A QP Artificial Neural Network Inverse Kinematic Solution for Accurate Robot Path Control

Şahin Yıldırım*, İkbal Eski

Erciyes University, Faculty of Engineering, Mechanical Engineering Department, Kayseri, 38039, Turkey

In recent decades, Artificial Neural Networks (ANNs) have become the focus of considerable attention in many disciplines, including robot control, where they can be used to solve nonlinear control problems. One of these ANNs applications is that of the inverse kinematic problem, which is important in robot path planning. In this paper, a neural network is employed to analyse of inverse kinematics of PUMA 560 type robot. The neural network is designed to find exact kinematics of the robot. The neural network is a feedforward neural network (FNN). The FNN is trained with different types of learning algorithm for designing exact inverse model of the robot. The Unimation PUMA 560 is a robot with six degrees of freedom and rotational joints. Inverse neural network model of the robot is trained with different learning algorithms for finding exact model of the robot. From the simulation results, the proposed neural network has superior performance for modelling complex robot's kinematics.

Key Words: Artificial Neural Network, Robot Path Control, PUMA Robot

Nomenclature

a_i : The distance from Z_i to Z_{i+1} measured along X_i

α_i : The angle between Z_i to Z_{i+1} measured about X_i

d_i : The distance from X_{i-1} to X_i measured along Z_i

θ_i : The angle between X_{i-1} to X_i measured about X_i

p_x, p_y, p_z : End-effector coordinate of x, y, z frame

${}^0_6T$: Transform matrices from have to joint 6

c_i : Cos θ_i

s_i : Sin θ_i

c_{23} : $c_2c_3 - s_2s_3$

s_{23} : $c_2s_3 + s_2c_3$

c_{12} : $c_1c_2 - s_1s_2$

s_{12} : $c_1s_2 + s_1c_2$

s_ϕ : s_{123}

c_ϕ : c_{123}

η : Learning rate

α : Momentum term

$\Delta w_{ij}(t)$: The weight matrices from i.layer to j. layer variations after update

$E(t)$: Error variations for weights

$\delta(t)$: Derivation of error in weights

$\bar{\delta}(t)$: Exponential average of past value of δ

K : Constant value for learning rate

φ : Correction factor of learning rate

θ : Coefficient of the exponential average value

μ : Maximum growth factor

W_{ij} : Weight matrices from i.layer to j.layer

N : Training numbers

n_I : Number of neurons in the input layer

n_H : Number of neurons in the hidden layer

n_O : Number of neurons in the output layer

$z_j(t)$: The output vector of the hidden layer

AF : Activation function

* Corresponding Author,
E-mail : sahiny@erciyes.edu.tr
Erciyes University, Faculty of Engineering, Mechanical Engineering Department, Kayseri, 38039, Turkey.
(Manuscript **Received** February 18, 2005; **Revised** May 8, 2006)

1. Introduction

The ability of ANNs to approximate highly non-linear functions has been broadly applied in diverse applications such as pattern recognition, robot control and image processing. In particular, the use of ANNs in control applications has recently experienced rapid growth. Various control strategies have been suggested using ANNs technology (Tchon, 2000), which learn an approximation of systems's characteristics and then use the ANNs to generate the appropriate control signal. In practical robot control applications, ANNs can provide tools for system identification, including forward and inverse plant identification. Although ANNs applicable to the solution of robot control problems are, in fact, neurocontrollers, their function is specialized mainly to provide solutions to robot arm kinematics problems.

In the practical use of manipulators, the two fundamental operations that relate to the locations of the end-effector are forward and inverse kinematics. Specifically, given a set of joint angles, the forward kinematics problem is to compute the position and orientation of the end-effector, is to calculate all possible sets of joint angles which could be used to attain this given position and orientation. Solving the forward kinematics problem is straightforward. Inverse kinematics is not as simple as forward kinematics as they are a multi-solution problem. Because of its highly non-linear characteristics, the solution of inverse kinematics is not always easy or even possible in a closed form (Aspragathos, 1998 ; Pashkevich, 1997).

Since 1979, many research papers have been published to solve the kinematics problem of robot manipulators using ANNs, which is basically a function approximation problem (Bravo, 1990 ; Guez and Ahmad, 1989). Most of papers have applied Back Propagation (BP) neural networks to approximate the complicated inverse kinematics solution.

A three-fingered anthropomorphic robot hand, called SKK Robot Hand I, has been investigated by Kang et al.(2003). Their proposed hand was developed for the use as a testbed for dextrous manipulation. It was expected to resolve the increasing demand for robotic applications in unstructured environments.

A 3-PPR planar parallel manipulator, which consists of three active prismatic joints, three passive prismatic joints, and three passive rotational joints, has been proposed (Choi, 2003). The analysis of the kinematics and the optimal design of the manipulator have also been discussed and one example using the optimal design was presented.

A new method of estimating the pose of a mobile-task robot has been developed based upon an active calibration scheme. The utility of a mobile-task robot has been widely recognized, which is formed by the serial connection of a mobile robot and a task robot. Their proposed active calibration scheme was verified experimentally (Jin and Lee, 2003).

The control scheme using fuzzy modelling and Parallel Distributed Compensation (PDC) concept has been proposed to provide asymptotic tracking of a reference signal for the flexible joint manipulators with the uncertain parameters (Lee et al., 2004).

A robust position control with the bound function of neural network structure has been investigated for uncertain robot manipulators. The uncertain factors come from imperfect knowledge of system parameters, payload change, friction, external disturbance, and etc. Therefore, uncertainties were often nonlinear and time varying. Simulation was performed to validate this law for four-axis SCARA type robot manipulator (Ha and Han, 2004).

This paper proposes a neural network strategy, namely the neural network modelling method, to avoid the difficulties of local minima and other residual errors to improve the accuracy of the inverse solution using ANNs technology. The method iteratively employs the easily obtained analytical inverse solution and an ANNs with different learning algorithm to calculate the complex inverse solution.

