Paper: ASAT-15-087-CT

15th International Conference on
AEROSPACE SCIENCES & AVIATION TECHNOLOGY,
ASAT - 15 – May 28 - 30, 2013, Email: asat@mtc.edu.eg ,
Military Technical College, Kobry Elkobbah, Cairo, Egypt,
Tel: +(202) 24025292 –24036138, Fax: +(202) 22621908

Neural-Networks-Based Inverse Kinematics for a Robotic Manipulator

M. A. Fouz[*], A. M. Bayoumy[†] , Sohair F. Rezeka[‡]

Abstract: The solution of inverse kinematics problem of robotic manipulator is a fundamental problem in robot control. This paper involves the study of forward and inverse kinematics of a robotic manipulator platform with revolute joints. The (Denavit-Hartenberg) kinematic model of the prescribed manipulator is presented to robot links and joints. In addition inverse kinematics solution has been provided using geometric approach. This solution is used to develop a dataset used to train three artificial feed forward neural networks (ANN). Each network is used to calculate one joint variable using position and orientation of the end effector as an input. Experimental results have shown a good mapping over the working area of the robot with smaller RMS error than that reported in the cited literature.

Keywords: Robotics, Denavit-Hartenberg, Inverse kinematics, Artificial Neural Network.

Nomenclature

θ_1:	Base angle
θ_2:	Shoulder angle
θ_3:	Elbow angle
θ_4:	Wrist-pitch angle
φ:	Wrist pitch angle
ANN:	Artificial neural network
D-H:	Denavit and Hartenberg notation
DoF:	Degree of freedom
Exp( ):	Exponential function
Hyp-Tan-Sig:	Hyperbolic tangent sigmoid transfer function
RMS:	Root mean square
${}_0T^5$:	Homogeneous transformation matrix that transform from frame No.5 to frame No.0
MSE:	Mean square error

Introduction

The robot servo control system requires the reference inputs to be in joint coordinates whereas the task is generally stated in terms of Cartesian coordinate system. For efficient control of the position and orientation of the robot end-effector to reach its object, the understanding of kinematic relationship between the joint coordinate system and the Cartesian coordinate system is essential. The kinematics problem of robots is generally categorized into forward

[*] GTA in AASTMT, Cairo, Egypt, (B.Sc.), moustafafouz@hotmail.com .
[†] Egyptian Armed Forces, Egypt, dramgad@mep-ls.com .
[‡] Professor in Alexandria University, Alexandria, Egypt (currently on leave at AASTMT), srezeka@yahoo.com .

kinematics and inverse kinematics problems. For successful task execution in various industrial applications like spot welding, precision assembly, packaging, etc., the path planning is done at the background or in offline mode in order to determine the Cartesian path for robot. The execution of this Cartesian path demands for conversion of Cartesian coordinates into joint angle coordinates using inverse kinematics relations. Hence, inverse kinematics computations for a serial robot are elemental for design, analysis of workspace for path planning, trajectory planning and control and offline programming of robots. This conversion is done by mapping Cartesian space of robot into its joint space by using inverse kinematics relations. This mapping process is nonlinear due to association of nonlinear trigonometric equations and becomes more complex for the robot with complex geometry and multi-degree of freedom. Moreover, the associated problems like the coupled nature of position and orientation kinematics of the robot, existence of multiple solutions and the presence of singularities add to the computational complexities.

Kinematical analysis based solutions are very vital when one wants to perform modeling of robotic arm. It turns out to be a difficult task to find the solution through inverse kinematics with increase in DoF (Degree of Freedom) of robot. The conventional methods used for calculating inverse kinematics of any robot manipulator are: geometric [1, 2], algebraic [3] [4] [5] and iterative approaches[6]. While algebraic methods cannot promise closed form solutions, geometric methods must be able to produce closed form solutions for the first three joints of the manipulator. On the other hand, the iterative methods result in a single solution only, and that solution also depends on the starting point. To solve the inverse kinematics problem for three different cases of a 3-degrees-of freedom (DoF) manipulator in three dimensional spaces, a solution was proposed in [7] using feed-forward neural networks. This introduces the fault-tolerant and high-speed advantages of neural networks to the inverse kinematics problem. Hierarchical control technique based on the establishment of a non-linear mapping between Cartesian and joint coordinates using fuzzy logic was proposed in [8] in order to direct each individual joint and control a robotic manipulator. Commercial Microbot with 3DoF was utilized to evaluate the proposed method. A novel modular neural network system to overcome the discontinuity of the inverse kinematics function was proposed in [9] and it consists of a number of expert neural networks. Neural network based three-joint robotic manipulator simulation software was developed in [10] for inverse kinematics solution of a robotic manipulator. Then a designed neural network was used to solve the inverse kinematics problem. An Artificial Neural Network (ANN) based on Bees Algorithm using back propagation algorithm was applied in [11] to solve inverse kinematics problems of industrial robot manipulator. That in turns used to train multi-layer perceptron neural networks in [12] to model the inverse kinematics of an articulated robot manipulator arm. An Artificial Neural Network (ANN) based approach for fast inverse kinematics computation and effective geometrically bounded singularities prevention of redundant manipulators was presented in [13]. A fusion approach to determine inverse kinematics solutions of serial robot is presented in [14]. The presented solution makes use of radial basis function neural network for prediction incremental joint angles. An adaptive learning strategy using artificial neural network has been presented in [15].

In this study, the inverse kinematics problem is split to two simple ones. The solution using three feed forward neural networks is presented. The paper is organized into four main sections: the following section is dedicated to identify the robot arm platform used in this work; followed by a section that in concern with explanation of forward and inverse kinematics of the specified robotic platform. Final section discusses the implementation of the proposed neural network.

Description of the “ED-7220C” Robotic Manipulator

The ED-7220C robotic manipulator is presented in Figure 1. It is based on a five joint system which is popular in industry. Therefore, the experiments using ED-7220C can be directly applied in the real business needs. Each robot joint is driven by a DC servo motor with a rotary encoder built-in. The overload sensing and controls are managed by a microprocessor.
The ED-7220C is designed such that the main driving mechanism, including the timing belt, is exposed to the user to offer the visual observation. Also, the robot is designed to maintain the holding positions of the grip fingers even though the elbow and shoulder are in motion[16].

