International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878, Impact Factor (2014): 3.05

Neural Network Based Inverse Kinematics Solution for 6-R Robot Using Levenberg-Marquardt Algorithm

Prashant Badoni

Mechanical Engineering Department, Graphic Era University, Dehradun - 248002, India

Abstract: ***The traditional approaches are insufficient to solve the complex kinematics problems of the redundant robotic manipulators. To overcome such intricacy, ANNs are used nowadays. The performance of the neural network is affected by the training algorithm and network topology. There are numerous training algorithms which are used in the training of neural networks. In this paper, Levenberg-Marquardt is used in training algorithm and its effect on the performance of the neural network on the inverse kinematics model learning of a 6-R robot is studied.***

Keywords: Inverse Kinematics Solution, MATLAB Toolbox, Neural Networks, Robot manipulator, Training Algorithm.

1. Introduction

Neural network is one of the prominent artificial intelligent techniques used in the robotics to accomplish more intelligence in systems with high degree of autonomy. ANN incorporates learning ability which provides flexibility to the robotic systems. Neural network can be implemented using MATLAB software. Procedure to train the neural network model is as follows:

- Data collection
- Network creation
- Network configuration
- Initialization of the weights and biases
- Network training
- Network validation
- Use the network

The working principal of a neural network is based on learning from the formerly obtained data set known as training set, and then go through the success of system using test data. The learning algorithm affects the employment of the neural network greatly. In this study, the effects of Levenberg-Marquardt learning algorithm have been tested for the inverse kinematics solution of a six joint robotic manipulator.

This paper is organized as follows: Section II provides the kinematics analysis of the 6-R robot. Section III of the paper deals with the neural network based inverse kinematics solution. Section IV describes training and testing. Section V gives results and a discussion, and finally Section VI concludes the paper.

2. Kinematic Analysis of 6-R Robot

A Robot manipulator is composed of a group of links (rigid bodies) connected together by revolute or prismatic joints which allow motion for the desired link. Robot Kinematics refers to the analytical study of the motion of a robot manipulator without regard to any factor (like force) which influence the robot movement. Robot Kinematics can be split into forward and inverse kinematics. In the forward kinematics problem, the end effector's location in the work space, that is position and orientation, is determined based on the joint variables [1] [2] [3]. The forward kinematics problem may express mathematically as follows:

$F(\theta_1, \theta_2, \theta_3 \ldots \theta_n) = [p_x, p_y, p_z, R]$

Where, $\theta_1, \theta_2, \theta_3 \ldots \theta_n$ are the input variables, $[p_x, p_y, p_z]$ are desired position and R is the desired rotation.

The inverse kinematics problem refers to finding the values of the joint variables that allows the manipulator to reach the given location. The inverse kinematics problem can be expressed mathematically as follows:

$F[p_x, p_y, p_z, R] = (\theta_1, \theta_2, \theta_3 \ldots \theta_n)$

The joint variables are the link extension in the case of prismatic joints, or the angles between the links in case of rotational joints.

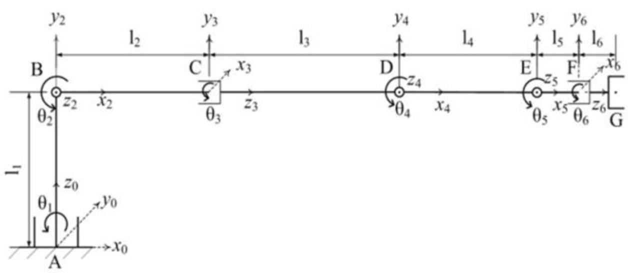

Figure 1: D-H coordinates of the robot

Figure 1 depicts the structure and coordinates of 6-DOF robot manipulator which is studied in during the work.

The D-H parameters of the manipulator are listed in Table1.

Table 1: D-H parameters of the manipulator

Joints	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	$\pi/4$	0	θ_2
3	l_2	$\pi/4$	l_1	θ_3

4	l_3	$-\pi/4$	0	θ_4
5	0	0	0	θ_5
6	l_5	$\pi/4$	l_4	θ_6

According the Denavit-Hartenberg method, the transformation can be formulated in the chain product of six successive homogeneous matrices $^{i-1}_{i}T$.

$^{i-1}_{i}T$ is the homogeneous transformation matrix relating the i^{th} coordinate frame to the $(i\text{-}1)^{th}$ coordinate frame [4].