The following sections describe the process for iteratively solving the inverse kinematic problem

for practical robot manipulators, using algebraic method. Section 2 introduces the inverse kinematic solution method of PUMA robot. Section 3 is described neural network theory and learning algorithms. The results of this simulation are used to demonstrate the feasibility of using neural network modelling method of PUMA robot (see Section 3). Finally, a discussion and conclusions are presented in the last section.

2. Inverse Kinematic Solution of Puma Robot Manipulator

Kinematics is the science of motion, which treats motion without regard the position, velocity, acceleration and all higher derivates of the position variables. Hence, the study of the kinematics of manipulators refers to all the geometrical and time-based properties of the motion. The relationships between these motions and the forces and torques, which cause them, are the problem of dynamics. The study of manipulator kinematics involves, among other things, how the locations of these frames change as the mechanism articulates.

In this section, the kinematics of PUMA 560 robot is worked out. The PUMA 560 is a robot manipulator with six degrees of freedom (dof) and all rotational joints. It is shown in Fig. 1 with link frame assignments in the position corresponding all joint angles equal to zero. The link parameters corresponding to this placement of the link frames are shown in Table 1. In the case of the PUMA 560 a gearing arrangement in the wrist of the manipulator couples together the motions of joints 4, 5 and 6.

The location of the PUMA robot can be described by three variables; (x, y, z) represent the position. Hence a network with 3 inputs. The PUMA 560 manipulator has a special physical structure for its kinematics that the first there joints determine the P position of the robot,

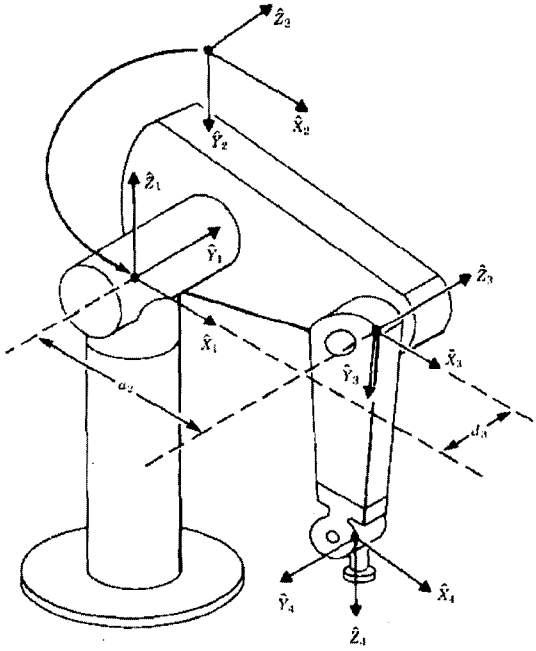

Fig. 1 Kinematic parameters and frame assignments for the PUMA 560 manipulator

Table 1 Desired kinematics parameters of the robot

P_X [mm]	P_Y [mm]	P_Z [mm]	θ_1 [degree]	θ_2 [degree]	θ_3 [degree]	θ_4 [degree]	θ_5 [degree]	θ_6 [degree]
86.35	829.34	671.94	73.76	−19.79	129.71	0.0	70.08	73.76
204.80	686.19	695.62	61.40	−39.23	164.16	0.0	55.06	61.40
314.52	563.52	709.55	47.47	−48.90	179.18	0.0	49.22	47.47
411.64	463.01	712.97	34.44	−51.42	184.12	0.0	47.30	34.44
524.27	350.38	712.97	20.08	-50.26	182.01	0.0	48.25	20.08
632.42	249.07	706.13	8.83	−44.34	17.21	0.0	52.13	8.83
708.15	190.43	689.03	3.32	−36.69	160.35	0.0	56.34	3.32
793.50	125.60	668.52	−1.70	−25.25	141.00	0.0	64.25	−1.70
834.12	197.83	555.67	3.33	−4.21	114.36	0.0	69.05	3.33

measured from the origin of the shoulder coordinate system to the point where the last three joint axes intersect, and the orientation of the end effector is determined by the last three joints. There for it is possible to solve the problem caused by limitation of the software by one step. The step has an ANNs to obtain a set of joint angles ; only step was used to solve for the six joints angles and rotational angles. The inverse neural network model of the robot kinematics is shown in Fig. 2. The inputs of the network include the position $P=(x,y,z)$. The outputs are joints angles for position and rotation $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$.

As an example of the algebraic solution technique applied to a manipulator with six degrees of freedom, they will solve the kinematics equations of the PUMA 560. This solution is in the style of (Craig, 1989).

$$ {}^0_6T=\begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_Y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

$${}^0_6T={}^0_1T(\theta_1)\cdot{}^1_2T(\theta_2)\cdot{}^2_3T(\theta_3)\cdot{}^3_4T(\theta_4)\cdot{}^4_5T(\theta_5)\cdot{}^5_6T(\theta_6)$$

for θ_i when ${}^0_6T$ is given as numeric values. A restatement of equation (1) which puts the dependence on θ_1 on the left-hand side of the equation can be written in the following form ;

$$ \begin{aligned} &[{}^0_6T(\theta_1)]^{-1}\cdot{}^0_6T \\ &={}^1_2T(\theta_2)\cdot{}^2_3T(\theta_3)\cdot{}^3_4T(\theta_4)\cdot{}^4_5T(\theta_5)\cdot{}^5_6T(\theta_6) \end{aligned} \tag{2} $$

Inverting ${}^0_1T$ above general equation can be written matrix form as follows,

$$ \begin{bmatrix} c_1 & s_1 & 0 & 0 \\ -s_1 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_Y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^1_6T \tag{3} $$