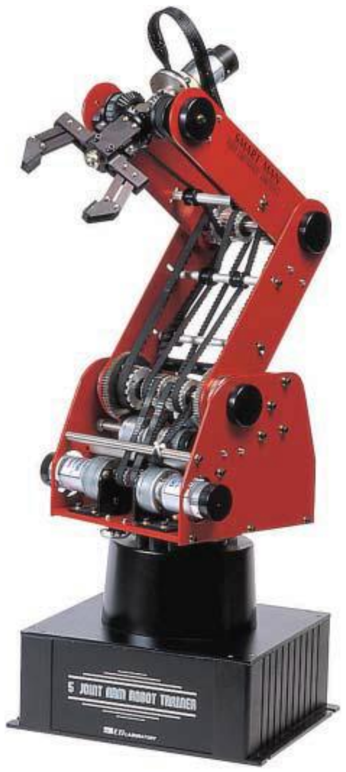


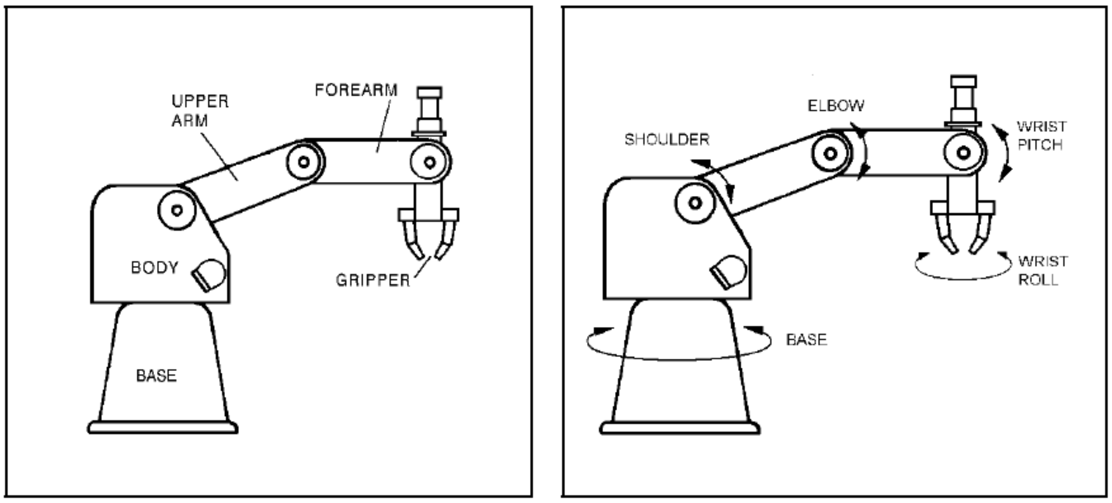
Figure 1 ED-7220C robotic manipulator

ED-7220C Structure

All joints are revolute as shown in Figure 2, and with an attached gripper it has six degree of freedom. Each joint is restricted by the mechanical rotation its limits are shown below.

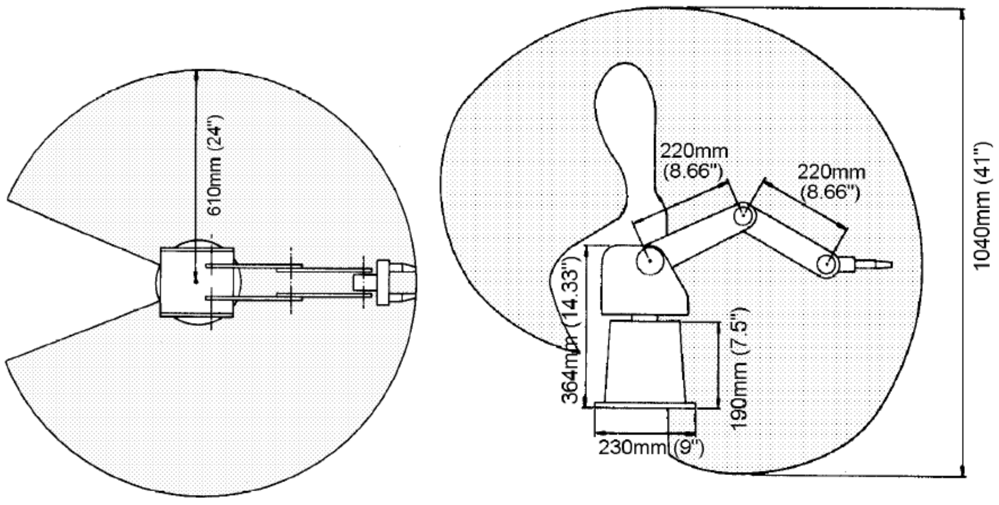
Joint Limits (all angles are measured from positive X-axis direction):
Axis 1: Base Rotation: 310°
Axis 2: Shoulder Rotation: + 32° / 123°
Axis 3: Elbow Rotation: + 190° / -75°
Axis 4: Wrist Pitch: ± 125°
Axis 5: Wrist Roll Unlimited (electrically 570°)

Maximum Gripper Opening: 65 mm without rubber pads 55 mm (2.6") with rubber pads. The length of the links and the degree of rotation of the joints determine the robot’s work envelope. Figure 3 shows the dimensions and reach of the ED-7220C. The base of the robot is normally fixed to a stationary work surface. It may, however, be attached to a slide base, resulting in an extended working range.



(a) Links (b) Joints

Figure 2 Robot arm links and joints



Top view Side view

Figure 3 The working envelop

Kinematic Model of ED-7220C

Forward Kinematics

Kinematic modeling of the robot arm can be made simple by using Denavit and Hartenberg (D-H) notation. The frame assignment and joint-link parameters for ED-7220C robot manipulator are shown in Figure 4 and Figure 5, respectively.

Table 1 states the corresponding D-H parameters for the robot.

Table 1 Symbolic D-H parameters for ED-7220C

Joint	**ϴ**	**D**	**a**	**α**
1	Θ_1	d_1	a_1	$\pi/2$
2	Θ_2	0	a_2	0
3	Θ_3	0	a_3	0
4	Θ_4	0	0	$\pi/2$
5	Θ_5	d_5	0	0

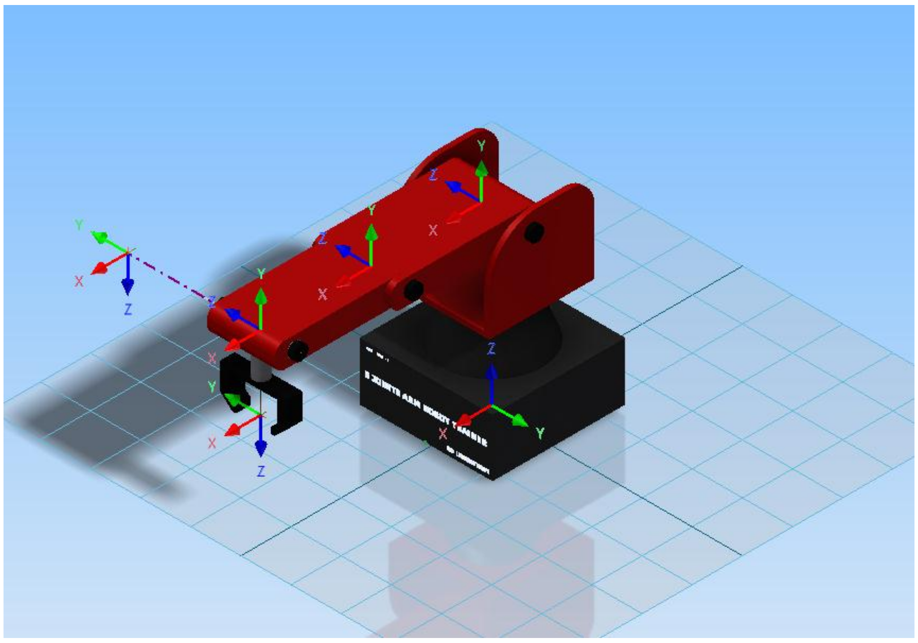


Figure 4 Representation of the robot arm showing joint frames

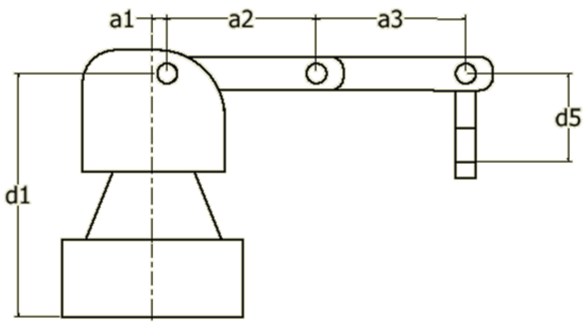


Figure 5 Robot links notation

Once the D-H coordinate system has been established for each link as shown in Figure 4 and Figure 5, a homogeneous transformation matrix can be developed.

