Inverse Kinematics Solution of Robot Manipulator End-Effector Position Using Multi-Neural Networks

Dr. Firas A. Raheem
Control and Systems Control Engineering Department, University of Technology /Baghdad
Email: dr.firas7010@yahoo.com
Dr. Azad R. Kareem
Control and Systems Control Engineering Department, University of Technology /Baghdad
Email: drazadnarj@hotmail.com
Dr. Amjad J. Humaidi
Control and Systems Control Engineering Department, University of Technology /Baghdad
Email: aaaacontrol2010@yahoo.com

Received on:28/3/2013 & Accepted on: 6/4/2014

ABSTRACT

This paper proposes multi-neural networks structure for solving the inverse kinematic problem of the robot manipulator end-effector position. It offers an opportunity to reduce substantially the error of the solution. This error frequently arises when only one neural network is used. In this structure, each neural network is a multilayer perceptron (MLP) trained by the back propagation algorithm. The proposed approach verified by including it within an overall Cartesian trajectory planning system. This structure could produce the robot joint variables that are not included in the training data with an average error ±0.06º, and ±0.15º, ±0.05º for joint angles θ_1, θ_2 and θ_3 respectively. From the simulation results, the proposed structure of multi-neural network has superior performance for modeling the complex robot kinematics.
Keywords: Robot Manipulator, Inverse Kinematics. Neural Networks.

INTRODUCTION

There has been much interest in the last years in applying neural network algorithms to research topics in the field of robotics. One attractive area of application is the calculation of the inverse kinematics solution for robot manipulators, which is a difficult theoretical problem in robotics due to the nonlinearity of the mapping between the joint space and Cartesian space and also due to the multiple solutions. Analytical solutions can be found only for simple robot configurations, and even in such cases the solution is a very complex process. There are several published researches applying neural networks to both the forward and inverse kinematics problem of planar manipulators and 3D manipulators [1]. Researches referred in [2-11] deal with giving different approaches to design or to learn different types of the neural networks to solve the problem under study. Two neural network structures are suggested by [2]; one for finding the inverse kinematics of the position of the robot end-effector and the other to find the inverse kinematics of the orientation of the end-effector. From the graphic analysis [2], the error gives approximately 3.6° for the position inverse kinematics and 8.5° for the orientation inverse kinematics. A method which suggested in [3] depends on the final matrix structure of the forward kinematics yields approximately the same errors as that indicated in [2]. Reference [4] has suggested three-layer partially recurrent neural network to perform trajectory planning and to solve the inverse kinematics and dynamics problems in a single processing stage. A non-algorithmic method is presented for the solution to the inverse kinematics problem of a robot [5]. This method is robot independent and involves a hybrid

https://doi.org/10.30684/etj.34.7A.9
2412-0758/University of Technology-Iraq, Baghdad, Iraq

approach, whereby a neural solution is augmented with an iterative procedure which provides the final solution within some specified tolerance. A neural network is employed to analyze the inverse kinematics of PUMA 560 type robot. The neural network is designed to find exact kinematics of the robot. The neural network is a feed-forward neural network trained with different types of learning algorithm for designing exact inverse model of the robot [6]. A study on generalization ability of neural network for manipulator inverse kinematics including the determination of optimal unit number in the hidden layer using three layered neural network is the interest subject of [7]. The work in [8] presented the performance comparison of two artificial neural networks; radial basis function and multi-layered with back propagation, both trained to learn data obtained from the kinematics model of a robotic arm. Solving the inverse kinematics using learning method of neural network represents the relations of both positions and velocities from the task space coordinate to the joint space coordinate simultaneously has been performed in [9]. While dividing one neural network into several neural networks has been presented in [10]; the first network represent the main network and the others are a secondary networks to estimates the possible errors. The use of multiple cooperating networks for the overall modeling of inverse kinematics was explored in [11], where radial basis function (RBF) neural networks were chosen. Training multiple neural networks for a specified path trajectory could minimize maximum position error to a 7°.

Reis Robot– RV12L

The inverse kinematics of the Reis Robot RV12L shown in Fig. (1) is not different from any 6 DOF robot manipulators. In this type of robot we can divide the 6 angles into two groups; the first comprises 3 angles for the position and the second has 3 angles for the orientation of the robot end effector. Therefore, the schematic shown in Fig. (2), represents the usual first 3 DOF of the robot end-effector position. Generally, there are two major inverse kinematic techniques [10]:

- Explicit, with exact solution (as for the 3 link manipulator);
- Iterative, for use when an infinite number of solutions exist.