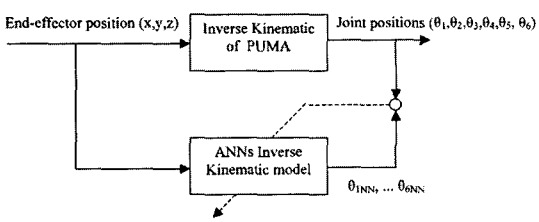

Fig. 2 An ANNs inverse kinematic modeling of PUMA

where ${}^1r_{32}$ is,

$$ {}^1r_{32}=s_{23}[c_4c_5s_6+s_4c_6]+c_{23}s_5s_6 \tag{4} $$

This simple technique of multiplying each side of a transform equation by an inverse is often used to advantage in separating out variables in search of a solvable equation. Equating the 2, 4 elements from both sides of equation (3),

$$ -s_1p_x+c_1p_y=d_3 \tag{5} $$

To solve an equation of this form, they make the trigonometric substitutions,

$$ \begin{aligned} p_x&=\rho\cos\phi \\ p_y&=\rho\sin\phi \end{aligned} \tag{6} $$

where,

$$ \rho=\sqrt{p_x^2+p_y^2},\ \phi=A\tan 2(p_y,\ p_x) $$

Substituting equation (6) into equation (5),

$$ c_1s_\phi-s_1c_\phi=\frac{d_3}{\rho} \tag{7} $$

Using the difference of angles formula,

$$ \sin(\phi-\theta_1)=\frac{d_3}{\rho} \tag{8} $$

Hence,

$$ \cos(\phi-\theta_1)=\pm\sqrt{1-\frac{d_3^2}{\rho^2}} \tag{9} $$

$$ \phi-\theta_1=A\tan 2\left(\frac{d_3}{\rho}\pm\sqrt{1-\frac{d_3^2}{\rho^2}}\right) \tag{10} $$

Finally, the solution for first joint rotation θ_1 may be written,

$$ \begin{aligned} \theta_1&=A\tan 2(p_x,\ p_y) \\ &-A\tan 2(d_3,\ \pm\sqrt{p_x^2+p_y^2+d_3^2}) \end{aligned} \tag{11} $$

Note that they have found two possible solutions for θ_1 corresponding to the plus-or-minus sign in equation (11). Now that θ_1 is known, the left-hand side of equation (3) is known. If they equate the 1, 4 elements from both sides of equation (3) and also the 3, 4 elements,

$$ \begin{aligned} &c_1p_x+s_1p_y=a_3c_{23}-d_4s_{23}+a_2c_2, \\ &-p_z=a_3s_{23}+d_4c_{23}+a_2s_2 \end{aligned} \tag{12} $$

If they square equations (12) and (5) and add the resulting equations,

$$a_3c_3-d_4s_3=K \tag{13}$$

$$K=\frac{p_x^2+p_y^2+p_z^2-a_2^2-a_3^2+d_3^2-d_4^2}{2a_2} \tag{14}$$

Note that dependence on θ_1 has been removed from (13). Equation (13) is of the same form as equation (5) and so may be solved by the same kind of trigonometric substitution to yield a solution for third joint rotation θ_3:

$$\theta_3=A\tan 2(a_3,\ d_4) - A\tan 2(K,\ \pm\sqrt{a_3^2+d_4^2-K^2}) \tag{15}$$

The plus-or-minus sign in equation (15) leads to two different solutions for third joint rotation θ_3. If they consider equation (1) again, they can now rewrite it so that all the left-hand side is a function of only known and θ_2:

$$[{}^0_3T(\theta_2)]^{-1}\cdot{}^0_6T={}^3_4T(\theta_4)\cdot{}^4_5T(\theta_5)\cdot{}^5_6T(\theta_6) \tag{16}$$

or,

$$\begin{bmatrix} c_1c_{23} & s_1c_{23} & -s_{23} & -a_2c_3 \\ -c_1s_{23} & -s_1s_{23} & -c_{23} & a_2s_3 \\ -s_1 & c_1 & 0 & -d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}={}^3_6T \tag{17}$$

Equating the 1, 4 elements from both sides of equation (17), as well as 2, 4 elements,

$${}^3_6T={}^3_4T\cdot{}^4_6T = \begin{bmatrix} -c_4c_5c_6-s_4s_6 & -c_4c_5s_6-s_4c_6 & c_4s_5 & a_3 \\ s_5c_6 & -s_5s_6 & c_5 & d_4 \\ -s_4c_5c_6-c_4s_6 & s_4c_5s_6-c_4c_6 & s_4s_5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{18}$$

$$\begin{aligned} c_1c_{23}p_x+s_1c_{23}p_y-s_{23}p_z-a_2c_3&=a_3 \\ -c_1s_{23}p_x-s_1s_{23}p_y-c_{23}p_z+a_2s_3&=d_4 \end{aligned} \tag{19}$$

These equations may be solved simultaneously for s_{23} and c_{23}, resulting in

$$\begin{aligned} s_{23}&=\frac{(-a_3-a_2c_3)\,p_z+(c_1p_x+s_1p_y)\,(a_2s_3-d_4)}{p_z^2+(c_1p_x+s_1p_y)^2} \\ c_{23}&=\frac{(a_2s_3-d_4)\,p_z+(-a_3-a_2c_3)\,(c_1p_x+s_1p_y)}{p_z^2+(c_1p_x+s_1p_y)^2} \end{aligned} \tag{20}$$

Since the denominators are equal and positive, they solve for the sum of θ_2 and θ_3 as,

$$\theta_{23}=A\tan 2\begin{bmatrix}(-a_3-a_2c_3)\,p_z-(c_1p_x+s_1p_y)\,(d_4-a_2s_3), \\ (a_2s_3-d_4)\,p_z-(a_3+a_2c_3)\,(c_1p_x+s_1p_y)\end{bmatrix} \tag{21}$$

Equation (21) computes four values of θ_{23} according to the four possible combinations of solutions for θ_1 and θ_3. Then, four possible solutions for θ_2 are computed as,

$$\theta_2=\theta_{23}-\theta_3 \tag{22}$$

Where the appropriate solution for θ_3 is used when forming the difference. Now the entire left side of equation (17) is known. Equating the 1, 3 elements from both sides of equation (17), as well as the 3, 3 elements,

$$\begin{aligned} r_{13}c_1c_{23}+r_{23}s_1c_{23}-r_{33}s_{23}&=-c_4s_5 \\ -r_{13}s_1+r_{23}c_1&=s_4s_5 \end{aligned} \tag{23}$$

As long as $s_5\neq 0$, they can solve for θ_4 as

$$\theta_4=A\tan 2(-r_{13}s_1+r_{23}c_1,\ -r_{13}c_1c_{23}-r_{23}s_1c_{23}-r_{33}s_{23}) \tag{24}$$

When θ_5 the manipulator is in a singular configuration in which joint axes 4 and 6 line up and cause the same motion of the last link of the robot. In this case, all that matters (and all that can be solved for) is the sum or difference of θ_4 and θ_6. This situation is detected by checking whether both arguments of the $A\tan 2$ in equation (24) are near zero. If so, θ_4 is chosen arbitrarily and when θ_6 is computed later, it will be computed accordingly.