$$T_0^5 = T_0^1 T_1^2 T_2^3 T_3^4 T_4^5 \tag{1}$$

Substituting for (d_1, a_1, a_2, a_3, d_5) by (360, 20, 220, 220, 150), this may lead to:

$$T_0^5 = \begin{bmatrix} R_{11} & R_{12} & R_{13} & X \\ R_{21} & R_{22} & R_{23} & Y \\ R_{31} & R_{32} & R_{33} & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where

$$R_{11} = Sin(\theta_1)\times Sin(\theta_5) + Cos(\theta_2+\theta_3+\theta_4)\times Cos(\theta_1)\times Cos(\theta_5)$$
$$R_{12} = Cos(\theta_5)\times Sin(\theta_1) - Cos(\theta_2+\theta_3+\theta_4)\times Cos(\theta_1)\times Sin(\theta_5)$$
$$R_{13} = Sin(\theta_2+\theta_3+\theta_4)\times Cos(\theta_1)$$
$$R_{21} = Cos(\theta_2+\theta_3+\theta_4)\times Cos(\theta_5)\times Sin(\theta_1) - Cos(\theta_1)\times Sin(\theta_5)$$
$$R_{22} = -Cos(\theta_1)\times Cos(\theta_5) - Cos(\theta_2+\theta_3+\theta_4)\times Sin(\theta_1)\times Sin(\theta_5)$$
$$R_{23} = Sin(\theta_2+\theta_3+\theta_4)\times Sin(\theta_1)$$
$$R_{31} = Sin(\theta_2+\theta_3+\theta_4)\times Cos(\theta_5) \tag{3}$$
$$R_{32} = -Sin(\theta_2+\theta_3+\theta_4)\times Sin(\theta_5)$$
$$R_{33} = -Cos(\theta_2+\theta_3+\theta_4)$$
$$X = Cos(\theta_1)\times(150\times Sin(\theta_2+\theta_3+\theta_4) + 220\times Cos(\theta_2+\theta_3) + 220\times Cos(\theta_2) + 20)$$
$$Y = Sin(\theta_1)\times(150\times Sin(\theta_2+\theta_3+\theta_4) + 220\times Cos(\theta_2+\theta_3) + 220\times Cos(\theta_2) + 20)$$
$$Z = 220\times Sin(\theta_2+\theta_3) - 150\times Cos(\theta_2+\theta_3+\theta_4) + 220\times Sin(\theta_2) + 360$$

Hereby; one can use these equations to locate the end-effector position and orientation.

Inverse kinematics

To solve for the inverse kinematics, equations have to be found for the five joint variables in terms of the position (X, Y, and Z), and the orientation (roll (φ), pitch (θ)), of the end-effector. From Figure 6, the value for the first joint is:

$$\theta_1 = Tan^{-1}\left(\frac{Y}{X}\right) \tag{4}$$

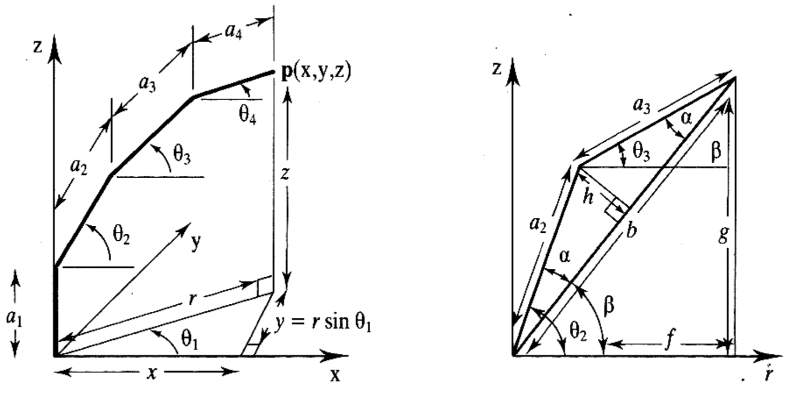


Figure 6 Kinematical model with elbow triangle

If the manipulator arm is placed with all links in a vertical position, roll of the end-effector represents a rotation around the Z axis, as does rotation around the waist. Thus, the value of the last joint angle is:

$$\theta_5 = \phi - \theta_1 \tag{5}$$

Similarly, pitch can be considered to be a rotation about the X axis, giving:

$$\theta_4 = \theta \tag{6}$$

If we consider the triangle formed by the second and third links as shown in

Figure 6, the following relationships can be obtained:

$$\begin{aligned} r &= \frac{X}{Cos\theta_1} \quad \text{if } \theta_1 \neq \frac{\pi}{2} \\ r &= \frac{Y}{Sin\theta_1} \quad \text{if } \theta_1 \neq 0 \end{aligned} \tag{7}$$

$$\begin{aligned} f &= r - a_1 - a_4 \times Cos\theta_4 \\ g &= Z - d_1 - a_4 \times Sin\theta_4 \\ b &= \sqrt{g^2 + f^2} \\ h &= (a_2)^2 - \left(\frac{b}{2}\right)^2 \\ \alpha &= Tan^{-1}\left[\frac{h}{b/2}\right] \\ \beta &= Tan^{-1}\left[\frac{g}{f}\right] \end{aligned} \tag{8}$$

$$\begin{aligned} \theta_2 &= \beta + \alpha \quad \text{for elbow up configuration} \\ \theta_2 &= \beta - \alpha \quad \text{for elbow down configuration} \end{aligned} \tag{9}$$

$$\begin{aligned} \theta_3 &= \beta - \alpha \quad \text{for elbow up configuration} \\ \theta_3 &= \beta + \alpha \quad \text{for elbow down configuration} \end{aligned} \tag{10}$$

To calculate the inverse kinematics for certain position, there will be more than one configuration. Assuming that the Cartesian coordinates of desired position are (376, 0, 718)' with end effector pitch angle '$\theta_4 = 0^o$', this leads to two possible configurations as shown in Figure 7. In upper elbow configuration, the solution is (0, 40, and 80) ^{o} for ($\theta_1, \theta_2, \theta_3$) respectively, where for lower configuration, the solution is (0, 80, and 40) ^{o} for ($\theta_1, \theta_2, \theta_3$) respectively.

Proposed Neural Networks for the Inverse Kinematics of ED-7220C

To simplify the problem of creating artificial neural network (ANN) that transform from Cartesian space into joint space (performing inverse kinematics of ED-7220C), an attempt is done to split the big problem into two small problems. First, consider the robot to be planar, as shown in Figure 8(a), and get shoulder angle (θ_2) and elbow angel (θ_3), then, getting base angle (θ_1) would be easy, as presented in Figure 8(b). That means there are three ANNs each used to produce a specific angle of the robot.