Problems that one may encounter when performing inverse kinematics with these methods are:

- Both methods require a computer capable of mathematical calculations;
- The methods do not adapt to compensate for damage, calibration errors;
- Iterative solutions may be slow;
- Solutions are valid only for a specific robot.

Such reasons made designing neural network for solving this problems is very important. After modeling the forward kinematics according [12] and obtaining the final transformation matrix, the process of minimizing the kinematic equations to a position set and orientation matrix will result the following robot end-effector equations:

$$\begin{aligned} P_x &= cos\theta_1[L_2cos\theta_2 + L_3cos(\theta_2+\theta_3)] \\ P_y &= sin\theta_1[L_2cos\theta_2 + L_3cos(\theta_2+\theta_3)] \\ P_z &= L_1 + L_2sin\theta_2 + L_3sin(\theta_2+\theta_3) \end{aligned} \quad (1)$$

Where P_x, P_y and P_z represent the position point of the robot end-effector. L_1 can be eliminated by transferring the point of analysis for the first link in the center of connection between L_1 and L_2. Usual analytical solution given in [12] for such type of robots leads the following equations of θ_1, θ_2 and θ_3 for left, right robot arm and for elbow up, elbow down robot arm:

$$\theta_1 = atan2(P_x, P_y) \text{ and } \theta_1 = \pi + atan2(P_x, P_y) \quad (2)$$

$$D = \frac{P_x^2 + P_y^2 + (P_z - L_1)^2 - L_2^2 - L_3^2}{2L_2L_3} \quad (3)$$

$\theta_3 = atan2\left(D, \sqrt{1-D^2}\right)$ and $\theta_3 = atan2\left(D, -\sqrt{1-D^2}\right)$ – for elbow down and elbow up robot arm respectively.

The two solutions of θ_2, which depends on θ_3, will be given by;

$$\theta_2 = atan2(r, s) - atan2(L_2 + L_3 cos\theta_3, L_3 sin\theta_3)$$

Where the radius $r = \sqrt{P_x^2 + P_y^2}$ and$s = P_z - L_1$.

From the manual of Reis Robot RV12L: $L_1 = 0.783$ m, $L_2 = 0.7025$ m, $L_3 = 0.651$ m and the range of angles is $(-165°$ to $165°)$ for θ_1, $(15°$ to $165°)$ for θ_2 and $(-135°$ to $135°)$ for θ_3.

Methodology and Design

For a successful neural network design, there are important problems needs to be solved:

- How to find a suitable multi-neural networks structure;
- How to determine an appropriate representation of input/output data;
- How to create a well-structured learning data set, and
- How to train the network successfully.

It is worthy to notify that a unique structure of neural network could not cope with entire robot work-space. Therefore, it is suggested to divide the workspace into eight sub-workspaces and then assign one neural structure for each subspace. This would permit to enlarge data base of the full work-space. Therefore, one may expect to reach near exact learning process and decrease the error to a large extent.

For $(x - y)$ plane, the data file can be generated by representing θ_1 as circles. Similarly, the data needed for θ_2 and θ_3 can be represented as circles in $(r - z)$ plane. Increasing the number of circles means increasing the accuracy of training of every neural network in the entire neural network scheme. This will give an effective multi-neural networks design trained by the data file and would allow θ_1 to be divided into four parts according the following ranges: (0° to 90°), (90° to 165°), (0° to -90°) and (-90° to - 165°). On the other hand, the data files for θ_2 and θ_3 would be divided according to the four inverse kinematics solutions of the end-effector position (left arm with elbow up and elbow down) and (right arm with elbow up and elbow down). The final structure will consist of eight large trained neural networks supplied with a unit of addressing control; responsible for choosing the proper set of angles for every robot configuration. Every neural network in the structure has simple and fixed construction with well-training; and this is one of the main advantages of the suggested design. In order to make the results obtained from every neural network very closed to the real and desired results, the input will not rely on the point (P_x, P_y, P_z) but also would depends on another component derived from the point. These components will work as index for the neural network and would increase the speed of training for every neural network. The suggested structure for each sector is illustrated in Fig.(3). It is evident from the figure that the inputs of the neural networks of every joint variable (θ_1, θ_2 and θ_3) will be different from each other. The distances P and r indicated in the figure are given by $P = \sqrt{P_x^2 + P_y^2 + P_z^2}$ and $r = \sqrt{P_x^2 + P_y^2}$.