If they consider equation (1) again, they can now rewrite it so that all the left-hand side is a function of only known and θ_4 by rewriting it as,

$$[{}^0_4T(\theta_4)]^{-1}\cdot{}^0_6T={}^4_5T(\theta_5)\cdot{}^5_6T(\theta_6) \tag{25}$$

where $[{}^0_4T(\theta_4)]^{-1}\cdot{}^0_6T$ is given by,

$$\begin{bmatrix} c_1c_{23}c_4+s_1s_4 & s_1c_{23}c_4-c_1s_4 & -s_{23}c_4 & -a_2c_3c_4+d_3s_4-a_3c_4 \\ c_1c_{23}s_4+s_1c_4 & -s_1c_{23}s_4-c_1c_4 & s_{23}s_4 & a_2c_3c_4+d_3c_4+c_3s_4 \\ -c_1s_{23} & -s_1s_{23} & -c_{23} & a_2s_3-d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{26}$$

Equating the 1, 3 elements from both sides of equation (25), as well as the 3, 3 elements,

$$r_{13}(c_1c_{23}c_4+s_1s_4)+r_{23}(s_1c_{23}c_4-c_1s_4)-r_{33}(s_{23}c_4)=-s_5 \tag{27a}$$

$$r_{13}(-c_1s_{23})+r_{23}(-s_1s_{23})+r_{33}(-c_{23})=c_5 \tag{27b}$$

So they can solve for θ_5 as,

$$\theta_5=A\tan 2(s_5,\ c_5) \tag{28}$$

Where s_5 and c_5 are given by equation (27a and

27b) above. Applying the same method one more time, they compute $({}^0_5T)^{-1}$ and write (1) in the form,

$$({}^0_5T)^{-1}\cdot{}^0_6T={}^5_6T(\theta_6) \tag{29}$$

Equating the 3, 1 elements from both sides of equation (25), as well as the 1, 1 elements as they have done before,

$$\theta_6=A\tan 2(s_6,\ c_6) \tag{30}$$

where,

$$s_6=-r_{11}(c_1c_{23}s_4-s_1s_4)-r_{21}(s_1c_{23}s_4+c_1c_4)+r_{31}(s_{23}s_4)$$

$$c_6=r_{11}[(c_1c_{23}c_4+s_1s_4)c_5-c_1s_{23}s_5]$$
$$+r_{21}[(s_1c_{23}c_4-c_1s_4)c_5-s_1s_{23}s_5]-r_{31}(s_{23}c_4c_5+c_{23}s_5)$$

Because of the plus-or-minus signs appearing in (11) and (15), these equations compute four solutions. Additionally, there are four more solutions obtained by "flipping" the wrist of the manipulator. For each of the four solutions computed above, they obtain the flipped solution by,

$$\begin{aligned}\theta_4'&=\theta_4+180^\circ\\ \theta_5'&=-\theta_5\\ \theta_6'&=\theta_6+180^\circ\end{aligned} \tag{31}$$

After all eight solutions have been computed, some or all of them may have to be discarded because of joint limit violations. Of the remaining valid solutions, usually the one closest to the present manipulator configuration is chosen.

3. Feedforward Neural Network

A Feedforward Neural Network (FNN) shown in Fig. 3 very loosely based on these ideas. In the most general terms, a FNN consist of large number of simple processor linked by weighted connections. By analogy, the processing nodes may be called 'neurons'. Each node output depends only on information that is locally available at the node, either stored internally or arriving via the weighed connections. Each unit receives inputs from many other nodes and transmits its output to yet other nodes. By itself, a single processing element is not very powerful ; it generates a scalar output with a single numerical value, which is a simple nonlinear function of its inputs. The power of the system emerges from the combination of many units in an appropriate way.

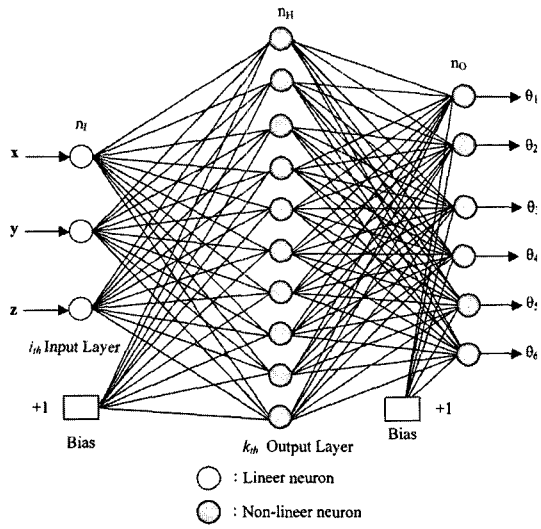

Fig. 3 Inverse Neural network model of PUMA robot

A network is specialised to implement different functions by varying the connection topology and the values of the connecting weights. Complex functions can be implemented by connecting units together with appropriate weights. In fact, it has been shown that a sufficiently large network with an appropriate structure and property chosen weights can approximate with arbitrary accuracy and function satisfying certain broad constraints.

Usually, the processing units have response like,

$$y_j=f\left(\sum_i u_i\right) \tag{32}$$

Where u_i are the output signals of hidden layer to output layer, $f(.)$ is a simple nonlinear function such as the sigmoid, or logistic function. This unit computers a weighted linear combination of its inputs and passes this through the nonlinearity to produce a scalar output. In general, it is a bounded non-decreasing nonlinear function ; the logistic function is a common choice.

$$g(z)=\frac{1}{1+e^{-z}} \tag{33}$$

This model is, of course, a drastically simplified approximation of real nervous systems. The in-

tend is to capture the major characteristics important in the information processing functions of real networks without varying too much about physical constraints imposed by biology.

