.34E+00
8.95E-01	8.18E-01	8.19E-01	-9.43E-01	6.65E-01	1.34E+00
8.93E-01	8.12E-01	8.12E-01	-9.42E-01	6.61E-01	1.34E+00
8.91E-01	7.29E-01	7.32E-01	-9.31E-01	4.76E-03	1.54E+00
3.28E-01	7.32E-01	7.32E-01	-9.31E-01	4.08E-03	1.54E+00
-6.87E-01	4.11E-01	4.11E-01	-5.36E-01	-7.41E-01	2.01E+00
-6.86E-01	4.07E-01	4.06E-01	8.12E-01	-6.35E-01	2.09E+00
-6.84E-01	-4.76E-01	-4.76E-01	8.11E-01	-6.33E-01	2.09E+00
-6.83E-01	-5.15E-01	-5.13E-01	6.01E-02	-1.94E-01	2.20E+00
-7.62E-01	-5.09E-01	-5.08E-01	-7.95E-01	-8.10E-01	2.10E+00
-7.93E-01	5.47E-01	5.45E-01	-7.83E-01	-8.07E-01	2.10E+00
-7.99E-01	-8.66E-01	-8.66E-01	9.82E-01	-5.48E-01	2.24E+00
-8.36E-01	-8.65E-01	-8.65E-01	9.81E-01	-5.49E-01	2.25E+00
7.51E-01	-7.51E-01	-7.50E-01	-1.42E-02	1.32E+00	2.23E+00
6.60E-01	-7.49E-01	-8.19E-01	3.02E-02	1.29E+00	2.24E+00
6.27E-01	-8.20E-01	-8.20E-01	3.05E-02	1.28E+00	2.24E+00
3.38E-01	1.17E+00	1.17E+00	-8.75E-01	-6.36E-01	1.18E+00
3.61E-01	1.16E+00	1.13E+00	-1.24E+00	-1.35E-01	1.34E+00
5.33E-01	1.12E+00	1.12E+00	-1.23E+00	-1.27E-01	1.34E+00
-3.94E-01	-4.31E-01	-4.29E-01	4.30E-01	-2.18E-01	2.00E+00
-3.92E-01	-4.24E-01	-4.23E-01	-3.56E-01	-1.92E-01	2.00E+00
-3.87E-01	4.49E-02	4.51E-02	-3.56E-01	-1.93E-01	2.00E+00
-3.86E-01	1.19E-01	1.15E-01	1.98E-02	7.06E-01	1.49E+00
6.74E-01	1.15E-01	1.14E-01	2.08E-02	7.01E-01	1.49E+00
3.01E-02	-6.91E-01	-6.91E-01	6.49E-01	2.70E-01	1.98E+00
2.76E-02	-6.91E-01	-6.67E-01	6.54E-01	2.51E-01	1.97E+00
2.92E-02	-6.58E-01	-6.56E-01	6.48E-01	2.47E-01	1.96E+00
1.91E-01	-6.16E-01	-6.14E-01	8.73E-01	2.92E-01	1.88E+00
1.96E-01	-6.07E-01	-6.06E-01	1.32E-01	-6.54E-01	1.60E+00
-4.23E-01	5.71E-01	5.69E-01	-1.21E-01	-1.05E+00	1.71E+00
-5.37E-01	-8.39E-02	-8.41E-02	-9.98E-02	-2.76E-01	1.99E+00
-4.59E-01	-8.50E-02	-8.52E-02	-1.13E-01	-2.89E-01	2.01E+00
-4.99E-01	-9.37E-02	-9.44E-02	-1.10E-01	-2.90E-01	2.01E+00
-5.02E-01	4.37E-02	-1.23E-02	-1.87E-01	1.29E+00	1.75E+00
9.25E-01	-1.51E-02	-1.59E-02	-1.83E-01	1.28E+00	1.75E+00
8.51E-01	5.55E-01	5.56E-01	-7.58E-01	8.45E-01	1.49E+00
8.48E-01	5.58E-01	6.50E-01	5.56E-01	1.58E-01	7.92E-01
9.96E-01	6.43E-01	6.42E-01	5.99E-01	8.53E-02	1.01E+00
-6.18E-01	-4.19E-01	-4.20E-01	1.50E+00	-1.04E+00	2.20E+00

-6.15E-01	9.92E-01	6.76E-01	-1.29E+00	-3.22E-01	2.21E+00
-4.24E-01	6.54E-01	6.54E-01	-1.30E+00	-2.32E-01	1.99E+00
4.71E-01	5.86E-01	5.86E-01	-9.55E-01	4.02E-01	1.67E+00
2.24E-01	-6.62E-01	-6.62E-01	2.26E-01	1.29E-01	2.12E+00
-3.14E-01	-6.60E-01	-6.60E-01	2.47E-01	9.83E-02	2.07E+00
-3.20E-01	5.50E-02	5.33E-02	1.12E+00	-1.03E+00	1.81E+00
-3.22E-01	4.70E-02	-3.35E-02	-3.20E-01	-1.66E-02	1.98E+00
-2.46E-01	-2.75E-02	-2.59E-02	-3.26E-01	-1.86E-02	1.98E+00
6.72E-01	-6.44E-01	-6.43E-01	1.99E-01	1.21E+00	2.06E+00
6.69E-01	-6.41E-01	-6.40E-01	5.85E-01	1.16E+00	2.08E+00
6.58E-01	-8.78E-01	-8.78E-01	5.86E-01	1.15E+00	2.08E+00
6.52E-01	-1.00E+00	-1.00E+00	2.52E-01	-3.11E-02	2.27E+00
-7.46E-01	-1.00E+00	-1.00E+00	2.52E-01	-2.99E-02	2.27E+00