The entire multi-neural networks system is depicted in Fig. (4). Solution of θ_1 includes four neural networks with the control addressing unit responsible for choosing the proper neural network as shown in Fig. (5). The inputs of θ_1 are P_x, P_y and r. Fig. (6) shows the internal structure of the four neural networks; two networks are for θ_2 and the other for θ_3 which trains only for range of joint limit angles of Reis Robot. The inputs of θ_2 are r, P_z, θ_3 and P, where the solution of $\theta_3$is fed as input to find the solution of θ_2.

Fig. (7) exemplifies the proposed neural network internal structure to solve the inverse kinematics of θ_3 specifically. By trial and error design approach, the structure consists of four layers: input layer, two hidden layers and one output layer. Four inputs r, P_z, D, and P are used in the input layer for finding θ_3, twelve neurons in the first hidden layer, six neurons in the second hidden layer and finally one neuron in the output layer. The hyperbolic tangent function is used as the activation function in the hidden layers, while the linear activation function is employed in the output layer.

Results and Analysis

All the neural networks in the structure are trained by a relatively large data file which covers the robot workspace using the back-propagation algorithm. The performance index of all these neural networks is the mean square error that used to indicate the convergence error during the training process. Figure (8) shows the mean square error graph during training of θ_3 where the goal was 10^{-11}.

The results after training the neural networks of the θ_1 and θ_3 are better than those with θ_2; because for every value of θ_2 there is a corresponding range of θ_3 and there is dependency between the errors of θ_2 and θ_3. Figure (9) and (10) shows the after-training errors of θ_2 and θ_1, respectively.

The main difficulties with the neural training are the singular robot configurations which lead to singular forward kinematic matrix. From the analysis of the robot, the point $(0, 0, P_z)$ of end-effector position is considered the main singular configuration for θ_1, since there is no offset shoulder in Reis Robot RV12L to prevent this singular configuration. At this case the value of θ_1 will not affect the robot position, while θ_2 and θ_3 will give the desired robot configuration. The table below shows joint angles based on classical geometrical solution and those obtained using suggested multi-neural networks structure.

Verification Model

A Cartesian space trajectory planning system based on cubic polynomial Bezier curve design has been implemented using Matlab-Simulink software. In Cartesian space schemes, the path shape is described in terms of functions which compute Cartesian positions of the Robot End-Effector as a function of time. Thus the trajectory generation is performed in Cartesian space and is converted into corresponding joint space variables at time of executing the trajectory. Bezier curves are useful to design trajectories with certain roundness properties and defined by series of control points with associated weighting factors. During the motion of the robot end effector an Inverse Kinematics process is needed to convert the points in the Cartesian path to the proper joint angles which represent the same path and realize it [13, 14]. The Bezier curve design for Cartesian trajectory planning of robot manipulator end-effector can be described by the following Equations:

$$x(t)=\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]^3 x_o+3\left(\frac{t-t_o}{t_f-t_o}\right)\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]^2 x_1+3\left(\frac{t-t_o}{t_f-t_o}\right)^2\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]x_2$$
$$+\left(\frac{t-t_o}{t_f-t_o}\right)^3 x_3$$

$$y(t)=\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]^3 y_o+3\left(\frac{t-t_o}{t_f-t_o}\right)\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]^2 y_1+3\left(\frac{t-t_o}{t_f-t_o}\right)^2\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]y_2$$
$$+\left(\frac{t-t_o}{t_f-t_o}\right)^3 y_3$$

$$z(t)=\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]^3 z_o+3\left(\frac{t-t_o}{t_f-t_o}\right)\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]^2 z_1+3\left(\frac{t-t_o}{t_f-t_o}\right)^2\left[1-\left(\frac{t-t_o}{t_f-t_o}\right)\right]z_2$$
$$+\left(\frac{t-t_o}{t_f-t_o}\right)^3 z_3$$