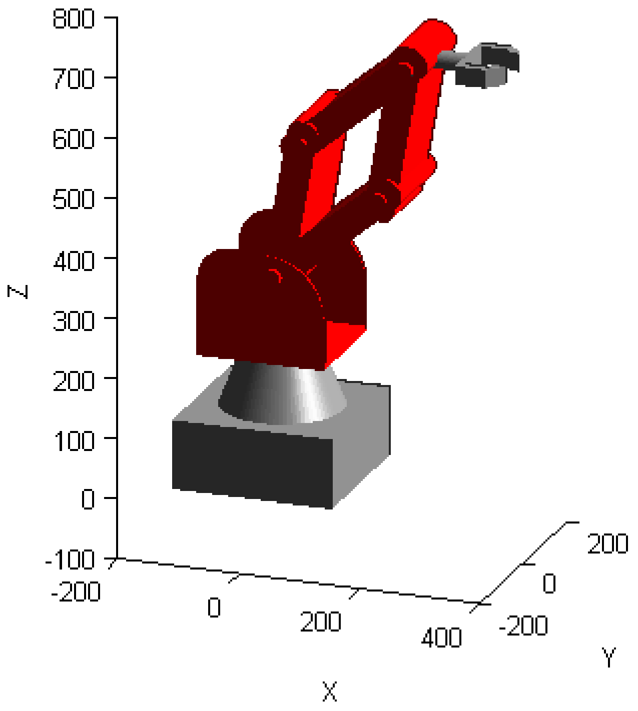


Figure 7 Representation of two possible configurations

Regarding first two ANNs that generates ($\theta_{2,}$ θ_3), the inputs of the ANNs will be the (R, Z-axis) coordinates and the wrist pitch angle (θ_4), so there will be totally three inputs to the ANN and one output. The last ANN is responsible of generating the base angle (θ_1), and this network has only two inputs (X, Y-axis) coordinates.

Constructing Training Data Set

A Matlab® code has been developed to cover the most volume of the robotic working space with resolution of (2 mm) for (R, Z-axis) as shown in Figure 8(a), and 5 degrees for (θ_4). The total available points after rejecting points that violates robotic arm constrains, reached (1 231 401) point. $\theta_{2,}$ θ_3 is then generated by feeding the recorded points to the inverse kinematics equations described above (refer section 0). Then I/O pairs are created between the Cartesian space (I/Ps) and the joints space. These pairs are stored in a separate data file in order to be called in the training process.

The same procedure is done to generate a data set that contains training data of the third ANN that produces (θ_1). Where I/Ps are X, Y-axis with resolution of (5 mm) and the O/P is θ_1. The total number of pairs is (45 167) pair. These pairs have been saved also in separate file.

As discussed before, in section 0 above, there are only two possible configurations to reach certain point in allowable working space, the two possible configurations are elbow up or elbow down configuration. Only elbow up configuration is used in creating the training set to train the networks. That is because: this type of configuration is widely popular in manipulation process.

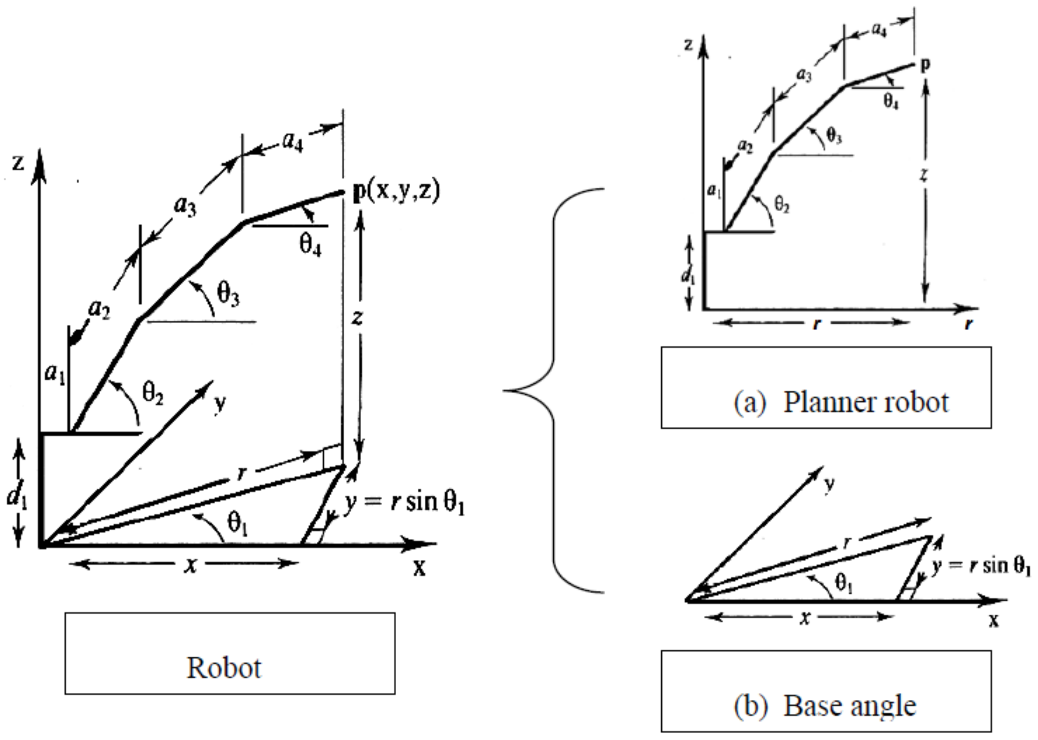


Figure 8 Inverse kinematics problem is divided into two smaller problems

ANN Construction

As mentioned before, three ANNs have been constructed to calculate (θ_2, θ_3, and θ_1). The first two ANN are responsible for generating (θ_2, θ_3) are the same in structure. Each has one input layer, two hidden layers, and one output layer. Regarding the input layer, it is responsible for preparing the inputs to be fed to first hidden layer. This can be done by passing the inputs to a preprocess function, this function normalize inputs to fall in range of (-1, 1) this is useful in training the network. The number of neurons in the first and second hidden layers will be optimized during training based on the minimum square error. The output layer contains one neuron in order to generate single output as shown in Figure 9 and Figure 10.

As shown in Figure 9, the neurons in hidden layers have hyperbolic tangent sigmoid transfer function, while the output layer has only one neuron with linear transfer function.
Regarding the third ANN (the ANN that produces the values of θ_1), it has two hidden layer and the number of neurons will be optimized during training based on the minimum square error. The output layer contains one neuron. The neurons in the hidden layers have hyperbolic tangent sigmoid transfer function (T), while the output layer has only one neuron with linear transfer function (L).

The ANNs is fully connected with weighted connections, and there are biases in each layer. As shown in Figure 9 and Figure 10, there is a bias 'b' connected to each neuron, this bias has a value that is summed to the product of inputs 'p' by weights 'w', the result is then fed to transfer function '*f*' of its corresponding neuron as shown in eq. (11) to produce its output '*a*' that may be considered an input to the next layer. The values of these weights and biases change in the learning process.