2.89E-01	-3.27E-01	-3.27E-01	-8.13E-02	6.61E-01	1.96E+00
2.86E-01	-3.26E-01	8.39E-02	-8.04E-01	5.81E-01	2.02E+00
2.78E-01	1.70E-01	1.70E-01	-8.03E-01	5.78E-01	2.02E+00
9.52E-03	2.14E-01	2.13E-01	6.64E-01	-6.21E-01	1.50E+00
9.28E-03	2.09E-01	-2.28E-01	1.38E+00	2.50E-01	2.03E+00
2.35E-01	-8.80E-01	-8.79E-01	1.38E+00	2.50E-01	2.02E+00
2.53E-01	-1.01E-01	-1.01E-01	2.25E-01	2.67E-01	1.67E+00
2.51E-01	-2.14E-01	-9.91E-02	-4.76E-01	-1.12E-01	2.20E+00
-6.13E-01	-1.01E-01	-1.02E-01	-4.73E-01	-1.13E-01	2.20E+00
-6.92E-01	8.98E-01	8.95E-01	-1.45E+00	-7.22E-01	2.30E+00
-6.99E-01	8.87E-01	2.96E-01	-7.83E-01	-4.12E-01	2.19E+00
-7.33E-01	2.97E-01	2.98E-01	-7.85E-01	-4.15E-01	2.19E+00

Appendix D: Source Code for Training of ANNC for Delta 3 Robot

```
from tensorflow.keras.models import load_model,Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Input, LSTM, Reshape, BatchNormalization
from tensorflow.keras.layers import MaxPooling2D, GRU
from tensorflow.keras.layers import Flatten, ReLU
from tensorflow.keras.layers import Dense,Dropout, Concatenate
from tensorflow.keras.preprocessing import image
from tensorflow.keras import regularizers
import tensorflow.keras as keras
import matplotlib.pyplot as plt
import numpy as np
def data_gen(X, Y, batch_size=100):
    n_batches = X.shape[0] // batch_size
    current = 0
    while True:
        batch_x = X[current:current + batch_size]
        y = Y[current:current + batch_size]
        batch_y = [y[:, 0], y[:, 1], y[:, 2]]
        current += 1
        if current >= n_batches:
            current = 0
        yield batch_x, batch_y
inputs = Input(shape=(3,), name="position")
x = Dense(20, name="dense1",  kernel_initializer='normal', activation="relu")(inputs)
x = Dense(20, name="dense2", activation="relu")(x)
```

```
x = Dense(20, name="dense3", activation="relu")(x)
ax = Dense(1, kernel_initializer='normal', name="theta1")(x)
bx = Dense(1,kernel_initializer='normal', name="theta2")(x)
tx = Dense(1, kernel_initializer='normal',name="theta3")(x)
model = Model(inputs=inputs, outputs=[ax, bx, tx])
optm = keras.optimizers.Nadam(lr=0.001)
model.compile(optimizer=optm, loss="mse")
model.summary()
data = np.loadtxt("data.csv", delimiter=",")
train_size = int(.7 * data.shape[0])
train = data[:train_size]
test = data[train_size:]
vX = test[:, 3:]
vY = test[:, :3]
X = train[:, 3:]
Y = train[:, :3]
print(X.shape)
batch_size = 100
val_steps = vX.shape[0]//batch_size
n_batches = X.shape[0] // batch_size
gen= data_gen(X, Y, batch_size=batch_size)
val_gen = data_gen(vX, vY, batch_size=batch_size)
history = model.fit_generator(gen, epochs=20, steps_per_epoch=n_batches,
validation_data=val_gen, validation_steps=val_steps)
train_loss = history.history['loss']
valid_loss = history.history['val_loss']
x = range(len(train_loss))
```

```
plt.plot(x, train_loss)
plt.plot(x, valid_loss)
plt.show()
```