Where t = time parameter (second), t_o = starting time (second), t_f = final time (second). The Bezier equations $x(t)$, $y(t)$ and $z(t)$ represents the robot Cartesian trajectory between the start (x_o,

y_o, z_o) and the goal (x_3, y_3, z_3) points. Other trajectory parameters x_1, y_1, z_1, x_2, y_2 and z_2 can be calculated using the following equations:

$$x_1 = A|x_3 - x_o|, \qquad y_1 = B|y_3 - y_o|, \qquad z_1 = C|z_3 - z_o|$$

$$\theta_{xy} = \tan^{-1}\left(\frac{y_3 - y_o}{x_3 - x_o}\right) \qquad zz = \sqrt{(x_3 - x_o)^2 + (y_3 - y_o)^2}$$

$$\theta_z = \tan^{-1}\left(\frac{z_3 - z_o}{zz}\right) \qquad x_2 = D[x_3 - (x_1 \cos\theta_{xy})]$$

$$y_2 = E[y_3 - (y_1 \sin\theta_{xy})] \qquad z_2 = F[z_3 - (z_1 \sin\theta_z)]$$

The parameters (*A, B, C, D, E* and *F*) are related to the control point's calculations and have been chosen to achieve appropriate curvature of Bezier curve. Also its easy to differentiate the equations of *x*(t), *y*(t), and *z*(t) with respect to time to find the velocity and the acceleration, where $V_x(t)$ and $A_x(t)$ are *x*-component linear velocity and acceleration of Bezier trajectory, respectively;

$$Vx(t) = -3\left(1 - \frac{t - t_o}{t_f - t_o}\right)^2 \frac{x_o}{t_f - t_o} + \frac{3}{t_f - t_o}\left(1 - \frac{t - t_o}{t_f - t_o}\right)^2 x_1 - 6\frac{t - t_o}{(t_f - t_o)^2}\left(1 - \frac{t - t_o}{t_f - t_o}\right)x_1$$

$$+ 6\frac{t - t_o}{(t_f - t_o)^2}\left(1 - \frac{t - t_o}{t_f - t_o}\right)x_2 - 3\frac{(t - t_o)^2}{(t_f - t_o)^3}x_2 + 3\frac{(t - t_o)^2}{(t_f - t_o)^3}x_3$$

$$Ax(t) = 6\left(1 - \frac{t - t_o}{t_f - t_o}\right)\frac{x_o}{(t_f - t_o)^2} - \frac{12}{(t_f - t_o)^2}\left(1 - \frac{t - t_o}{t_f - t_o}\right)x_1 + 6\frac{t - t_o}{(t_f - t_o)^3}x_1$$

$$+ \frac{6}{(t_f - t_o)^2}\left(1 - \frac{t - t_o}{t_f - t_o}\right)x_2 - 12\frac{t - t_o}{(t_f - t_o)^3}x_2 + 6\frac{t - t_o}{(t_f - t_o)^3}x_3$$

The velocities and accelerations in *y* and *z*-axes are similarly computed. The desired trajectory has been simulated for t_f = 5 (second), the start and the goal points are (0.7, -0.7, 1.5) and (-0.6, 0.6, 1.6), respectively. For the comparison purposes, the overall trajectory planning system shown in Fig. 11 includes the classical inverse kinematic solution and the solution resulting from suggested approach. The simulated Bezier trajectory is shown in Fig. (12), while the produced joint variables θ_1, θ_2 and θ_3 are shown in Fig. (13). The error traces of different joint variables between the classical computation and multi-neural structure is shown in Fig. (14).

CONCLUSIONS

Multi-neural networks structure trained using back-propagation algorithm has been applied to inverse kinematic problem. As a case study Reis Robot RV12L were used for testing this model. Dividing the workspace of the robot into eight neural networks gives the ability to increase the size of the data file and, then, to increase the accuracy of the solution. However, a trade-off of training time would appear. The training process may take a long time that may reaches to 2 hours and many epochs number up to 10000. On the other hand the feed-forward mode of this model can give a very fast results comparing with classic analytical (mathematical) methods. Results showed that θ_1 and θ_3 are better than θ_2. The accuracy of θ_2 deeply depends on θ_3; since θ_3 is one of the dependent input of θ_2 neural networks. This approach was tested in a Bezier trajectory planning system. The comparison between the inverse kinematic classical solution and the proposed multi-neural network solution shows that the overall 8-neural networks structure was able to predict robot joint angles that were not included in the training data with an approximate average joint errors ±0.06º, ±0.15º, and ±0.05º for θ_1, θ_2 and θ_3, respectively.

REFERENCES:

[1] Glazkov V.P., Egorov I.V., Pchelintseva C.V. Iterational Checking for the Inverse Kinematic Solution of the Robot Manipulator Using Neural Network // Mechatronics, Automation, Control. 2005. No. 4. P. 15-17.
[2] Kozakiewicz K., Ogiso T., and Miyake N. (1991). Partitioned Neural Network Architecture for Inverse Kinematic Calculation of a 6 DOF Robot Manipulator, IEEE Conference.
[3] Bingul Z., Ertuc H.M., and Oysu C. (2005). Comparison of Inverse Kinematics Solutions Using Neural Network for 6R Robot Manipulator with Offset, IEEE Conference .
[4] Araujo A.F., and D'Arbo H. (1998). A Partially Recurrent Neural Network to Perform Trajectory Planning, Inverse Kinematics, and Inverse Dynamics, IEEE Conference.
[5] Ahmad Z., and Guez A. (1990). On the Solution to the Inverse Kinematic problem, IEEE Conference.
[6] Yildirim S., and Eski I. A PQ Artificial Neural Network Inverse Kinematic Solution for Accurate Robot Path Control. Journal of Mechanical Science and Technology. July 2006, Volume 20, Issue 7, pp. 917-928.
[7] Watanabe E., and Shimizu H.A. (1991). Study on Generalization Ability of Neural Network for Manipulator Inverse Kinematics, IEEE Conference.
[8] Yang S.S., Moghavvemi M., and Tolman J.D. (2000). Modeling of Robot Kinematics Using Two ANN Paradigms, IEEE Conference, 2000.
[9] Kuroe Y., Nakai Y., and Mori T. (1993). A New Neural Network Approach to the Inverse Kinematics Problem in Robotics, IEEE Conference.
[10] Hugh Jack, Neural Network Calculation of Inverse Kinematics, Lecture Notes, http://hughjack.com/V2/notes/engineeronadisk-1542.html.
[11] Driscoll A.J. (2000). Comparison of Neural Network Architectures for the Modeling of Robot Inverse Kinematics. IEEE Conference, 2000.
[12] Spong M.W., Hutchinson S. and Vidyasagar M. (2006). Robot Modeling and Control / Mark W. Spong, M. Vidyasagar. John Wiley&Sons. P. 478.
[13] Bazaz, S.A. and Tondu, B. (1998). "3-Cubic Spline for On-Line Cartesian Trajectory Planning of an Industrial Manipulator", IEEE Conference, 1998.
[14] Hwang, J.H., Arkin, R.C. and Kwon, D.S. "Mobile robots at your Fingertip: Bezier Curve On-line Trajectory Generation for Supervisory Control, IEEE Conference, 2003.

Table (1):

θ_1 desired	θ_1 neural	θ_2 desired	θ_2 neural	θ_3 desired	θ_3 neural
45	45.00177	-12.3842	-12.3807	100.5398	100.5399
56.3099	56.3109	-44.5798	-44.5695	71.3369	71.3367
123.6901	123.6892	1.7165	1.708	52.6273	52.6275
29.0546	29.0555	-16.1804	-16.1748	73.1458	73.1459
99.8226	99.8212	13.8448	13.8542	2.3804	2.3573
100	99.9977	14.6664	14.7171	0.693	0.60354559
129.9976	129.9969	105.0035	105.0023	134.9986	134.9977