3.1 Learning algorithm

The learning algorithm topology, which was employed for the neural network updating the weight can be described as follows; define the error function as,

$$J=\frac{1}{2}\sum_{i=1}^{n_O}(y_{di}(t)-y_i(t))^2 \tag{34}$$

Where $y_{di}(t)$ are the ith desired outputs and $y_i(t)$ are the ith outputs of the network. This error function is to be minimised with respect to all the unknown parameters Θ. In the steepest descent approach the parameter vector $\Theta=[\theta_1, \theta_2, \cdots, \theta_n]^T$ is adjusted using the increment vector $[\Delta\theta_1, \Delta\theta_2, \cdots, \Delta\theta_n]^T$ defined along the negative gradient direction of J,

$$\Delta\theta_i=-\eta\frac{\partial J}{\partial\theta_i} \tag{35}$$

Although the one-hidden layer model is used in the present application, it is useful to derive the gradient of J for the general case, and the result for the one-hidden-layer model can readily be obtained as a special case.

Starting from the output layer m of the network and setting $\theta_i=W_{ij}^m$, the application of the chain rule gives rise to,

$$\frac{\partial J}{\partial W_{ij}^m}=\frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial W_{ij}^m} \tag{36}$$

From equation (34)

$$\frac{\partial J}{\partial y_i}=-(y_{di}-y_i)=-\delta_i^m \tag{37}$$

Where δ_i^m is called the error signal of the i-th neuron in the mth layer. From equation (36)

$$\frac{\partial y_i}{\partial W_{ij}^m}=x_j^{m-1} \tag{38}$$

Thus,

$$\frac{\partial J}{\partial W_{ij}^m}=-\delta_i^m x_j^{m-1} \tag{39}$$

Next consider the $(m-1)$th layer. Using the chain rule yields:

$$\frac{\partial J}{\partial w_{ij}^{m-1}}=\sum_{k=1}^{n_O}\frac{\partial J}{\partial y_k}\times\frac{\partial y_k}{\partial x_i^{m-1}}\times\frac{\partial x_i^{m-1}}{\partial z_i^{m-1}}\times\frac{\partial z_i^{m-1}}{\partial W_{ij}^{m-1}} \tag{40}$$

Then

$$\frac{\partial x_i^{m-1}}{\partial z_i^{m-1}}=g'(z_i^{m-1}) \tag{41}$$

and

$$\frac{\partial z_i^{m-1}}{\partial W_{ij}^{m-1}}=x_j^{m-2} \tag{42}$$

$$g'(z)=\frac{\partial g(z)}{\partial z} \tag{43}$$

and $g(z_i)$ is the activation of neuron i. By defining the error signal for the ith neuron of the $(m-1)$th layer as,

$$\delta_i^{m-1}=g'(z_i^{m-1})\sum_{k=1}^{n_O}\delta_k^m W_{ki}^m \tag{44}$$

Equation (34) can be rewritten as,

$$\frac{\partial J}{\partial W_{ij}^{m-1}}=-\delta_i^{m-1}x_j^{m-2} \tag{45}$$

Similarly, it can be shown that,

$$\frac{\partial J}{\partial b_i^{m-1}}=-\delta_i^{m-1} \tag{46}$$

Where b_i^{m-1} is the bias input to neuron i in layer $m-1$. By carrying on this procedure, Equations (44)-(46) can be used as a general algorithm for updating weights in other layers. Equations (44)-(46) indicate how the error signals propagate backwards from the output layer of the network through the hidden layer to the input layer, hence the name "BP".

The steepest-descent minimisation of the error function defined in Equation (34) produces the following increments for updating Θ (Yıldırım, 2004):

$$\Delta W_{ij}^m(t)=\eta_w\delta_i^m(t)x_j^{m-1}(t) \tag{47}$$

$$\Delta b_i^m(t)=\eta_b\delta_i^m(t) \tag{48}$$

where in the output layer,

$$\delta_i^m(t)=y_{di}(t)-y_i(t) \tag{49}$$

and in other layers,

$$\delta_i^m(t)=g'(z_i^m(t))\sum_j\delta_j^{m+1}(t)\,W_{ji}^{m+1}(t-1) \tag{50}$$

The constants $\eta_w(0<\eta_w<1)$ and $\eta_b(0<\eta_b<1)$ represent the learning rates for the weights and

biases respectively. In practice, a large value of the learning rate would be preferable, because this would result in rapid learning. Unfortunately, a large value of the learning rate can also lead to oscillation or even divergence. To help speed up learning but avoid undue oscillations, a momentum term is usually included so that equations (47) and (48) become,

$$\Delta W_{ij}^m(t)=\eta_w\delta_i^m(t)x_j^{m-1}(t)+\alpha_w\Delta W_{ij}^m(t-1) \quad (51)$$

$$\Delta b_i^m(t)=\eta_b\delta_i^m(t)+\alpha_b\Delta b_i^m(t-1) \quad (52)$$

Where α_w and α_b are momentum constants, which determine the effect of past changes of $\Delta W_{ij}^m(t)$ and $\Delta b_i^m(t)$ on the current updating direction in the weight and the bias space respectively. This effectively filters out high frequency variations in the error surface. To summarise, the BP algorithm updates the weights and thresholds of the networks according to,

$$W_{ij}^m(t)=W_{ij}^m(t-1)+\Delta W_{ij}^m(t) \quad (53)$$

and

$$b_i^m(t)=b_i^m(t-1)+\Delta b_i^m(t) \quad (54)$$

Where the increments $\Delta W_{ij}^m(t)$ and $\Delta b_i^m(t)$ are given in equations in (51) and (52). The neural network was trained and tested with five types of learning algorithms. The algorithms can be described in the following forms.