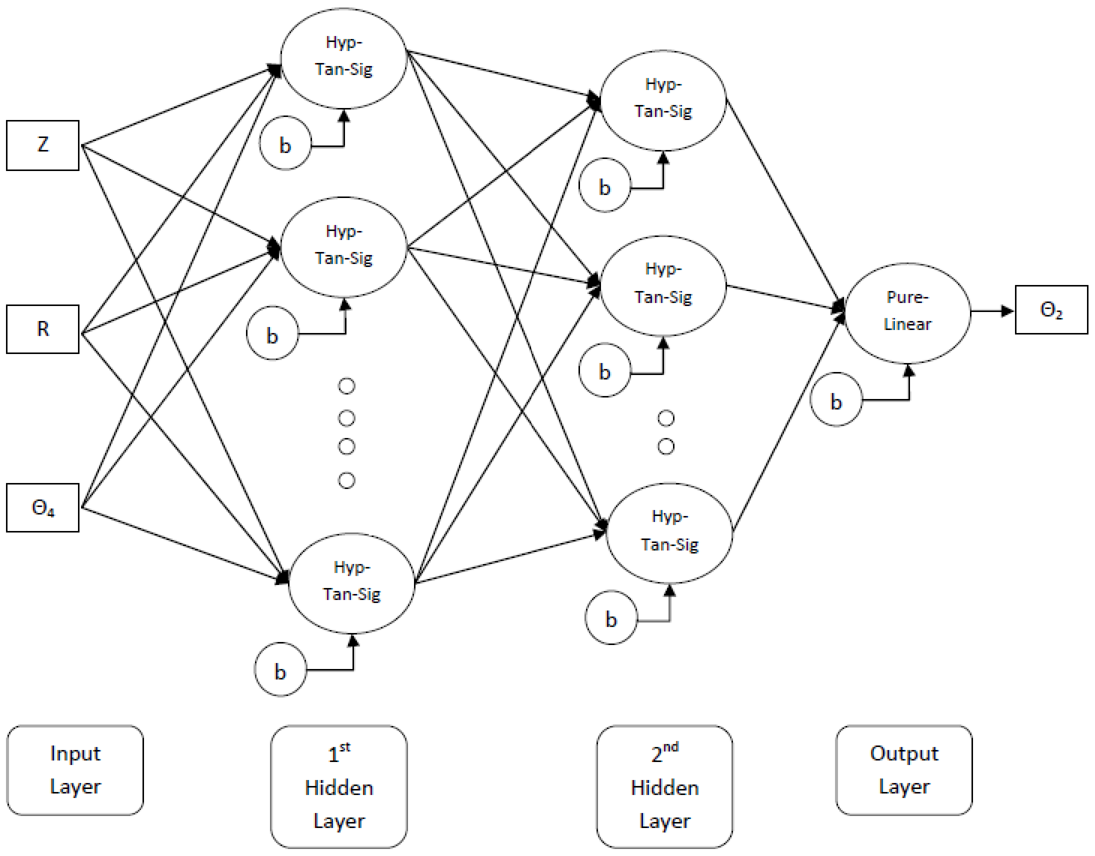


Figure 9 Detailed representation of ANN that generates (θ_2, θ_3)

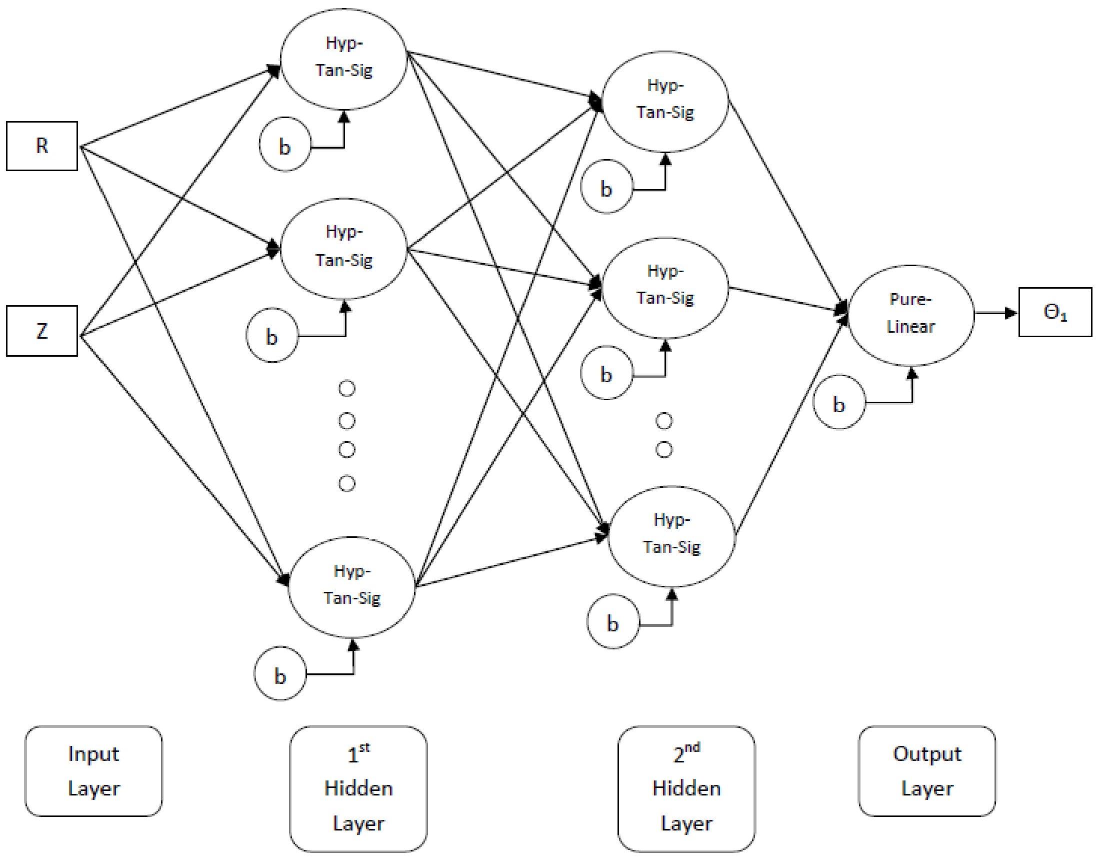


Figure 10 Detailed representation of ANN that generates (θ_1)

$$a = f\ (wp + b) \tag{11}$$

It’s notably that the number of layers and neurons in each layer is chosen experimentally according to some rules of thumb.

ANN Learning

In order to let the ANN learn the inverse kinematics of ED-7220C, the data set created previously should be used. The saved file has relatively large data, and if this amount of data pairs is used, this will consume relatively long time. So the solution was to reduce the amount of data used without reducing the distribution of the points that cover the working envelop of the robot. One pair of 30 pairs has been chosen to form the new data set, so the total amount used to train the ANN is (41 047) pairs, these pairs also were filtered from configurations those violate robots’ geometric constrains; this is done using Matlab code presented in the following chart, Figure 11.