$^{0}_{i}T = {}^{0}_{1}T\,{}^{1}_{2}T \ldots\ldots\ldots {}^{5}_{6}T$ **(1)**

Equation (1) contains a large set of trigonometric functions:

$\sin\theta_i = S_i$ *and* $\cos\theta_i = C_i$.

$^{0}_{i}T$ can be calculated by following:

$$^{0}_{i}T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2)$$

In (2), n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, a_z show the rotational elements of the transformation matrix and p_x, p_y, p_z refer to the elements of the position vector.

$n_x = -C_1S_2C_3C_{45}S_6 + S_1S_3C_{45}S_6 - C_1C_2S_{45}S_6 - C_1S_2S_3C_6 - S_1C_3C_6$

$n_y = -S_1S_2C_3C_{45}S_6 - C_1S_3C_{45}S_6 - S_1C_2S_{45}S_6 - C_1S_2S_3C_6 + C_1C_3C_6$

$n_z = C_2C_3C_{45}S_6 - S_2S_{45}S_6 + C_2S_3C_6$

$o_x = -C_1S_2C_3C_{45}S_6 + S_1S_3C_{45}C_6 - C_1C_2S_{45}C_6 + C_1S_2S_3S_6 + S_1C_3S_6$

$o_y = -S_1S_2C_3C_{45}C_6 - C_1S_3C_{45}C_6 - S_1C_2S_{45}C_6 + S_1S_2S_3S_6 - C_1C_3S_6$

$o_z = C_2C_3C_{45}C_6 - S_2S_{45}C_6 - C_2S_3S_6$

$a_x = -C_1S_2C_3C_{45} + S_1S_3C_{45} + C_1C_2S_{45}$

$a_y = -S_1S_2C_3C_{45} - C_1S_3C_{45} + S_1C_2S_{45}$

$a_z = C_2C_3S_{45} + S_2S_{45}$

$p_x = (-C_1S_2C_3S_{45} + S_1S_3S_{45} + C_1C_2C_{45})\, l_5 + (-C_1S_2C_3S_4 + S_1S_3S_4 + C_1C_2C_4)\, l_4 + C_1C_2\, l_{23}$

$p_y = (-S_1S_2C_3S_{45} - C_1S_3S_{45} + S_1C_2C_{45})\, l_5 + (-S_1S_2C_3S_4 - C_1S_3S_4 + S_1C_2C_4)\, l_4 + S_1C_2\, l_{23}$

$p_z = (C_2C_3S_{45} + S_2C_{45})\, l_5 + (C_2C_3S_4 + S_2C_4)\, l_4 + S_2l_{23} + l_1$

The inverse kinematics solution for the robot is indicated as follows:

$\cos\theta_4 = -(l_{23}^2 + l_4^2 - d^2) / 2\, l_{23}\, l_4$ **(3)**

The equation obtained from (2) as:

$(S_{45}\, l_5 + S_4\, l_4)^2 + (C_{45}\, l_5 + C_4\, l_4 + l_{23})^2 = p_x^{\,2} + p_y^{\,2} + (p_z - l_1)^2$ **(4)**

Then,

$\theta_5 = \arcsin\, [\{p_x^{\,2} + p_y^{\,2} + (p_z - l_1)^2 - l_5^2 - S_4^2\, l_4^2 - (C_4\, l_4 + l_{23})^2\}/A] - \arctan\, [(C_4\, l_4 + l_{23}) / S_4\, l_4] - \theta_4$ **(5)**

We could also obtain the following expressions:

$\theta_2 = \arcsin\, [(C_{45}\, l_5 + C_4\, l_4 + l_{23}) / \sqrt{(p_z - l_1)^2 + p_x^{\,2}}] - \arctan\, [p_x / (p_z - l_1)]$

$\theta_1 = \arcsin\, [(S_2C_3S_4 - C_2C_4 - C_2l_{23}) / \sqrt{(p_x - a_x\, l_5)^2 + (p_y - a_y\, l_5)^2}] - \arctan\, [(p_x - a_x\, l_5) / (p_y - a_y\, l_5)]$

$\theta_3 = \arcsin\, [(C_1 + p_x - a_x\, l_5) / S_1S_4l_4\, (a_z - S_2C_{45})] - \arctan\, [(p_x - a_x\, l_5) / (p_y - a_y\, l_5)]$

The strategy to solve inverse kinematics problem tend to be time consuming, so there is usually low interest in applying this technique for kinematic calculations. The trained neural network can give the inverse kinematics solution quickly for any given Cartesian coordinate in a robotic system.

