A study of neural network based inverse kinematics solution for a three-joint robot

Article *in* Robotics and Autonomous Systems · December 2004

DOI: 10.1016/j.robot.2004.09.010 · Source: dx.doi.org

CITATIONS
164

READS
2,917

4 authors:

Raşit Köker
Sakarya University of Applied Sciences
50 PUBLICATIONS **1,449** CITATIONS
SEE PROFILE

Cemil Oz
Sakarya University
46 PUBLICATIONS **871** CITATIONS
SEE PROFILE

Tarik Cakar
Sakarya University
26 PUBLICATIONS **489** CITATIONS
SEE PROFILE

Hüseyin Ekiz
Sakarya University
37 PUBLICATIONS **637** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Robotics and Autonomous Systems 49 (2004) 227–234
Robotics and Autonomous Systems
www.elsevier.com/locate/robot

A study of neural network based inverse kinematics solution for a three-joint robot

Raşit Köker[a,*], Cemil Öz[a], Tarık Çakar[b], Hüseyin Ekiz[c]

[a] *Department of Computer Engineering, Sakarya University, 54187 Sakarya, Turkey*
[b] *Department of Industrial Engineering, Sakarya University, 54187 Sakarya, Turkey*
[c] *Department of Electronics Education, Sakarya University, 54187 Sakarya, Turkey*

Received 8 March 2002; received in revised form 26 February 2003

Abstract

A neural network based inverse kinematics solution of a robotic manipulator is presented in this paper. Inverse kinematics problem is generally more complex for robotic manipulators. Many traditional solutions such as geometric, iterative and algebraic are inadequate if the joint structure of the manipulator is more complex. In this study, a three-joint robotic manipulator simulation software, developed in our previous studies, is used. Firstly, we have generated many initial and final points in the work volume of the robotic manipulator by using cubic trajectory planning. Then, all of the angles according to the real-world coordinates (x, y, z) are recorded in a file named as training set of neural network. Lastly, we have used a designed neural network to solve the inverse kinematics problem. The designed neural network has given the correct angles according to the given (x, y, z) cartesian coordinates. The online working feature of neural network makes it very successful and popular in this solution.

Keywords: Artificial neural networks; Robotics; Inverse kinematics solution

1. Introduction

The inverse kinematics problem for a robotic manipulator is obtaining the required manipulator joint values for a given desired end point position and orientation. The inverse kinematics problem is usually complex for robotic manipulators. There are three traditional methods used to solve inverse kinematics problem: geometric [6,12], algebraic [5,13,15,8] and iterative [10] methods. Every method has some disadvantages. The algebraic methods do not guarantee closed form solutions. In case of using geometric methods, closed form solutions for the first three

* Corresponding author. Tel.: +90 264 346 0353; fax: +90 264 346 0351.
E-mail address: rkoker@sakarya.edu.tr (R. Köker).

0921-8890/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2004.09.010

joints of the manipulator must exist geometrically. The iterative methods converge to only a single solution and this solution depends on the starting point. For simple manipulator geometry, the problem is solved using trigonometry approaches involving tedious mathematical steps. If the joint structure of the manipulator is more complex, the inverse kinematics solution by using these traditional methods is a time consuming study. The computation of inverse kinematics using artificial neural network is particularly useful where shorter calculation times are required, such as in real-time adaptive robot control. In other words, for a more generalized m-degrees of freedom manipulator, traditional methods will become prohibitive due to the high complexity of mathematical structure of the formulation. To compound the problem further, robots have to work in the real world that cannot be modeled concisely using mathematical expressions [9].

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. Nonlinear dynamic systems, including robots executing tasks repetitively, were shown to benefit from the use of neural network controllers [2]. The results from these and other experiments reported in the literature demonstrated that neural network is useful in many problems in robotics area [3]. In this study inverse kinematics solution using neural networks is presented with theoretical background of neural networks.

2. Inverse kinematics problem in robotics

Robot arm kinematics deals with the analytical study of the geometry of motion of a robot arm with respect to a fixed reference coordinate system as a function of time without regard to the forces/moments that cause the motion. For a given manipulator, given the joint angle vector at time t $q(t) = q_1(t), q_2(t), \ldots, q_m(t)$ and the geometric link parameters, where n is the number of degrees of freedom, the position and orientation of the end-effector of the manipulator with respect to a reference coordinate system is usually refered to as the direct kinematics problem.

Given a desired position and orientation of the end-effector of the manipulator and the geometric link parameters with respect to a reference coordinate system, the joint angle vector is the inverse kinematics problem. In this paper, we designed a neural network to solve the inverse kinematics problem.

Since the independent variables in a robotic manipulator are the joint variables and a task is usually stated in terms of the reference coordinate frame, the inverse kinematics problem is used more frequently in many industrial robotic applications [1,14].

3. Artificial neural network

Artificial neural network (ANN) is a parallel-distributed information processing system. This system is composed of operators interconnected via one-way signal flow channels. ANN stores the samples with a distributed coding, thus forming a trainable nonlinear system. It includes hidden layer(s) between the inputs and outputs. The main idea of the ANN approach resembles the human brain functioning. Therefore, ANN has a quicker response and higher performance than a sequential digital computer. Given the inputs and desired outputs, it is also self-adaptive to the environment so as to respond different inputs rationally. However, it has a complex internal structure, so the ANN's realized to date are composite systems imitating basic biological functions of neurons. The generalization feature of neural network is used in this study [14].

Many neural network models use threshold units with sigmoidal activation function and gradient descent learning algorithm. In this study, the same activation function and learning algorithm are used, and the backpropagation algorithm is explained below.

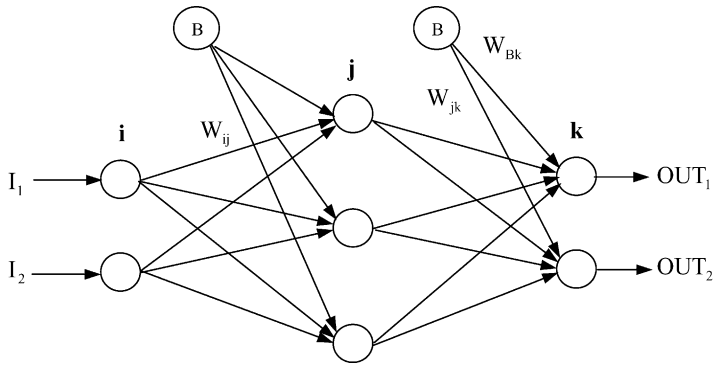

Fig. 1. A sample multi-layer feedforward net structure.

3.1. Backpropagation algorithm

The backpropagation is widely used algorithm, and it can map nonlinear processes. It is a feedforward network with the one or more hidden layers. The elementary architecture of the backpropagation network has three layers: input, hidden and output layers. There are no constraints about the number of hidden layers. Backpropagation is a systematic method for training multilayer artificial neural networks. It has a strong mathematical foundation based on gradient descent learning. In Fig. 1, a sample multi-layer feedforward net structure, which has one hidden layer, is given, and all parameters are given according to this figure below.

3.1.1. Forward pass process in neural network

NET values can be obtained by multiplying inputs and related weights. To calculate NET values out of perceptrons in hidden layer; the formulas (1 and 2) given below is used. Here, the indices *i*, *j*, and *k* refer to different neurons (perceptrons) in the network; with signals propagating through the network from left to right, neuron *j* lies in a layer to the right of neuron *i*, and neuron *k* lies in a layer to the right of neuron *j* when neuron *j* is a hidden layer as shown in Fig. 1. NET_j Refers to the net interval activity of neuron *j*. It constitutes the signal applied to the nonlinearity associated with neuron *j*. OUT_k, refers to the output value of the neuron *k* after the activation function is applied. The symbol $W_{ij}(n)$ denotes the synaptic weight connecting the output of neuron *i* to the input of neuron *j* at iteration *n*.

$$\mathrm{NET}_{j=1\text{–}3} = \sum I_i W_{ij} + W_{\mathrm{B}j}^{\mathrm{NEW}} \tag{1}$$

$$\mathrm{OUT}_{j=1\text{–}3} = \frac{1}{1 + \mathrm{e}^{-\mathrm{NET}_j}} \tag{2}$$

To calculate the OUT values, NET values are applied to an activation function. Similarly, the outs of hidden units are considered as inputs of the next hidden layer units or output layer units. $W_{\mathrm{B}j}^{\mathrm{NEW}}$ and $W_{\mathrm{B}k}^{\mathrm{NEW}}$ values present biases. OUT and NET are calculated by using formulas (3) and (4).

$$\mathrm{NET}_{k=1,2} = \sum I_j W_{jk} + W_{\mathrm{B}k}^{\mathrm{NEW}} \tag{3}$$

$$\mathrm{OUT}_{k=1,2} = \sum \frac{1}{1 + \mathrm{e}^{-\mathrm{NET}_k}} \tag{4}$$

3.1.2. Error propagation in backpropagation algorithm

The outputs of neural network model are obtained from the output layer units. The difference between target values and actual values are considered as system error. The obtained error values are propagated back to connection weights. This process is applied using the following equations:

Firstly, from output layer to last hidden layer;

$$\delta_k = f'(\mathrm{NET}_k)(\mathrm{TARGET}_k - \mathrm{OUT}_k) \tag{5}$$

$$\delta_k = \mathrm{OUT}_k(1 - \mathrm{OUT}_k)(\mathrm{TARGET}_k - \mathrm{OUT}_k) \tag{6}$$

$$\Delta W_{kj}(n+1) = \eta\delta_k \mathrm{OUT}_k + \alpha[\Delta W_{kj}(n)] \tag{7}$$

$$W_{kj}^{\mathrm{NEW}} = W_{kj}^{\mathrm{OLD}} + \Delta W_{kj}(n+1) \tag{8}$$

where TARGET_k presents desired output value, η the learning rate, α the momentum coefficient, $f'(\mathrm{NET}_k)$ represents activation function, n presents iteration number and ΔW is the change of related weight. This term is added to old weight of corresponding connection to obtain new weight. Here,δ_k refers error gradient at neuron k.

Secondly, from hidden layer to input layer;

$$\delta_j = f'(\mathrm{NET}_j)\sum \delta_k W_{kj} \tag{9}$$

$$\delta_j = \mathrm{OUT}_j(1 - \mathrm{OUT}_j)\sum \delta_k W_{kj} \tag{10}$$

$$\Delta W_{ji}(n+1) = \eta\delta_j \mathrm{OUT}_j + \alpha[\Delta W_{ji}(n)] \tag{11}$$

$$W_{ji}^{\mathrm{NEW}} = W_{ji}^{\mathrm{OLD}} + \Delta W_{ji}(n+1) \tag{12}$$

The bias affects the activation function in order to force the learning process; therefore the speed of learning process increases. Biases are recomputed as following, for the biases of output layer, where the letters and symbols have similar meanings;

$$\Delta W_{\mathrm{B}k}(n+1) = \eta\delta_k + \alpha[\Delta W_{\mathrm{B}k}(n)] \tag{13}$$

$$W_{\mathrm{B}j}^{\mathrm{NEW}} = W_{\mathrm{B}j}^{\mathrm{OLD}} + \Delta W_{\mathrm{B}k}(n+1) \tag{14}$$

and for the biases of hidden layer;

$$\Delta W_{\mathrm{B}j}(n+1) = \eta\delta_j + \alpha[\Delta W_{\mathrm{B}j}(n)] \tag{15}$$

$$W_{\mathrm{B}j}^{\mathrm{NEW}} = W_{\mathrm{B}j}^{\mathrm{OLD}} + \Delta W_{\mathrm{B}j}(n+1) \tag{16}$$

The training of the neural network model, as explained above, is carried out in two steps. The first step is called forward pass, which is composed of calculation for NET and OUT values. The second step is called backward pass that is composed of error propagation throughout connection weights. The iterative process is repeated until a satisfactory learning level is achieved, i.e. the differences between TARGET and OUT are minimized [4,7,16].

4. Cubic trajectory planning

In a constant time, let's investigate the problem of getting manipulator from an initial position to a target position. In order to find a function for each joint between the initial position, θ_0, and final position, θ_f, the group of angles can be calculated based on this target position and orientation of the manipulator by using system kinematics.

It is necessary to have at least four-limit value on the $\theta(t)$ function that belongs the joints. Here $\theta(t)$ denotes the angular position at time t. Two limit values of the function are the initial and final position of the joint.

$$\theta(0) = \theta_0 \tag{17}$$

$$\theta(t_\mathrm{f}) = \theta_\mathrm{f} \tag{18}$$

Additional two limit values shown below (19) and (20) come from the function that is continuous at the point of joint speed. In other words, the speed will be zero at the beginning and the target position of the joint.

$$\dot{\theta}(0) = 0 \tag{19}$$

$$\dot{\theta}(t_f) = 0 \tag{20}$$

These four conditions can be provided by using a third order polynomial, since a cubic polynomial has four coefficient, as given below (21) and (22). These conditions may determine the cubic path all by itself. A cubic trajectory can be written as,

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{21}$$

The speed and acceleration of the joint along the trajectory are given below in (22) and (23).

$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{22}$$

$$\ddot{\theta}(t) = 2a_2 + 6a_3 t \tag{23}$$

If Eqs. (22) and (23) are combined, four equations with four unknown, are obtained. When these equations are solved, formula (24) is obtained [11].

$$\begin{aligned} \theta_0 &= a_0 \\ \theta_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \\ 0 &= a_0 \\ 0 &= a_1 + 2a_2 t_f + 3a_3 t_f^2 \end{aligned} \tag{24}$$

Lastly, by the solution of these equations above, the coefficients are found as below,

$$\begin{aligned} a_0 &= \theta_0 \\ a_1 &= 0 \\ a_2 &= \tfrac{3}{t_f^2}(\theta_f - \theta_0) \\ a_3 &= \tfrac{-2}{t_f^3}(\theta_f - \theta_0) \end{aligned} \tag{25}$$

Using the equations above, the Eq. (26) is derived and used to compute the speed and position reference function. A sample of cubic trajectory is given in Fig. 2.

$$\theta_i(t) = \theta_{i0} + \tfrac{3}{t_f^2}(\theta_{if} - \theta_{i0})t^2 - \tfrac{2}{t_f^3}(\theta_{if} - \theta_{i0})t^3 \quad i = 1, \ldots, m \tag{26}$$

where m is the number of joints.

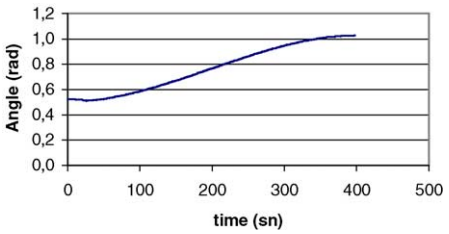

Fig. 2. A sample of cubic trajectory used in this study.

5. Neural network implementation

A backpropagation neural network with sigmoidal activation function is used to solve inverse kinematics problem. The schematic diagram of the system is given in Fig. 3. Firstly, some points in the work volume of manipulator are taken to use in the cubic path planning to generate the ($\theta_1, \theta_2, \theta_3$) joint angles according to different (x, y, z) cartesian coordinates. These values were recorded in a file to form the learning set of the neural network. We obtained the angles (θ_1, θ_2, θ_3) from a certain cubic by sampling 6000 for orbit trajectory a given job. Five thousand of these data was used in training of neural network, and the rest was used in testing. The training process is completed until the error is acceptable. To understand whether the error is acceptable or not, an information about distance that refers to difference between the end effector and the target point is used. This distance can be obtained easily in metric form by using three-dimensional (3D) distance equation between two points in 3D space. As a result, it can be decided that the error is acceptable or not depending on the obtained distance. Here, the sensitivity of the application is also important in the decision of acceptable error. It has been completed approximately in 3,000,000 iterations. The neural network is designed including 40 perceptrons in hidden layer, and 3 perceptrons in input and 3 perceptrons in output layer. Learning rate (η) and the momentum rate (α) are experimentally chosen as 0.3 and 0.8, respectively. The number of perceptrons in hidden layer is also found experimentally. Error at the end of the learning is 0.000121 for training set.

The kinematics model of model manipulator and its direct kinematics equations are given in Fig. 4. The relationship between the world coordinates and joint angles of the manipulator is also given. These equations were used to prepare the training set of the neural network. Also cubic trajectory planning explained in Section 4 is used to generate the points from initial point to target [11].

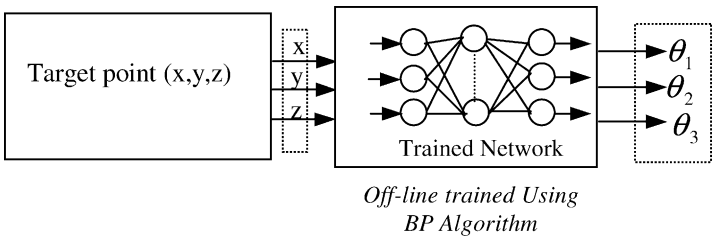

Fig. 3. A schematic diagram of the implemented system.

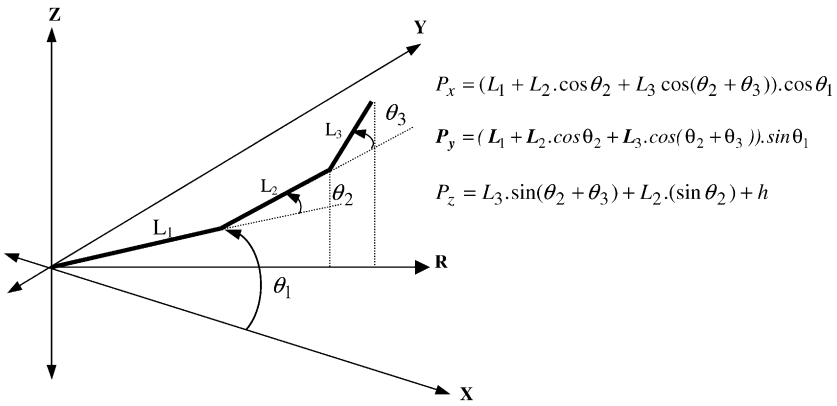

Fig. 4. The kinematics model of manipulator and its direct kinematics equations.

6. Conclusions

In this study, the inverse kinematics solution using neural network for a three-joint robotic manipulator is presented. The neural network is trained until the error is acceptable. It is observed that neural network can be used in inverse kinematics problem. The proposed method results in an acceptable error. For future study, quick backpropagation or radial function, etc. can be used in inverse kinematics solution.