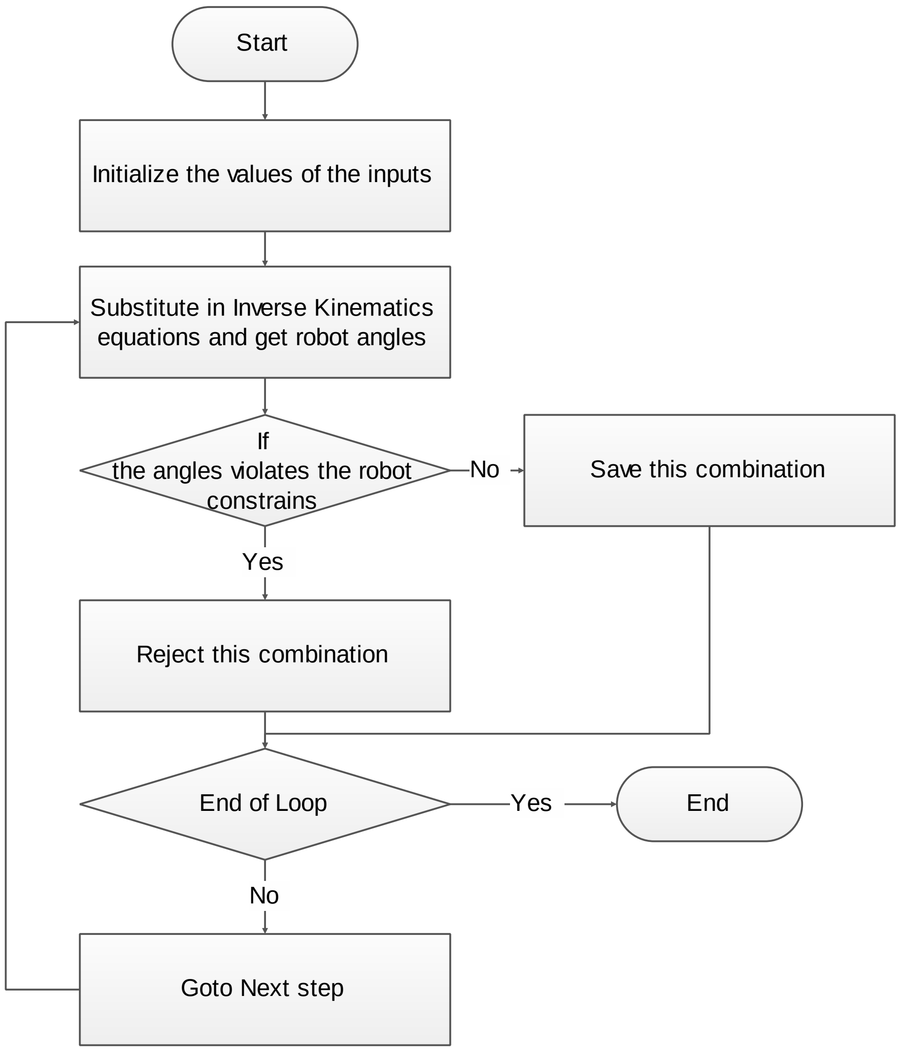

Start
Initialize the values of the inputs
Substitute in Inverse Kinematics equations and get robot angles
If the angles violates the robot constrains
No
Save this combination
Yes
Reject this combination
End of Loop
Yes
End
No
Goto Next step

Figure 11 Flow chart of creating the dataset

Neural network training can be made more efficient if certain preprocessing steps on the network inputs and targets are performed. They become part of the network object, so that whenever the network is used, the data coming into the network is preprocessed in the same way. In multilayer networks, sigmoid transfer functions are generally used in the hidden layers. These functions become essentially saturated when the net input is greater than three (exp (−3) = 0.05). If this happens at the beginning of the training process, the gradients will be very small, and the network training will be very slow. In the first layer of the network, the net input is a product of the input times the weight plus the bias. If the input is very large, then the weight must be very small in order to prevent the transfer function from becoming saturated. It is standard practice to normalize the inputs before applying them to the network.

Generally, the normalization step is applied to both the input vectors and the target vectors in the data set. In this way, the network output always falls into a normalized range. The network output can then be reverse transformed from the final layer

The dataset isn't totally used to train the ANN, but 70% only used for that, 15% is used for validation procedure, and the final 15% is used for testing the ANN. Segmentation of the data is done randomly.

Nguyen-Widrow [17] algorithm is used to initialize the network weights and biases. It generates random initial weight and bias values for a layer so that the active regions of the layer's neurons are distributed approximately evenly over the input space.

Mean square error function is used as performance function (e.g. cost function). Many training algorithms were tested to train the ANN, experimental results shows that Levenberg-Marquardt optimization algorithm provides relatively very good results with relatively short training time.

The Marquardt algorithm for nonlinear least squares is incorporated into the back propagation algorithm for training feed forward neural networks. The algorithm is tested on several function approximation problems, and is compared with a conjugate gradient algorithm and a variable learning rate algorithm. It is found that the Marquardt algorithm is much more efficient than either of the other techniques when the network contains no more than a few hundred weights [18].

The error surface of a nonlinear network (such as proposed networks) is more complex than the error surface of a linear network. The problem is that nonlinear transfer functions in multilayer networks introduce many local minima in the error surface. As gradient descent is performed on the error surface, depending on the initial starting conditions, it is possible for the network solution to become trapped in one of these local minima. Settling in a local minimum can be good or bad depending on how close the local minimum is to the global minimum and how low an error is required. One should be cautioned that although a multilayer back-propagation network with enough neurons can implement just about any function, back-propagation does not always find the correct weights for the optimum solution. One might want to reinitialize the network and retrain several times to obtain better solution.

Networks are sensitive to the number of neurons in their hidden layers. Too few neurons can lead to under-fitting. Too many neurons can contribute to over-fitting, in which all training points are well fitted, but the fitting curve oscillates wildly between these points. At any case, one should try to fit the function with minimum No. of layers those have also minimum No. of neurons, and increase the No. of neurons incrementally till reaching some sort of saturation (network performance doesn't improve at same rate), at which, it's time to increase the No. of hidden layers, and so on. Finally, comparison between results should be established, regarding the performance of ANN compared to its size and computation time.

Its notable to mention the machine used in calculating time needed for each neural network to calculate its output, is Intel® Core™ 2 Due CPU T7300 @ 2.00 GHz with 2.5 GB RAM, while the operating system type is 32-bit.
MATLAB® (2012b) with its Neural Network Toolbox is used in this study.

Results

After training several neural networks with different characteristics such as No. of layers, No. of neurons in each layer, transfer function of each layer and also applying different training algorithms, it was found the prescribed neural networks with their structure, Figure 12 and Figure 13, and training algorithms have high performance, as presented in Tables 2-5, with less calculation time related to other tested networks.