3. Neural Network based Inverse Kinematics Solution

Neural networks are generally used in the modeling of nonlinear processes. ANN is a parallel-distributed information processing system. To form a trainable nonlinear system, it stores the samples with distributed coding. Training of a neural network can be expressed as a mapping between any given input and output data set. Neural networks have some advantages, such as adoption, learning and generalization. Implementation of a neural-network model requires us to decide the structure of the model, the type of activation function and the learning algorithm.

In Figure 3, the schematic representation of a neural network based inverse kinematics solution is given. The solution system is based on training a neural network to solve an inverse kinematics problem based on the prepared training data set using direct kinematics equations. In Figure 2, "e" refers to error – the neural network results will be an approximation, and there will be an acceptable error in the solution.

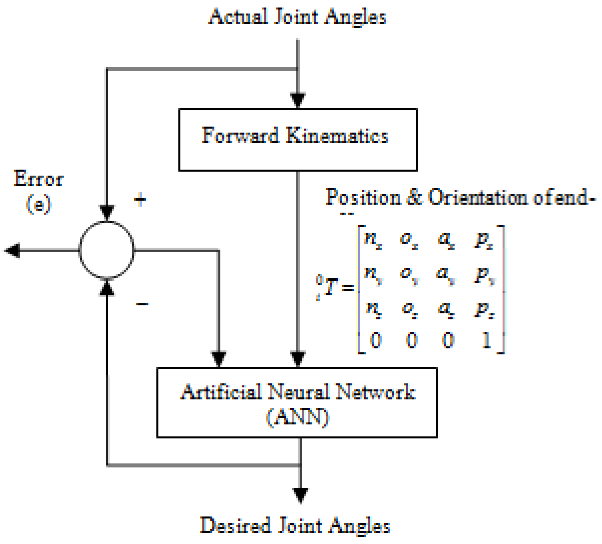

Figure 2: ANN based inverse kinematics solution system [5]

The designed neural-network topology is given in Figure 3. A feed-forward multilayer neural-network structure was designed including 12 inputs and 6 outputs. Only one hidden layer was used during the study.

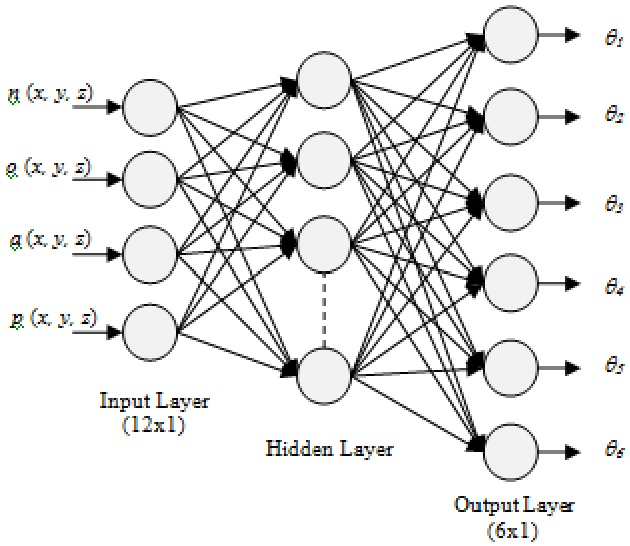

Figure 3: Structure of neural network used in this study

4. Training Method

To train the network we must provide the ANN with the dataset. ANN is trained with the data which is generated by a fifth-order polynomial trajectory planning algorithm. The equation for fifth-order polynomial trajectory planning is given in following equation:

$$\theta_i (t)= \theta_{io} + 10/t_f{}^3(\theta_{if} - \theta_{io})\, t^3 + 15/t_f{}^4 (\theta_{if} - \theta_{io})\, t^4 + 6/t_f{}^5(\theta_{if} - \theta_{io})\, t^5; \; i = 1, 2, 3 \ldots\ldots n \quad \textbf{(6)}$$