3.1.1 Online back propagation algorithm (Case 1)

Online Back Propagation (OBP), which updates the weights after each pattern is presented to the network. Back Propagation is the most commonly used training algorithm for neural networks. The weights are updated as follows,

$$\Delta w_{ij}(t)=-\eta\frac{\partial E(t)}{\partial w_{ij}(t)}+\alpha\Delta w_{ij}(t-1) \quad (55)$$

Where η is the learning rate, and α is the momentum constant.

3.3.2 Online back propagation random algorithm (Case 2)

Online Back Propagation with the order of the input patterns randomized prior to each epoch. This makes the learning process stochastic and is preferred in most cases.

3.1.3 Batch back propagation algorithm (Case 3)

Batch Back Propagation Algorithm (BBP) weight updates occurring after each epoch.

3.1.4 Delta bar delta algorithm (Case 4)

Delta Bar Delta (DBD) is an adaptive learning rate method in which every weight has its own learning rate. The learning rates are updated based on the sign of the gradient. If the gradient does not change signs on successive iterations then the step size is increased linearly. If the gradient changes signs, the learning rate is decreased exponentially. In some cases this method seems to learn much faster than non-adaptive methods. Learning rates $\eta(t)$ are updated as follows:

$$\Delta\eta(t)=\begin{cases}K & \text{if } \bar{\delta}(t-1)\,\delta(t)>0\\ -\varphi\eta(t) & \text{if } \bar{\delta}(t-1)\,\delta(t)<0\\ 0 & \text{else}\end{cases} \quad (56)$$

Where $\delta(t)=\partial E/\partial w$ at time t and $\bar{\delta}$ is the exponential average of past values of δ.

$$\bar{\delta}(t)=(1-\theta)\,\delta(t)+\theta\bar{\delta}(t-1) \quad (57)$$

3.1.5 Quick propagation algorithm (Case 5)

Quick Propagation Algorithm (QP) is a training method based on the following assumptions:

(1) $E(w)$ for each weight can be approximated by a parabola that opens upward

(2) The change in slope of $E(w)$ for this weight is not affected by all other weights that change at the same time. The weight update rule is:

$$\Delta w(t)=\frac{S(t)}{S(t-1)-S(t)}\Delta w(t-1)-\eta S(t) \quad (58)$$

where $S(t)=\partial E/\partial w(t)$. The numerator is the derivative of the error with respect to the weight and $[S(t-1)-S(t)]/\Delta w(t-1)$ is a finite difference approximation of the second derivative. Together these approximate Newton's method for minimizing a one-dimensional function $f(x)$: $\Delta x=-f'(x)/f''(x)$. To avoid taking an infinite

backward step, or a backward uphill step, a maximum growth factor parameter μ is introduced. No weight change is allowed to be larger than μ times the previous weight change. Quick Propagation has a fixed learning rate parameter, η, that needs to be chosen to suit the problem.

4. Simulation

The neural network a feedforward type with different learning algorithm. It is well known that the exact inverse kinematics solution of the accurate robot path control is difficult to find. Therefore, the use of the neural network to find the robot kinematics parameters avoids the need for a prior knowledge or assumptions about this mechanism. The simulation could be divided into five sub section for describing five cases. Desired kinematics parameters for following prescribed trajectory of the PUMA type robot five cases. Desired kinematics parameters for following prescribed trajectory of the PUMA type robot are given in Table 1.

4.1 Simulation I (Case 1)

In this section, simulation was carried out using Online Back Propagation (OBP) neural network for modelling of inverse kinematics of PUMA type robot. The theory of the OBP-Neural network is deeply described in section 3.1.1.

The results for both approaches are given in Table 2 for the case of OBP-Neural network. The results proved that the OBP-Neural network approximation is not suitable for modelling PUMA types kinematics.

4.2 Simulation II (Case 2)

An Online Back Propagation Random (OBPR)

Table 2 Desired and Neural network angular position results of the PUMA 560 robot for Case 1

θ_{1NN}	θ_{1D}	θ_{2NN}	θ_{2D}	θ_{3NN}	θ_{3D}	θ_{4NN}	θ_{4D}	θ_{5NN}	θ_{5D}	θ_{6NN}	θ_{6D}
73.582117	73.760000	−19.824662	−19.790000	129.754903	129.710000	0.000000	0.000000	69.991844	70.080000	73.579090	73.760000
61.500662	61.400000	−39.350606	−39.230000	164.343842	164.160000	0.000000	0.000000	55.016455	55.060000	61.510835	61.400000
47.652018	47.470000	−48.689952	−48.900000	179.661747	179.180000	0.000000	0.000000	49.124239	49.220000	47.638948	47.470000
34.425725	34.440000	−51.053372	−51.420000	183.681451	184.120000	0.000000	0.000000	47.389090	47.300000	34.466373	34.440000
20.077613	20.080000	−50.476767	−50.260000	182.592768	182.010000	0.000000	0.000000	48.094304	48.250000	20.073847	20.080000
8.841822	8.830000	−44.357177	−44.340000	18.248891	17.210000	0.000000	0.000000	52.117124	52.130000	8.836081	8.830000
3.437985	3.320000	−36.995390	−36.690000	160.934639	160.350000	0.000000	0.000000	56.169283	56.340000	3.439293	3.320000
−0.863167	−1.700000	−25.297784	−25.250000	141.124166	141.000000	0.000000	0.000000	64.170485	64.250000	−0.861409	−1.700000
3.381968	3.330000	−4.323822	−4.210000	114.390282	114.360000	0.000000	0.000000	69.026524	69.050000	3.383112	3.330000

Table 3 Desired and Neural network angular position results of the PUMA 560 robot for Case 2