References

[1] N. Abulafya, Neural networks for system identification and control, MSc thesis, University of London, 1995.
[2] P.C.Y. Chen, J.K. Mills, K.C. Smith, Performance improvement of robot continuous-path operation through iterative learning using neural networks, Machine Learn. J. 23 (1996) 191–220.
[3] C.M. Clark, J.K. Mills, Robotic system sensitivity to neural network learning rate: theory, simulation, and experiments, Int. J. Robotic. Res. 19 (10) (2000) 955–968.
[4] T. Çakar, İ. Çil, A. Kurt, Flexible manufacturing system design and determination of priority rules by using artificial neural networks, in: Proceedings of the 8th International Machine Design and Production Conference, Ankara, Turkey, 9–11 September 1998.
[5] J. Duffy, Analysis of Mechanism and Robot Manipulators, Wiley, New York, 1980.
[6] R. Featherstone, Position and velocity transformation between robot end-effector coordinate and joint angle, Int. J. Robotic. Res. 2 (2) (1983) 35–45.
[7] J. Freeman, D. Sakapura, Neural Networks: Algorithms, Applications and Programming Techniques, Reading, Mass, Addison Wesley, 1991.
[8] K.S. Fu, R.C. Gonzalez, C.S.G. Lee, Robotics-Control, Sensing, Vision and Intelligence, McGrow-Hill, 1987.
[9] Z.G. Grudic, P.D. Lawrange, Iterative inverse kinematics with manipulator configuration control, IEEE Trans. Robotic. Autom. 9 (4) (1993).
[10] J.U. Korein, N.I. Balder, Techniques For generating the goal-directed motion of articulated structures, IEEE Comput. Graphics Appl. 2 (9) (1982) 71–81.
[11] R. Köker, Model based intelligent control of a three-joint robot with a vision system, Ph.D. thesis, Sakarya University, 2002.
[12] G.C.S. Lee, Robot arm kinematics, dynamics and control, Computer 15 (12) (1982) 62–79.
[13] D. Manocha, J.F. Canny, Efficient inverse kinematics for general 6r manipulators, IEEE Transact. Robotic. Autom. 10 (5) (1994) 648–657.
[14] C. Öz, R. Kazan, A. Ferikoğlu, N. Yumuşak, An inverse kinematics solution for robotic manipulators with artificial neural networks, in: Prooceedings of the 8th International Machine Design and Production Conference, Ankara TURKEY, 9–11 September 1998.
[15] R.P. Paul, B. Shimano, G.E. Mayer, Kinematic control equations for simple manipulators, IEEE Transact. Syst. Man Cybernetics SMC-11 6 (1981) 66–72.
[16] P.D. Wasserman, Neural Computing, Van Nostrand Reinhold, New York, 1989.

Raşit Köker was born in Kayseri, Turkey in 1972. He received his BS degree in electronic education in 1994, and his MS degree in electronic and telecommunication education in 1996 from Marmara University, Istanbul. During the MS studies, he worked as an electronic teacher in a high school of the Turkish Ministry of Education. In 1996, he has begun his PhD study in Sakarya University. With the beginning of PhD studies he had been working as a research assistant till 2002 at the University of Sakarya, Department of Electronic and Computer Education. He has completed his PhD education in September 2002. Between September 2002 and June 2004, he worked as a lecturer in the Engineering Faculty, Department of Computer Engineering in Sakarya University, and since June 2004 he has been working as an assistant professor in the same department. His research interests include robotics, robot-vision, pattern recognition, and artificial neural networks.

Hüseyin Ekiz was born in Gaziantep, Turkey in 1963. He received his BS degree in electronics education in 1984, and his MS degree in electronic and computer education in 1992 from Gazi University, Ankara. He worked as a research assistant at Gazi University between 1990 and 1993. He did his PhD at the Sussex University, England, between 1993 and 1997. He became associated professor in 1998. He is the dean of Technical Education Faculty of Sakarya University. His research interests include robotics, computer networks, field buses, and microprocessors.

Tarik Cakar was born in Istanbul, Turkey in 1966. He received his BS degree in management engineering in 1988, and his MS degree in management engineering in 1991, and his PhD degree in management engineering in 1997 from Istanbul Technical University, Istanbul. He had been working as a research assistant from 1989 to 1998 at Sakarya University, Industrial Engineering Department. Since February 1998, he has been working as an assistant professor in Engineering Faculty, Department of Industrial Engineering in Sakarya University. His research interests include scheduling, production and operation management, expert systems, artificial neural networks and genetic algorithms.

Cemil Öz was born in Cankiri, Turkey in 1967. He received his BS degree in electronic and communication engineering in 1989 from Yildiz Technical University and his MS degree in electronics and computer education in 1993 from Marmara University, Istanbul. During the MS studies, he worked as a lecturer in Istanbul Technical University. In 1994, he has begun his PhD study in electronics engineering in Sakarya University. He has completed his PhD in September 1998. He has been working as an assistant professor in Engineering Faculty, Department of Computer Engineering in Sakarya University. His research interests include robotics, robot-vision, artificial intelligence and computer programming.