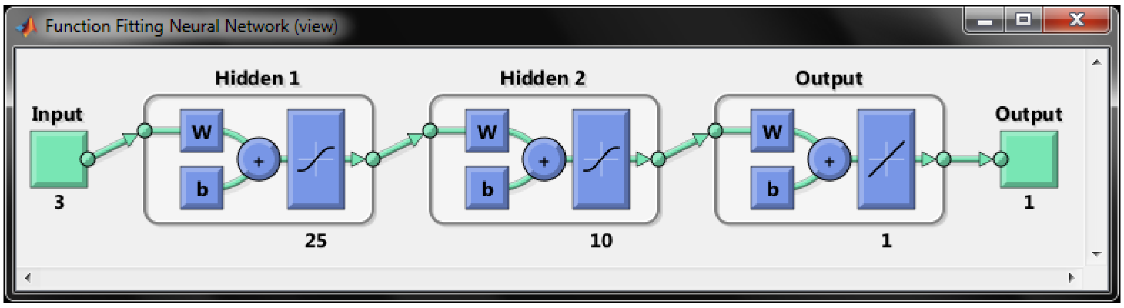


Figure 12 Simple representation of ANN that generates (θ_2, θ_3)

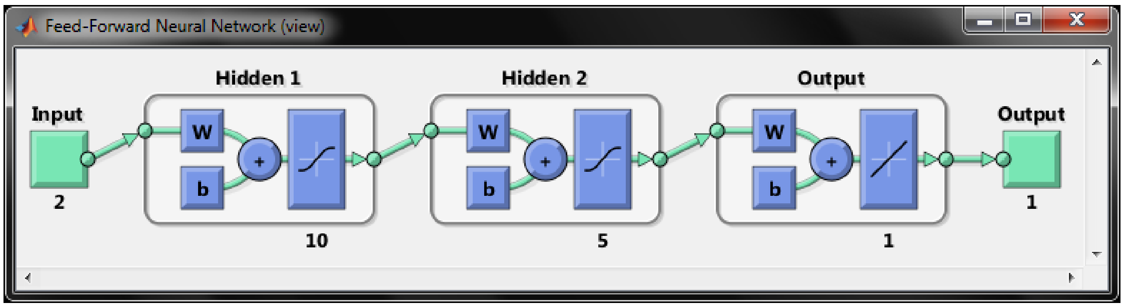


Figure 13 Simple representation of ANN that generates (θ_1)

Table 2 Results of trained networks that calculate Θ_1

No. of Hidden Layers	1	2	2	2	2
No. of Neurons	5	5,5	10,5	15,10	20,10
Transfer Function*	T,T	T,T,T	T,T,L	T,T,L	T,T,L
Mean Square Error	389.7	0.1475	5.67×10^{-5}	0.0072	1.3×10^{-4}
Computation time [sec]	0.0318	0.0348	0.0345	0.0346	0.0348

Table 3 Some trained networks that calculate Θ_2

No. of Hidden Layers	1	1	2	2	2
No. of Neurons	10	40	20,10	25,10	25,15
Transfer Function*	T,L	T,T	T,T,L	T,T,L	T,T,T
Mean Square Error	711	0.959	3.82	1.245×10^{-4}	0.0218
Computation time [sec]	0.0316	0.0322	0.0341	0.0346	0.0350

Table 4 Some trained networks that calculate Θ_3

No. of Hidden Layers	1	1	2	2	2
No. of Neurons	20	40	25,10	25,15	35,25
Transfer Function*	T,T	T,T	T,T,L	T,T,T	T,T,T
Mean Square Error	2.41	0.74	5.69×10^{-5}	0.0132	0.007
Computation time [sec]	0.0310	0.0320	0.0348	0.0350	0.0378

L: Pure Linear * T: Hyperbolic Tangent,

Table 5 Error statistics for proposed ANN

	Θ_1	Θ_2	Θ_3
Maximum error	0.2454	0.5094	0.4955
Minimum error	-0.3151	-0.4281	-0.4975
Mean Square Error	5.6684×10^{-05}	1.2447×10^{-04}	5.6964×10^{-05}
Root Mean Square Error	0.0075	0.0112	0.0075
Mean Absolute Error	0.0045	0.0061	0.0034
Mean Percentage of Absolute Error	0.0478%	0.0093%	0.0161%

* All units in degrees.

Figures (14-16) show the validation performance of the networks that produce (θ_1, θ_2, and θ_3 respectively) and the epoch that contains best performance while training. As depicted in figures, the Mean Square Error is almost under 10^{-3} degree, which indicates good learning performance compared to results discussed in literature, where minimum RMS error reached, was 0.02 degree in [14], minimum absolute error percentage reached, was 0.47% in [15], and the minimum RMS was 0.22 degrees in [19].

Figures (17-19) represent the error histogram for the proposed (θ_1, θ_2, and θ_3 respectively) networks. As presented in figures, the error histogram is composed of 20 bars showing the value of most repeated error. It also shows that error margins are almost between (-0.5, 0.5) degrees. That indicates that the percentage of absolute error is below 0.65%.

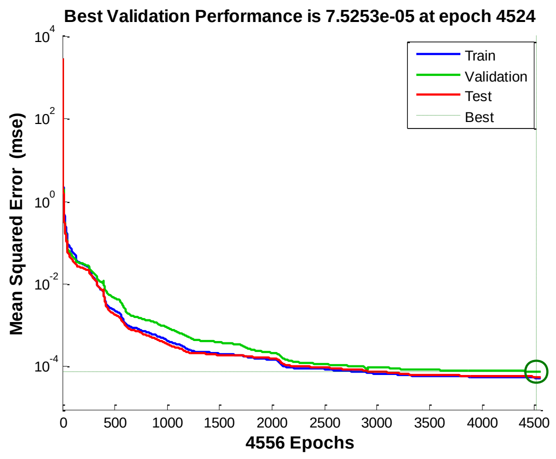


Figure 14 Validation performance for θ_1 (MSE in degrees)

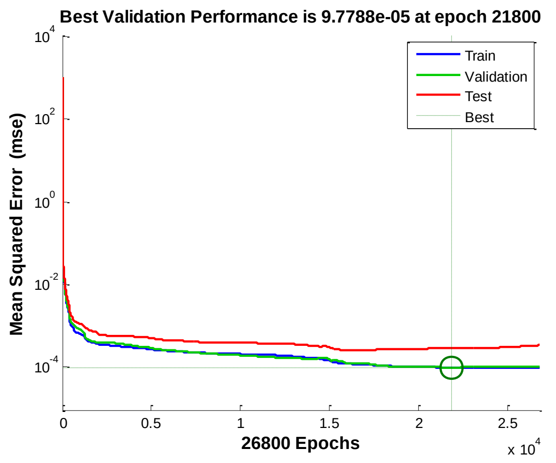


Figure 15 Validation performance for θ_2 (MSE in degrees)

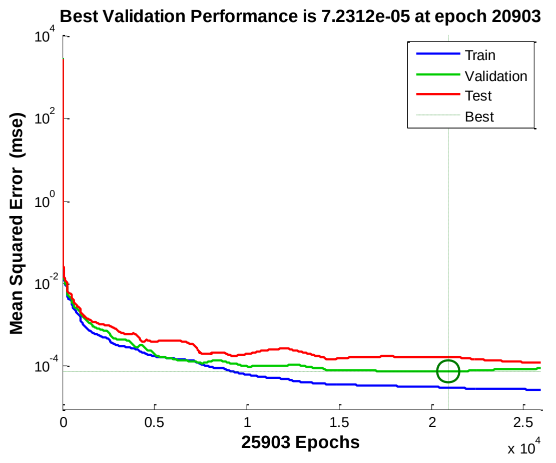


Figure 16 Validation performance for θ_3 (MSE in degrees)