θ_{1NN}	θ_{1D}	θ_{2NN}	θ_{2D}	θ_{3NN}	θ_{3D}	θ_{4NN}	θ_{4D}	θ_{5NN}	θ_{5D}	θ_{6NN}	θ_{6D}
73.582117	73.760000	−19.824662	−19.790000	129.754903	129.710000	0.000000	0.000000	69.991844	70.080000	73.579090	73.760000
61.500662	61.400000	−39.350606	−39.230000	164.343842	164.160000	0.000000	0.000000	55.016455	55.060000	61.510835	61.400000
47.652018	47.470000	−48.689952	−48.900000	179.661747	179.180000	0.000000	0.000000	49.124239	49.220000	47.638948	47.470000
34.425725	34.440000	−51.053372	−51.420000	183.681451	184.120000	0.000000	0.000000	47.389090	47.300000	34.466373	34.440000
20.077613	20.080000	−50.476767	−50.260000	182.592768	182.010000	0.000000	0.000000	48.094304	48.250000	20.073847	20.080000
8.841822	8.830000	−44.357177	−44.340000	18.248891	17.210000	0.000000	0.000000	52.117124	52.130000	8.836081	8.830000
3.437985	3.320000	−36.995390	−36.690000	160.934639	160.350000	0.000000	0.000000	56.169283	56.340000	3.439293	3.320000
−0.863167	−1.700000	−25.297784	−25.250000	141.124166	141.000000	0.000000	0.000000	64.170485	64.250000	−0.861409	−1.700000
3.381968	3.330000	−4.323822	−4.210000	114.390282	114.360000	0.000000	0.000000	69.026524	69.050000	3.383112	3.330000

algorithm is employed to update the weights of the neural network. As depicted in section 3.1.2, the results of OBPR-Neural network are not acceptable for calculating the kinematic parameters of the PUMA robot. The results of the both approach are given in Table 3.

4.3 Simulation III (Case 3)

In this section, Batch Back Propagation (BBP) learning rule is used to adjust the weights of the neural network for comparison. Table 4 indicates the results of desired and neural network approach.

4.4 Simulation IV (Case 4)

The section presents the results of, Delta Bar Delta (DBD) learning algorithm for finding exact kinematic parameters of the PUMA type robot. The results of DBD-Neural network are considerably better than the cases of BBP, OBR and OBP Neural network approach (see Table 5).

4.5 Simulation V (Case 5)

Quick Propagation (QP) learning algorithm is used to update the weights of the neural network. For that reason, the network could be named QP-Neural network. In this approach, the results of the neural network are exactly following the results of desired approach.

Finally, QP-Neural network could be used as neural predictor of the PUMA type robot in real time applications. On the other hand, it is expected that the proposed QP-ANN method will be considerably faster than the generalized analytical model for robots that do not have a straightforward closed-form solution (see Table 6).

Table 4 Desired and Neural network angular position results of the PUMA 560 robot for Case 3

θ_{1NN}	θ_{1D}	θ_{2NN}	θ_{2D}	θ_{3NN}	θ_{3D}	θ_{4NN}	θ_{4D}	θ_{5NN}	θ_{5D}	θ_{6NN}	θ_{6D}
73.525084	73.760000	−19.792821	−19.790000	129.695004	129.710000	0.000000	0.000000	70.001358	70.080000	73.523104	73.760000
61.248994	61.400000	−39.288109	−39.230000	163.805406	164.160000	0.000000	0.000000	55.064854	55.060000	61.244168	61.400000
47.621977	47.470000	−48.640461	−48.900000	179.319854	179.180000	0.000000	0.000000	49.184029	49.220000	47.609410	47.470000
34.611889	34.440000	−51.090943	−51.420000	183.453362	184.120000	0.000000	0.000000	47.411163	47.300000	34.652953	34.440000
20.090462	20.080000	−50.135568	−50.260000	181.964331	182.010000	0.000000	0.000000	48.143067	48.250000	20.103024	20.080000
8.834381	8.830000	−44.343374	−44.340000	18.237062	17.210000	0.000000	0.000000	52.128698	52.130000	8.833020	8.830000
3.218828	3.320000	−36.735299	−36.690000	161.176848	160.350000	0.000000	0.000000	56.231682	56.340000	3.214785	3.320000
−1.042621	−1.700000	−25.281256	−25.250000	141.249989	141.000000	0.000000	0.000000	64.247111	64.250000	−1.050617	−1.700000
3.325512	3.330000	−4.378204	−4.210000	114.358283	114.360000	0.000000	0.000000	69.053819	69.050000	3.326043	3.330000

Table 5 Desired and Neural network angular position results of the PUMA 560 robot for Case 4

θ_{1NN}	θ_{1D}	θ_{2NN}	θ_{2D}	θ_{3NN}	θ_{3D}	θ_{4NN}	θ_{4D}	θ_{5NN}	θ_{5D}	θ_{6NN}	θ_{6D}
73.700788	73.760000	−19.792870	−19.790000	129.707215	129.710000	0.000000	0.000000	70.075776	70.080000	73.699255	73.760000
61.410220	61.400000	−39.274314	−39.230000	163.907198	164.160000	0.000000	0.000000	55.063390	55.060000	61.412342	61.400000
47.460848	47.470000	−48.745389	−48.900000	179.392289	179.180000	0.000000	0.000000	49.212819	49.220000	47.443501	47.470000
34.388821	34.440000	−51.058308	−51.420000	182.991986	184.120000	0.000000	0.000000	47.362387	47.300000	34.429493	34.440000
20.078258	20.080000	−50.219459	−50.260000	182.883876	182.010000	0.000000	0.000000	48.223369	48.250000	20.074645	20.080000
8.826358	8.830000	−44.331187	−44.340000	17.211942	17.210000	0.000000	0.000000	52.127689	52.130000	8.809625	8.830000
3.295494	3.320000	−36.667034	−36.690000	160.471260	160.350000	0.000000	0.000000	56.314065	56.340000	3.297846	3.320000
−1.139768	−1.700000	−25.227837	−25.250000	141.108433	141.000000	0.000000	0.000000	64.257171	64.250000	−1.137601	−1.700000
3.330382	3.330000	−4.291308	−4.210000	114.354208	114.360000	0.000000	0.000000	69.049262	69.050000	3.331647	3.330000

Table 6 Desired and Neural network angular position results of the PUMA 560 robot for Case 5

θ_{1NN}	θ_{1D}	θ_{2NN}	θ_{2D}	θ_{3NN}	θ_{3D}	θ_{4NN}	θ_{4D}	θ_{5NN}	θ_{5D}	θ_{6NN}	θ_{6D}
73.694181	73.760000	−19.791702	−19.790000	129.709856	129.710000	0.000000	0.000000	70.079939	70.080000	73.706879	73.760000
61.409799	61.400000	−39.276189	−39.230000	163.965698	164.160000	0.000000	0.000000	55.054047	55.060000	61.389241	61.400000
47.391736	47.470000	−48.656978	−48.900000	179.015866	179.180000	0.000000	0.000000	49.230765	49.220000	47.418487	47.470000
34.371468	34.440000	−51.250877	−51.420000	183.150520	184.120000	0.000000	0.000000	47.421525	47.300000	34.394114	34.440000
20.112724	20.080000	−50.115327	−50.260000	181.640313	182.010000	0.000000	0.000000	48.140810	48.250000	20.123819	20.080000
8.833952	8.830000	−44.335007	−44.340000	17.499646	17.210000	0.000000	0.000000	52.129205	52.130000	8.834512	8.830000
2.985874	3.320000	−37.227865	−36.690000	160.543932	160.350000	0.000000	0.000000	56.333207	56.340000	2.988753	3.320000
−0.923795	−1.700000	−25.250669	−25.250000	141.132699	141.000000	0.000000	0.000000	64.248849	64.250000	−0.918493	−1.700000
3.332300	3.330000	−4.210000	−4.210000	114.335754	114.360000	0.000000	0.000000	69.048959	69.050000	3.331194	3.330000

5. Conclusion and Discussion

In this study, five types of neural network predictor strategies have been used and developed inverse position controller. The PUMA robot has configuration, in so far as its last three adjacent joint axes intersect. Thus, the PUMA 560 robot has an efficient closed-form inverse kinematic solution. However, disadvantages can be found ith this style of robot. This geometry may not be appropriate for applications requiring the arm to be able to reach a desired position in more than one way. In addition, the joints of the robots without wrist offsets do not have complete rotational capabilities. These robots usually do not have closed-form inverse kinematic solution. Thus, QP-Neural network method can be implemented as a single-stage network. The proposed QP-Neural network structure previously described was shown to produce better training performance and adaptability than the other structures. Structural and learning parameters of these Cases are detailed in Table 7. Root Mean Square Errors (RMSEs) for these approaches are also given in Table 8.

The proposed iterative strategy provides a superior alternative for solving the inverse kinematics problem of manipulators for which it is difficult to derive the kinetic model or to obtain the closed-form solution. The strategy can generate the solution from measurement of the physical location of the manipulator in both the cartesian and the inbuilt learning abilities of the ANNs.

Table 7 Structural and training parameters of the feedforward neural networks with different learning algorithms

NN Type	η	μ	N	n_I	n_H	n_O	AF
OBP	0.05	0.5	1000000	3	10	6	Logistic
OBPR	0.05	0.5	1000000	3	10	6	Logistic
BBP	0.05	0.5	1000000	3	10	6	Logistic
DBD	0.05	0.5	1000000	3	10	6	Logistic
QP	0.05	0	1000000	3	10	6	Logistic

Table 8 RMS Errors for all cases

Cases	Max RMS Error
Case 1	0.291278
Case 2	0.230304
Case 3	0.224622
Case 4	0.294192
Case 5	**0.21345**

Acknowledgments

This research results consisted of part of project DPT-05-06. The authors would like to express their thanks to Erciyes University for supporting this project.

References

Aspragathos, N. A. and Dimitros, J. K., 1998,

"A Comparative Study of Three Methods for Robot Kinematics," *IEEE Transcations on Systems Man and Cybernetics Part B-Cybernetics*, Vol. 28, No. 2, pp. 135~145.

Bravo, F. J. A., 1990, "Multi Layer Back Propagation Network for Learning The Forward and Inverse Kinematic Equations," *In Proceedings of The International JCNNs*, Washington D. C., Vol. 2, pp. 319~322.

Choi, K. B., 2003, "Kinematic Analysis and Optimal Design of 3-PPR Planar Parallel Manipulator," *KSME International Journal*, Vol. 17, No. 4, pp. 528~537.

Craig, J. J., 1989, *Introduction to Robotics : Mechanics and Control*, Addison Wesley Publishing Company.

Guez, A. and Ahmad, Z., 1989, "Accelerated Convergence in The Inverse Kinematics Via Multilayer Feed Forward Neural Networks," *In Proceeding of The International JCNNs*, Washington D. C., Vol. 2, pp. 341~347.

Guo, J. and Cherhassky, V., 1989, "A Solution to The Inverse Kinematic Problem in Robotics Using Neural Network Processing," *In Proceeding of The International JCNNs*, Washington D. C., Vol. 2, pp. 299~304.

Ha, I. C. and Han, M. C., 2004, "A Robust Control with a Neural Network Structure for Uncertain Robot Manipulator," *KSME International Journal*, Vol. 18, No. 11, pp. 1916~1922.

Jin, T. S. and Lee, J. -M., 2003, "The Position/Orientation Determination of Mobile-Task Robot Using an Active Calibration Scheme," *KSME International Journal*, Vol. 17, No. 10, pp. 1431~1442.

Kang, T., Choi, H. and Kim, M., 2003, "Development of Anthropomorphic Robot Hand SKK Robot Hand I," *KSME International Journal*, Vol. 17, No. 2, pp. 230~238.

Lee, J., Lim, J., Park, C. W. and Kim, S., 2004, "Adaptive Model Reference Control Based on Tagaki-Sugeno Fuzzy Models with Applications to Flexible Joint Manipulators," *KSME International Journal*, Vol. 18, No. 3, pp. 337~346.

Pashkevich, A., 1997, "Real-Time Inverse Kinematics for Robots with Offset and Reduced Wrist," *Control Engineering Practice*, Vol. 5, No. 10, pp. 1443~1450.

Tchon, K., 2000, "Hyperbolic Normal Forms for Manipulator Kinematics," *IEEE Transactions on Robotics and Automation*, Vol. 16, No. 2, pp. 196~201.

Yıldırım, Ş., 2004, "Adaptive Robust Neural Controller for Robots," *Robotics and Autonomous Systems*, Vol. 46, No. 3, pp. 175~184.