Where,

$\theta_i (t)$ = angular position at time t
θ_{io} = initial position of the i^{th} joint
θ_{if} = final position of the i^{th} joint
n = number of joints
t_f = arrival time from initial position to the target

Initial and final angular positions are defined to produce data in the workspace of robot. After gathering data from the whole network is trained in the back propagation mode and all the weighs are updated according to the new training data. For the training, 100 data values corresponding to the (θ_1, θ_2, θ_3....θ_6) joint angles according to the different (n_x, o_x, a_x, p_x, n_y, o_y, a_y, p_y, n_z, o_z, a_z, p_z) Cartesian coordinate parameters were generated by using (6) based on kinematic equations given in (2). A sample data set produced for the training of neural networks is given in Table 2 and 3.

Table 2: A sample input data set for the training of neural networks

Inputs											
n_x	n_y	n_z	o_x	o_y	o_z	a_x	a_y	a_z	p_x	p_y	p_z
-0.9467	-0.0186	-0.3216	-0.3188	-0.0909	0.9435	-0.0468	0.9957	0.0802	10.7388	18.5368	-3.6269
0.4031	-0.4414	-0.8017	-0.8689	-0.4595	-0.1839	-0.2872	0.7707	-0.5688	-15.6523	-7.3269	7.3444
-0.9209	0.3787	-0.0921	-0.3898	-0.8943	0.2199	0.0009	0.2384	0.9712	1.0641	2.808	-13.0244
0.7646	-0.3951	-0.5091	0.6249	0.6478	0.4358	0.1576	-0.6514	0.7422	-9.4243	-0.5505	7.7646
0.2783	0.4263	-0.8607	0.9056	-0.4151	0.0872	-0.32	-0.8037	-0.5016	5.307	-5.8058	-0.9137
0.1042	-0.5075	-0.8553	0.0489	0.8616	-0.5053	0.9934	0.0108	0.1146	-1.071	1.0777	-3.8184
-0.7806	0.5435	0.3086	0.4541	0.1539	0.8775	0.4295	0.8251	-0.367	-2.8101	-1.7129	4.4017
-0.4807	0.2339	-0.8451	0.6899	0.6958	-0.1999	0.5413	-0.6791	-0.4958	2.7343	5.3178	2.8208
-0.7663	-0.6233	-0.1561	-0.4972	0.4213	0.7585	-0.407	0.6588	-0.6327	-8.1384	2.135	0.6667
-0.4693	0.873	-0.1325	0.8665	0.4264	-0.2594	-0.1699	-0.2366	-0.9566	-7.5946	2.9237	3.6417
-0.5937	-0.5062	-0.6255	-0.7906	0.5119	0.3361	0.1501	0.694	-0.7041	-4.8593	-1.8317	8.8997

Table 3: A sample target data set for the training of neural networks

Targets					
θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
169.779	74.3217	-79.0853	169.779	-6.3597	169.779
176.5911	78.1062	-73.0301	176.5911	-2.1211	176.5911
183.4089	81.8938	-66.9699	183.4089	2.1211	183.4089
190.221	85.6783	-60.9147	190.221	6.3597	190.221
197.0165	89.4536	-54.8742	197.0165	10.588	197.0165
203.7842	93.2135	-48.8585	203.7842	14.7991	203.7842
210.5132	96.9518	-42.8772	210.5132	18.986	210.5132
217.1925	100.6625	-36.94	217.1925	23.142	217.1925
223.8114	104.3396	-31.0566	223.8114	27.2604	223.8114
230.3591	107.9773	-25.2363	230.3591	31.3346	230.3591
236.8254	111.5696	-19.4886	236.8254	35.358	236.8254

Learning / Training Function

'Trainlm' from MATLAB toolbox is a network training function which updates weight and bias values according to Levenberg-Marquardt optimization [6]. It is the fastest back propagation algorithm in the MATLAB toolbox, and is immensely suggested as a first-choice supervised algorithm.

5. Result and Discussion

In this study, Neural Network Fitting Tool (using command: nftool) is used to create and train the network. The dataset is loaded into selected data window [6]. The network is trained using the input data and the performance plot, training state and regression plots are observed. In this training, Random (dividerand) rule divides the data where 70% data are assigned to training set, 15% to validation and 15% data to test set. As shown in Figure 4, this time the training continued for the maximum of 1000 iterations.