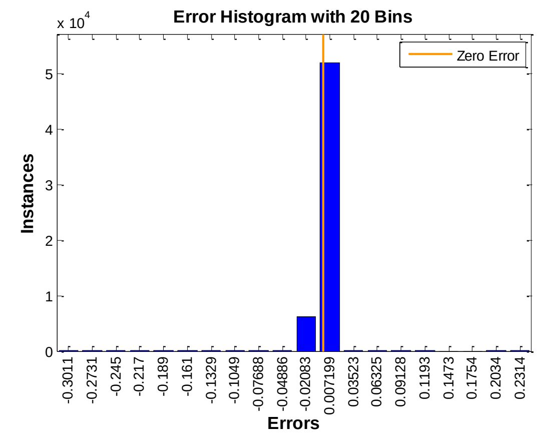


Figure 17 Error histogram for θ_1 (error in degrees)

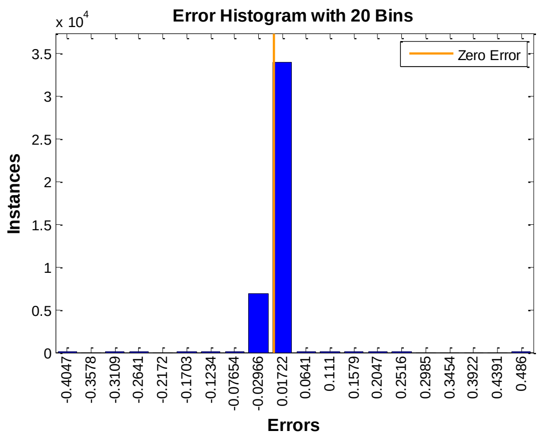


Figure 18 Error histogram for θ_2 (error in degrees)

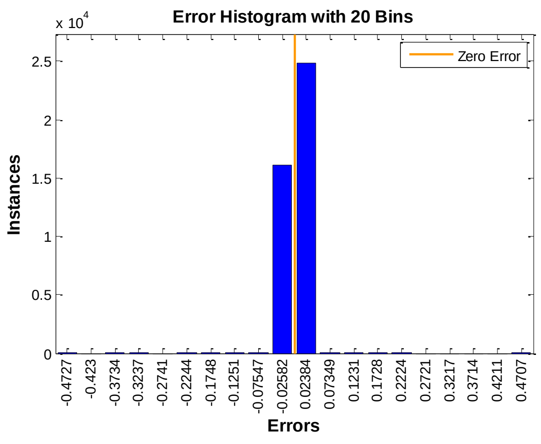


Figure 19 Error histogram for θ_3 (error in degrees)

Conclusion

Inverse kinematics of robotic manipulators is a non-trivial problem which has been addressed in recent publications [9-15, 19]. This paper focus on reducing the RMS error of generated robotic joint angles, this is accomplished by creating multiple ANNs; each one is dedicated to generate certain angle.

In this paper, it was suggested to split the inverse kinematic problem into two simple problems and solved by applying neural networks. Feasibility of using multilayer feed forward artificial neural network to model the inverse kinematics of ED-7220C is demonstrated. The data set is created using the inverse kinematics equations. Three ANNs are created, Number of hidden layers and also number of neurons in each hidden layer was chosen experimentally. After several experiments, hyperbolic tangent transfer function and L.M training algorithm shows satisfactory results compared to training time. The comparison between results achieved with the presented ANN and other stated studies in literature, indicates the former has lower level of RMS error. The proposed model showed that the ANN is a well generalized and it results a better error (RMS error is reduced by at least 90%) as compared to earlier studies [14, 15, 19].

References

1. Featherstone, R., *Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles.* The International Journal of Robotics Research, 1983. **2**(2): p. 35-45.
2. Lee, C.S.G., *Robot Arm Kinematics, Dynamics, and Control.* Computer, 1982. **15**(12): p. 62-80.
3. Duffy, J., *Analysis of mechanisms and robot manipulators*. 1980: Wiley.
4. Manocha, D. and J.F. Canny, *Efficient inverse kinematics for general 6R manipulators.* Robotics and Automation, IEEE Transactions on, 1994. **10**(5): p. 648-657.
5. Richard, P.P. and S. Bruce. *Kinematic control equations for simple manipulators*. in *Decision and Control including the 17th Symposium on Adaptive Processes, 1978 IEEE Conference on*. 1978.
6. Korein, J.U. and N.I. Badler, *Techniques for Generating the Goal-Directed Motion of Articulated Structures.* Computer Graphics and Applications, IEEE, 1982. **2**(9): p. 71-81.
7. Gasparetto, A. and V. Zanotto, *Optimal trajectory planning for industrial robots.* Advances in Engineering Software, 2010. **41**(4): p. 548-556.
8. Howard, D. and A. Zilouchian, *Application of Fuzzy Logic for the Solution of Inverse Kinematics and Hierarchical Controls of Robotic Manipulators.* Journal of Intelligent and Robotic Systems, 1998. **23**(2-4): p. 217-247.
9. Oyama, E., et al. *Inverse kinematics learning by modular architecture neural networks with performance prediction networks*. in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. 2001.
10. Köker, R., et al., *A study of neural network based inverse kinematics solution for a three-joint robot.* Robotics and Autonomous Systems, 2004. **49**(3–4): p. 227-234.
11. Bingul, Z., H.M. Ertunc, and C. Oysu. *Comparison of inverse kinematics solutions using neural network for 6R robot manipulator with offset*. in *Computational Intelligence Methods and Applications, 2005 ICSC Congress on*. 2005.

12. Pham, D.T., M. Castellani, and A.A. Fahmy. *Learning the inverse kinematics of a robot manipulator using the Bees Algorithm*. in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. 2008.
13. Mayorga, R.V. and P. Sanongboon, *Inverse kinematics and geometrically bounded singularities prevention of redundant manipulators: An Artificial Neural Network approach.* Robotics and Autonomous Systems, 2005. **53**(3–4): p. 164-176.
14. Chiddarwar, S.S. and N. Ramesh Babu, *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, 2010. **23**(7): p. 1083-1092.
15. Hasan, A.T., et al., *An adaptive-learning algorithm to solve the inverse kinematics problem of a 6 D.O.F serial robot manipulator.* Advances in Engineering Software, 2006. **37**(7): p. 432-438.
16. ED-R&D-Center, *ARM ROBOT TRAINER, Instruction Manual Ed-7220C*. 2007: Yong-Hoo Park.
17. Widrow, D.N.a.B. *Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights*. in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. June 1990.
18. Hagan, M.T. and M.B. Menhaj, *Training feedforward networks with the Marquardt algorithm.* Neural Networks, IEEE Transactions on, 1994. **5**(6): p. 989-993.
19. Chaudhary, H. and R. Prasad, *INTELLIGENT INVERSE KINEMATIC CONTROL OF SCORBOT-ER V PLUS ROBOT MANIPULATOR.* International Journal of Advances in Engineering & Technology, IJAET, 2011. **Vol. 1**(5): p. 158-169.