Figure (1) Reis Robot RV12L

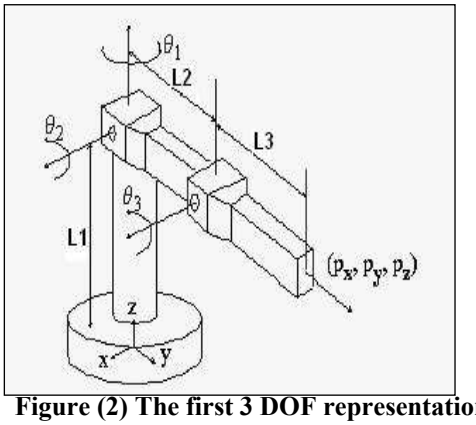

Figure (2) The first 3 DOF representation of the Reis Robot

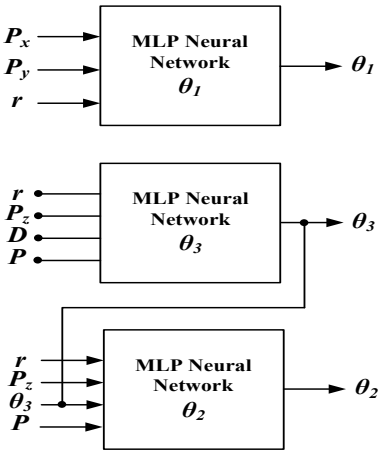

Figure (3) Inputs design of the neural networks of θ_1, θ_2 and θ_3

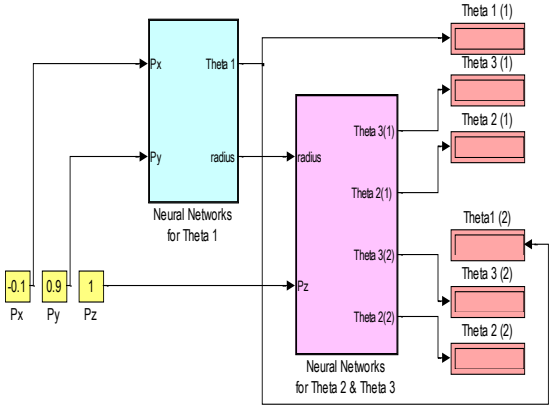

Figure (4) Overall system for multi-neural inverse kinematics

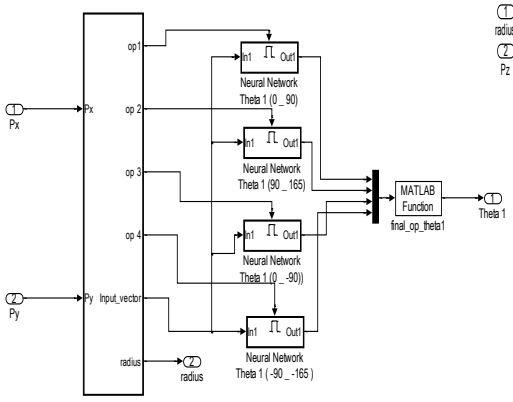

Figure (5) Four neural-networks for inverse kinematics of θ_1

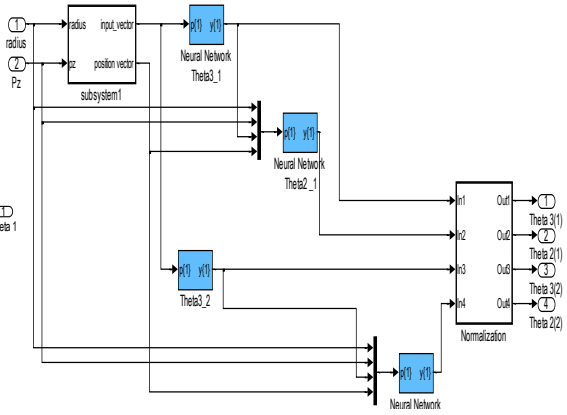

Figure (6) Four neural-networks for inverse kinematics of θ_2 and θ_3

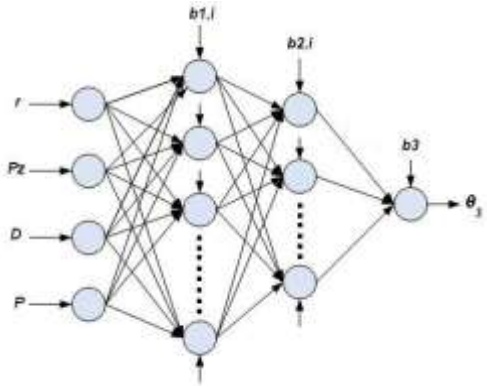

Figure (7) Neural network structure for inverse kinematics of θ_3

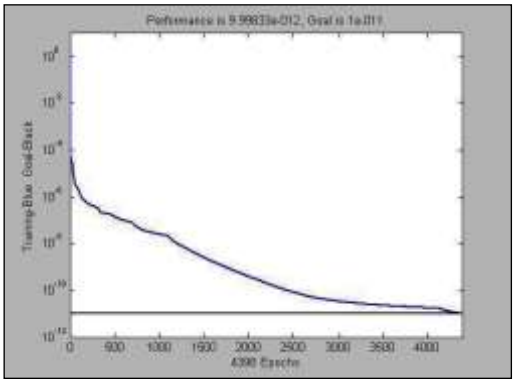

Figure (8) Mean square error graph during training θ_3

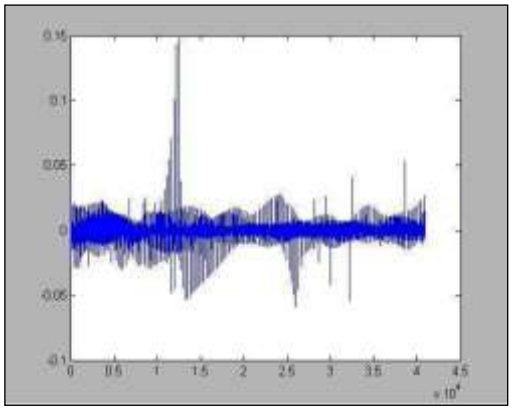

Figure (9) Error of θ_2 (degree) after training

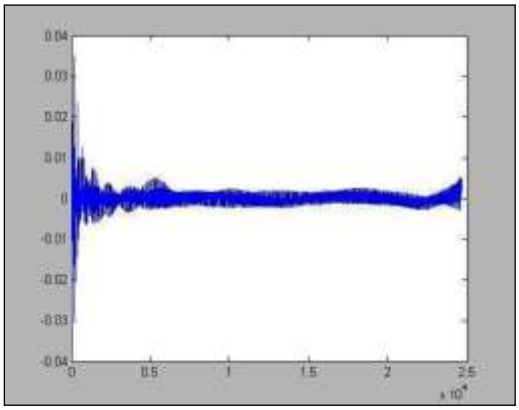

Figure (10) Error of θ_1 (degree) after training

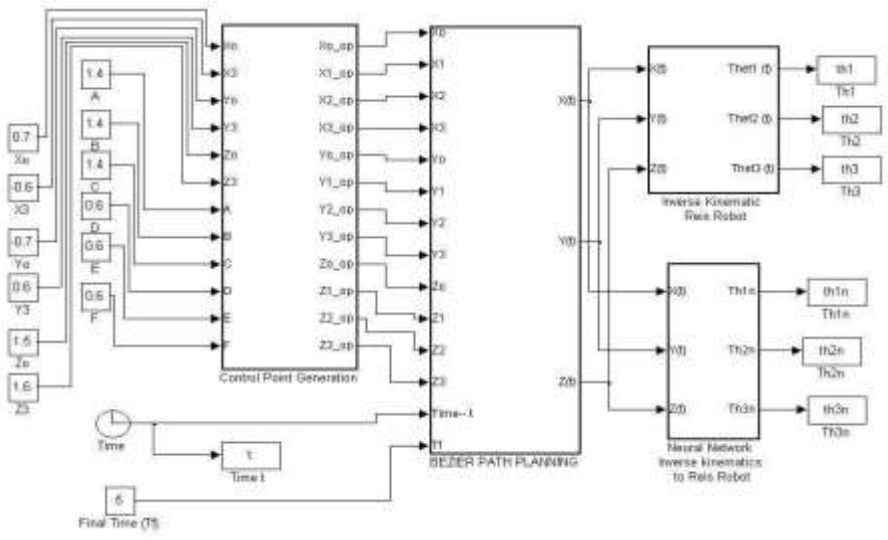

Figure (11) Overall planning system for testing the multi-neural solution of the inverse kinematics