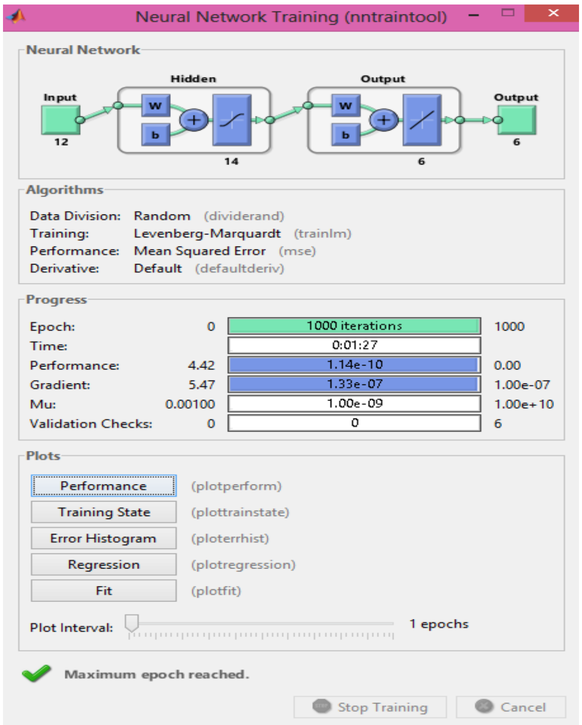

Figure 4: Neural Network Training

From the training state plot it is seen that training continued for iterations before the training stopped. The performance plot shown in Figure 6 does not indicate any major problems with the training. The validation and test curves are very similar.

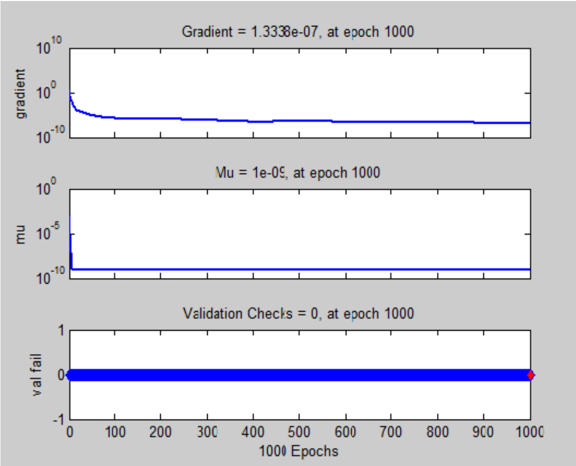

Figure 5: Training Plot

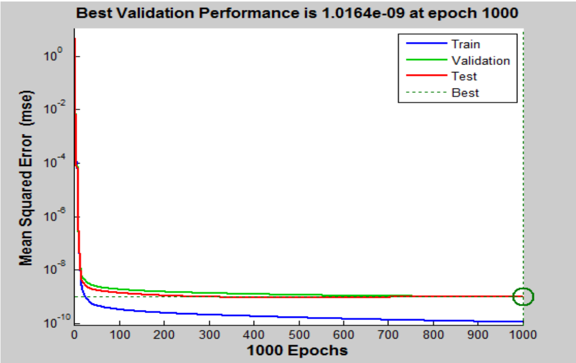

Figure 6: Performance Plot

6. Conclusion

Mostly mathematical models fail to simulate the complex nature of inverse kinematics problem. In contrast, ANN is based on the data input/output data pairs to determine the structure and parameters of the model. Also, ANN's can always be updated in order to achieve better results by presenting new training examples as new data become available.

References

[1] M.W. Spong, S. Hutchinson and M. Vidyasagar, Robot Modeling and Control, 1st Edition, Jon Wiley & Sons, Inc, 2005.
[2] J. Angeles, Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms, 2nd Edition, Springer, 2003.
[3] J. J. Crage, Introduction to Robotics Mechanics and Control, 3rd Edition, Prentice Hall, 2005.
[4] J. Denavit and R. Hartenberg, “A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices” of Applied Mechanics, pp. 215-221, 1955.
[5] R. Köker, T. Çakar, Y. Sari, A neural-network committee machine approach to the inverse kinematics problem solution of robotic manipulators. Engineering with Computers, Springer-Verlag London, DOI 10.1007/s00366-013-0313-2, 2013.
[6] The Mathworks Neural Network Toolbox user guide. Available on line on: http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf