Development of an intelligent system for prediction of inverse kinematics of robot manipulator

[1]Jha, P. and [2]Biswal, B. B.

National Institute of Technology, Rourkela, Odisha, India
[1]*jha_ip007@hotmail.com*
[2]*bbbiswal@nitrkl.ac.in*

Abstract: - Inverse kinematics comprises the computation need to find the joint angles for a given Cartesian position and orientation of the end effectors. There is no unique solution for the inverse kinematics thus necessitating application of artificial neural network models. This paper proposes three different types of structured artificial neural network (ANN) models to find the solution of inverse kinematics. The first one is an ANN model which is MLP (multi-layer perceptron's) and is popular as back propagation neural network model. In this gradient descent type of learning rules are applied. The second kind of ANN model is PPN (polynomial poly-processor neural network) where polynomial equation is used and the last one is Pi-network.

Keywords: - *Robot manipulator, Inverse kinematics, neural network*

1. INTRODUCTION

Robot manipulator is composed of a serial chain of rigid links connected typically to each other by revolute or prismatic joints. A revolute joint rotates about a motion axis whereas prismatic joint slides along a motion axis. Each robot's joint location is usually defined relative to the neighbouring joint. The relation between successive joints is described by 4X4 homogeneous transformation matrices that have orientation and position data of robots. The number of those transformation matrices determines the degrees of freedom of robots. The product of such matrices produces final orientation and position data of an n degrees of freedom robot manipulator. This is of fundamental importance in calculating desired joint angles for robot manipulator design and control. In most robotic applications the desired positions and orientations of the end effectors are specified by the user in Cartesian coordinates. The corresponding joint values must be computed at high speed by the inverse kinematics transformation [1-3]. For a manipulator with n degree of freedom, at any instant of time joint variables is denoted by $\theta_i = \theta(t)$, i = 1, 2,3 .....n and position variables x_j = x(t), j = 1, 2, 3 .....m.The relations between the end-effectors position x(t) and joint angle θ(t) can be represented by forward kinematic equation

$$x(t) = f(\theta(t)) \quad (1)$$

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

$$\theta(t) = f^{'}(x(t)) \quad (2)$$

Solution of (2) is not unique due to nonlinear, uncertain and time varying nature of the governing equations. The different techniques used for solving inverse kinematics can be classified as algebraic, geometric and iterative. The algebraic methods do not guarantee closed form solutions. In case of geometric methods, closed form solutions for the first three joints of the manipulator must exist geometrically.

The iterative methods converge to only a single solution depending on the starting point and may not work near singularities [4, 5]. If the joints of the manipulator are more complex, the inverse kinematics solution by using these traditional methods becomes quite time consuming. To compound the problem further, robots have to work in the real world that cannot be modelled concisely using mathematical expressions. In recent years, there have been increasing research interest of artificial neural networks and many efforts have been made on applications of neural networks for the inverse kinematics problems. The most

significant features of neural networks are the extreme flexibility due to the learning ability and the capability of nonlinear functions approximations. This fact leads us to expect that neural networks can be an excellent tool for solving the inverse kinematics problem in robot manipulators with overcoming the difficulties of algebraic, geometric and iterative methods [6-11]. Several Neural Network approaches have been proposed such as MLP (multiple layer perceptrons) and PPN (Polynomial poly-processor neural network) method. This unsupervised method learns the functional relationship between input (Cartesian) space and output (joint) space based on a localized adaptation of the mapping, by using the manipulator itself under joint control and adapting the solution based on a comparison between the resulting locations of the manipulator's end effectors in Cartesian space with the desired location. The forward kinematic equations always have a unique solution, and the resulting Neural net can be used as a starting point for further refinement when the manipulator does become available. Artificial neural network especially MLP and PPN are used to learn the forward and the inverse kinematics equations of two degrees freedom (DOF) robot arm.