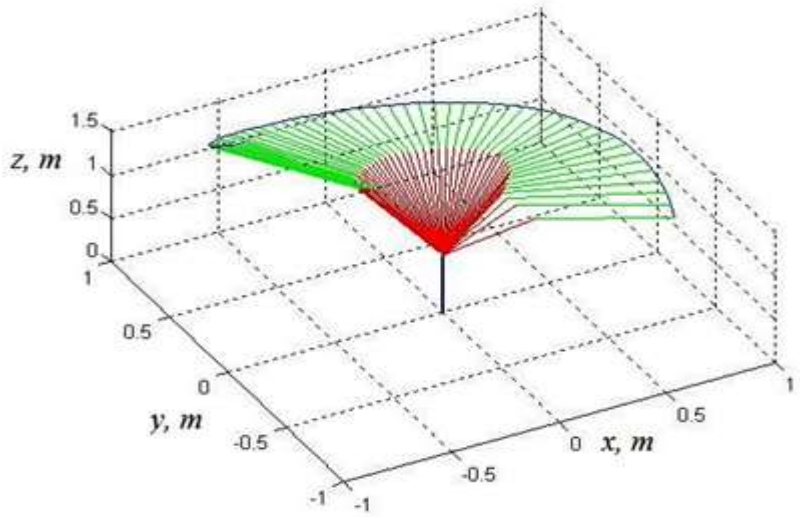

Figure (12) The resulted Bezier Robot Trajectory

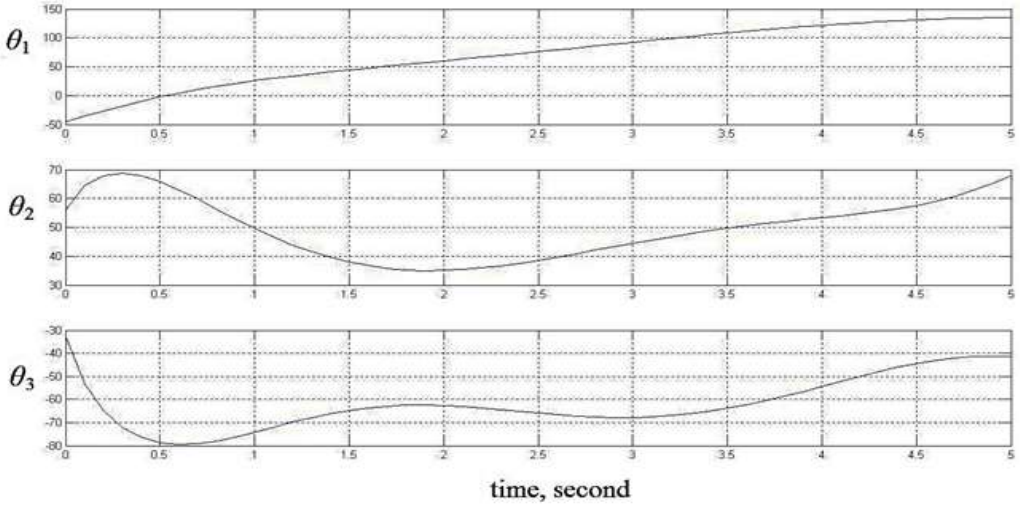

Figure (13) Joint variables θ_1, θ_2, and θ_3 (degrees)

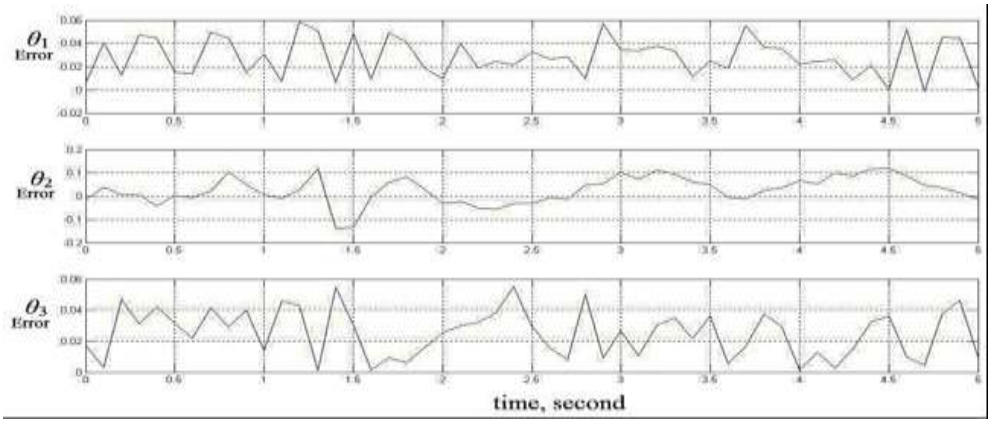

Figure (14) Computation Error of θ_1, θ_2, and θ_3 (degrees)