A neural network based inverse kinematics solution method yields multiple and precise solutions with an acceptable error and it is suitable for real-time adaptive control of robotic manipulators. Neural networks were capable of learning complex functions, which led to their use in applications including pattern recognition, function approximation, data fitting, and control of dynamic systems. The details of inverse kinematics were briefly discussed as follows.

The objective of the present work is to solve the inverse kinematic problem of 3R manipulator using different models of neural network. The advantage of neural network is that it can be used in any non-redundant or redundant manipulator. Most of the problems are dealing with kinematics control and this paper highlighted one application on inverses kinematics hence dynamic control not discussed here. The comparative study and results presented in this paper indicate the feasibility of using these ANN for learning complex input/output relations of robot kinematics control (based on computation of forward and inverse mapping between joint space and Cartesian space). The details of inverse kinematics briefly discussed as follows. Solution of equation (2) is not unique due to nonlinear, uncertain and time varying nature of the governing equations.

2. MATHEMATICAL MODELLING OF MANIPULATOR

In this section, we consider the traditional mathematical models for forward kinematics and inverse kinematics of robot manipulators, only to build correct model for the proposed neural network model. The purpose of this application is to introduce to robot kinematics, and the concepts related to both open and closed kinematics chains. The Inverse Kinematics is the opposite problem as compared to the forward kinematics, forward kinematics gives the exact solution but in case of inverse kinematics it gives multiple solutions. The set of joint variables when added that give rise to a particular end effectors or tool piece pose. Figure 1 shows the basic joint configuration of 3R planar manipulator. Position and orientation of the end effectors can be written in terms of the joint coordinates in the following way:

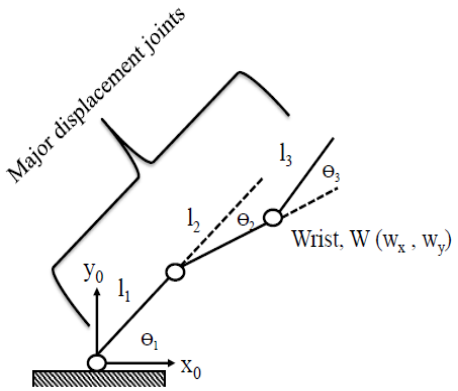

Figure 1: 3R manipulator

Forward kinematics is given by,

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \quad (3)$$

$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \quad (4)$$

$$\phi = \theta_1 + \theta_2 + \theta_3 \quad (5)$$

ϕ represents orientation of the tool.

All the angles have been measured counter clockwise and the link lengths are assumed to be positive going from one joint axis to the immediately distal joint axis. Equation (3) is a set of three nonlinear equations that describe the relationship between end effectors coordinates and joint coordinates.

The set of equations are given for the end effectors coordinates in terms of joint coordinates.

However, to find the joint coordinates for a given set of end effectors coordinates (x, y, θ); one needs to solve the nonlinear equations for $\theta_1, \theta_2 \text{ and } \theta_3$. Inverse kinematics,

$$\theta_2 = a\tan 2(\sin\theta_2, \cos\theta_2)$$
$$= a\tan 2(\pm\sqrt{1-(\cos\theta_2)^2}, \frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2} \quad (6)$$
$$\theta_1 = a\tan 2(y,x) - a\tan 2(k_2,k_1) \quad (7)$$

Where,

$$k_2 = l_1 + l_2\cos\theta_2$$
$$k_1 = l_2\sin\theta_2$$
$$\theta_3 = \phi - \theta_1 - \theta_2 \quad (8)$$

3. ANN APPROACH FOR 3R PLANAR MANIPULATOR

There are many solutions that are available for any single point or position of the end effector. We propose the solution using various ANN models. The network is trained with data for a number of end effector positions expressed in Cartesian co-ordinates and the corresponding joint angles. The data consist of the different configurations available for the arm. For the 3R revolute manipulator, there are various orientations or poses that are possible for every position of the end effector in Cartesian space. The different poses of the arm are then used to train a three-layer, fully connected back-propagation model (Figure 2). In this study, various neural-network structures have been studied, and it was found that when there are multiple outputs and multiple inputs with hidden layers, the MLP gives better results. For both neural-network structures, when the number of neurons in the hidden layer(s) is equal to the number of neurons in the input layer, the ANN generates better results.

Multi-layered perceptron network has a better ability to learn the correspondence between the input patterns and teaching values obtained from many data samples by means of the error back-propagation algorithm. The model uses the error back-propagation algorithm. In this study MLP model with back-propagation is giving better result. This result in three sets of weights for each manipulator arm after the training session was over.

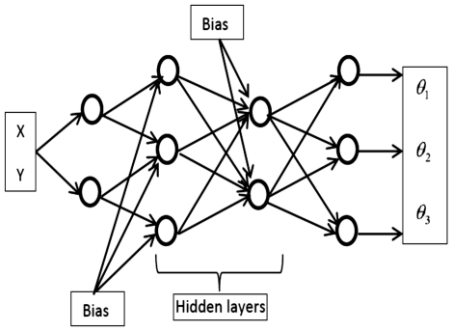

Figure 2: Block Diagram of the System using ANN.

The signals, n_h, are presented to a hidden layer neuron in the network via the input neurons. Each of the signals from the input neurons is multiplied by the value of the weights of the connection, w_j, between the respective input neurons and the hidden neuron. The net input to a hidden neuron is calculated as the sum of the values for all connections coming into the neuron [12, 13].

Net input of hidden neurons (for k inputs) =

$$n_h = \sum_{j=1}^{k} w_j \times o_{jn} \quad (9)$$

The output, o_{mj} of a hidden neuron as a function of its net input. The sigmoid function is:

$$Output = o_{mj} = \frac{1}{1+e^{-n_h}} \quad (10)$$

Once the outputs of the hidden layer neurons have been calculated, the net input to each output layer is calculated in a similar manner as in equation 9. According to Fig. 2, the model of the 3-layer perceptron neural network state calculation is combined with backward error propagation and weight adjustment calculations that represent the network's learning, or training.

$$\delta = f'(n)(d - o_m) \quad (11)$$

And from equation 11:

$$\delta = o_m(1-o_m)(d-o_m) \quad (12)$$

Where, d is the target or desired value, and o_m is the actual value from output neuron after going through the feed forward calculation. The error calculation was implemented on a neuron-by-neuron basis over the entire set (epoch) of patterns.

This error value, δ, was used to perform the appropriate weight adjustments of the weight connection between the output layer and hidden layer.

$$\delta_h = f'(n_h)\sum_{l=0}^{k_l} w_{lh}\delta_l = o_h(1-o_h)\sum_{l=0}^{k_l} w_{lh}\delta_l \qquad (13)$$

Where, δ_h is the error value of the hidden layer, δ_l is the error value of the output layer, o_h is the output of the sigmoid function and w_{lh} is the connection weights between the output and hidden layers. The weight changes were calculated according to equation 14.

$$w(\text{old}) = w(\text{new}) + \eta\delta_o + \alpha[\Delta w(\text{old})] \qquad (14)$$

The speed of convergence of the network depends on the training rate, η, and the momentum factor, α.

In this work, a two hidden layer neural network with two inputs, *x*,and y, and three outputs, θ_1, θ_2 *and* θ_3 was trained using the back-propagation algorithm described earlier, this was done along a trajectory of the end-effector in the xy -plane.

4. PPN AND Pi-NETWORK FORMULATION

Weierstrass approximation theorem states that any function which is continuous in a closed interval can be uniformly approximated within any prescribed tolerance over that interval by some polynomial. Figure 3 depicts a PPN network where X is the input pattern given by

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

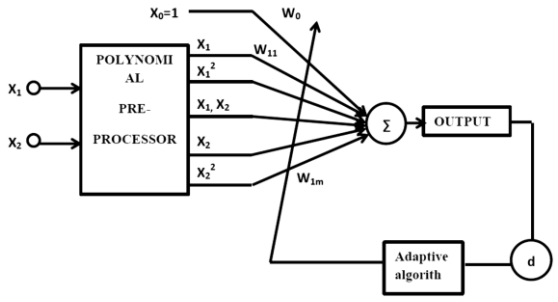

Figure 3: Polynomial perceptron network.

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

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

Where, $W = [w_0, x_{11}, x_{12}, w_1, w_2]^T$ and $X^* = [1, x^2{}_1, x_1, x_2, x^2{}_2, x_1, x_2]^T$

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

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

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

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

The input pattern X to the PPN at time n is the channel output vector X (n). This is then converted into X*(n) by passing it into a polynomial preprocessor. The weighted sum of the components of X*(n) is passed through a nonlinear function sigmoid and pure linear function to produce the output as shown in figure 3. The output of the PPN is compared with the desired response (d) to generate an error which is then used to update its weights by the BP algorithm.

Similarly in case of Pi-network the basic difference is polynomial equation is multiplied by Π=3.14.

5. SIMULATION RESULTS AND PERFORMANCE ANALYSIS OF MLP, PPN AND π-NETWORK

Three different models have been taken for the validation of the results and the models are: MLP (Multi-layer perceptron), PPN (Polynomial perceptron network) and π-network which are considered for the analysis of inverse kinematics problem, simulation studies are carried out by using MATLAB.

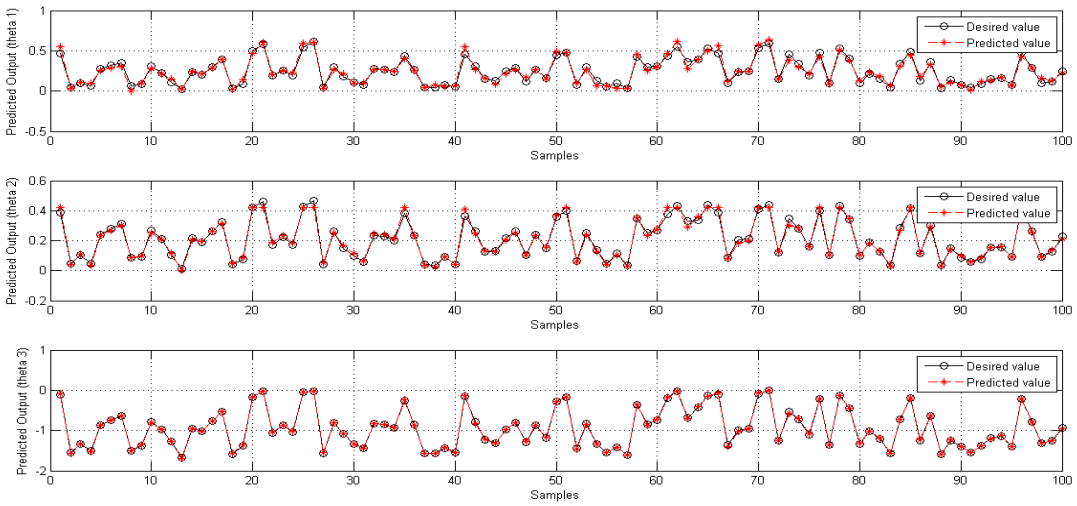

Figure 4: Comparison of desired and predicted value of joint angles for first configuration using MLFF model

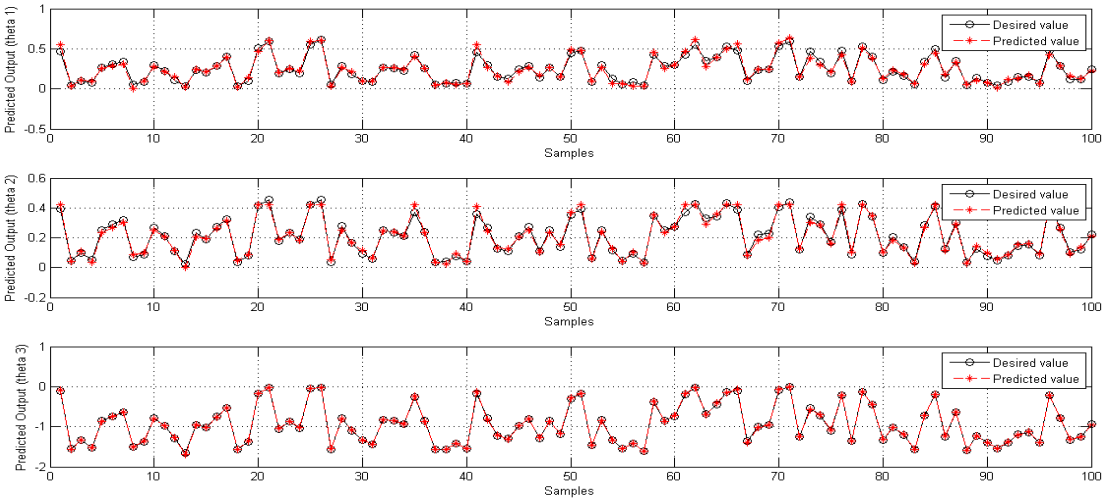

Figure 5: Comparison of desired and predicted value of joint angles for second configuration using MLFF model

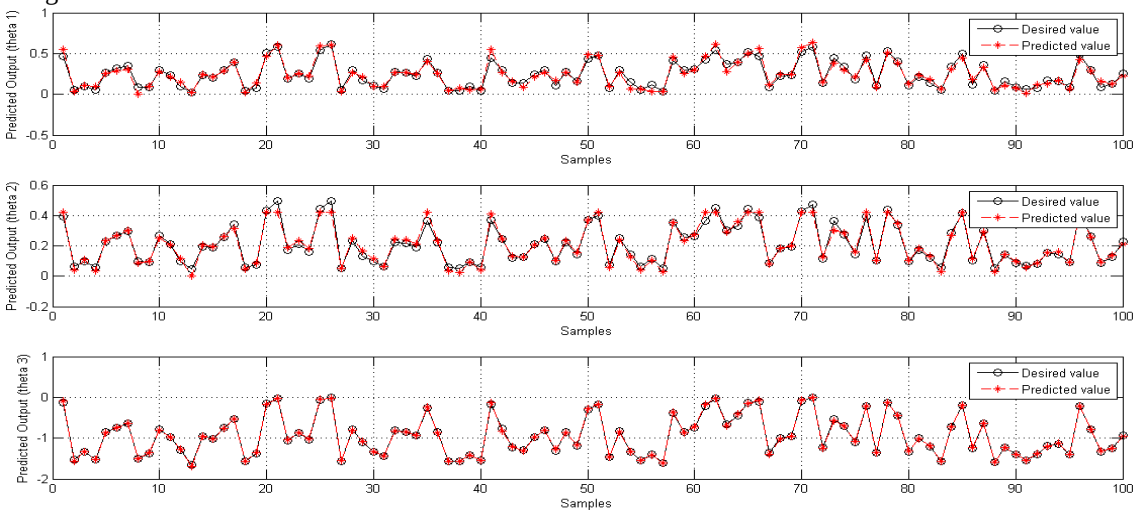

Figure 6: Comparison of desired and predicted value of joint angles for third configuration using MLFF model

Training data sets were generated by using equation (6, 7 and 8). A set of 1000 data sets were first generated as per the formula for this the input parameter X and Y coordinates in meters. These

data sets were basis for the training and evaluation or testing the ANN models. Out of the sets of 1000 data points, 900 were used as training data and 100 were used for testing for ANN models.

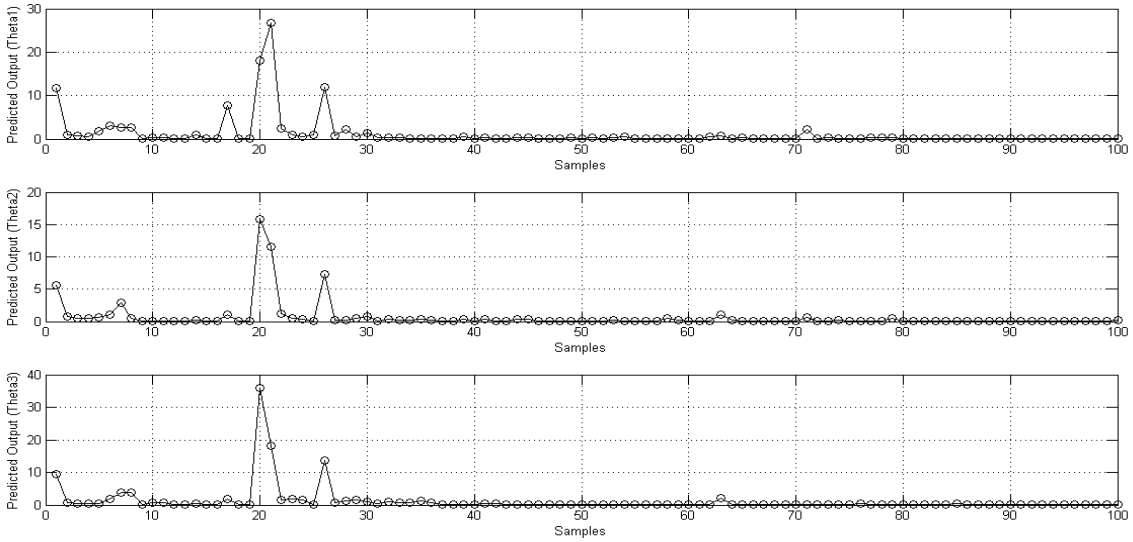

Figure 7: mean square error for joint angles using PPN model

Back-propagation algorithm was used for training the network and for updating the desired weights. In this work epoch based training method was applied. The comparison of desired and predicted value of joint angles of MLP model for 100 epochs for θ_1, θ_2 *and* θ_3 has been represented in Figure 4, 5 and 6. In this work three configurations have been taken for the better result. Figure 4 depicts the configuration which is giving better result as considered other configuration.

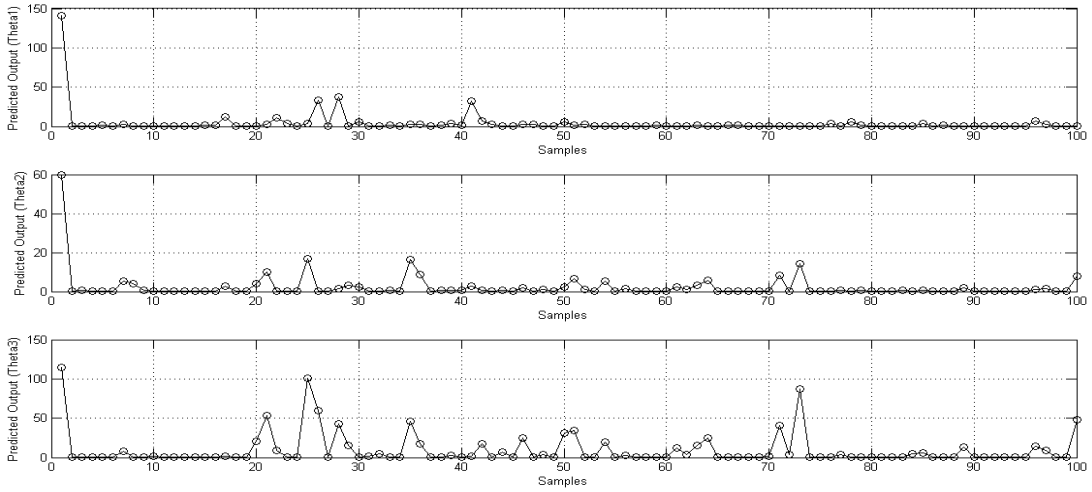

Figure 8: mean square error for joint angles using π-network model

To test the stability of the MLP two other models i.e. PPN and Pi-network models have been studied. Figure 5 (a), (b) and (c) shows the mean square error of joint angles using PPN network which gives relative poor result as compared to MLP model and in case of Pi-network the obtained result is also poor to compare with MLP as shown in figure 6 (a), (b) and (c).

6. CONCLUSION

ANN is based on the input/output data pairs to determine the structure and parameters of the model. Moreover, they can always be updated to obtain better results by presenting new training examples as new data become available. This

artificial neural network based joint angles prediction models can be a useful to estimate the position of the manipulator accurately. The results with the ANNs are better than for the other model mentioned earlier. The computation using an ANN is particularly useful where shorter calculation times are required, such as in this problem. In the future, data obtained from sensors will be able to be fed directly to the ANN, to carry out these calculations.From the present study, it is observed that the MLP gives better results for inverse kinematics problem considering the average percentage error as performance index.

REFERENCES

1. **Koker R et al.,** *"Study of neural network based inverse kinematics solution for a three-joint robot",* Robotics and Autonomous Systems (2004), 49, pp. 227–234.
2. **Shital S, Chiddarwar N and Ramesh B., 2010.** *"Comparison of RBF and MLP neural networks to solve inverse kinematic problem for 6r serial robot by a fusion approach",* Engineering Applications of Artificial Intelligence, 23 pp.1083–1092.
3. **Koker R., 2005.** *"Reliability-based approach to the inverse kinematics solution of robots using elman's networks",* Engineering Applications of Artificial Intelligence 18 pp. 685–693.
4. **Husty M L, Pfurner M and Schrocker H P., 2007.** *" A new and efficient algorithm for the inverse kinematics of a general serial 6r manipulator",* Mechanism and Machine Theory 42 pp. 66-81
5. **Mayorga R V and Sanongboon P., 2005.** *"Inverse kinematics and geometrically bounded singularities prevention of redundant manipulators: an artificial neural network approach",* Robotics and Autonomous Systems 53 pp. 164–176.
6. **Xie J et al., 2007.** *"Inverse kinematics problem for 6-dof space manipulator based on the theory of screws",* Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetic December 15 -18, 2007, Sanya, China.
7. **Hasan A et al., 2006.** *"An adaptive-learning algorithm to solve the inverse kinematics problem of a 6 D.O.F serial robot manipulator",* Advances in Engineering Software 37 pp. 432–438.
8. **Mitsi S, Bouzakis K D and Mansour G., 1995.** *"Optimization of robot link motion in inverse kinematic solution considering collision avoidance and joint limit",* Mech. Much, Theory, 30 (5) pp. 653-663.
9. **Karlik B and Aydin., 2000.** *"An improved approach to the solution of inverse kinematics problems for robot manipulators",* Engineering Applications of Artificial Intelligence 13 pp. 159-164.
10. **S. Morris and A. Mansor., 1997.** *"Artificial neural network for finding inverse kinematics of robot manipulator using look up table",* Robotica, 15, pp. 617 – 625.
11. **Hasan A et al., 2010.** *"Artificial neural network-based kinematics jacobian solution for serial manipulator is passing through singular configurations",* Advances in Engineering Software, 41 pp. 359–367.
12. **Olaru A et al., 2011.** *"Assisted research and optimization of the proper neural network solving the inverse kinematics problem",* Proceedings of 2011 International Conference on Optimization of the Robots and Manipulators.
13. **Alavandar, S and Nigam, M. J., 2008.** *"Neuro-fuzzy based approach for inverse kinematics solution of industrial robot manipulators",* Int. J. of Computers, Communications & Control, 3, pp. 224-234.