A Neuro-genetic Algorithm Approach for Solving the Inverse Kinematics of Robotic Manipulators [*]

P.Kalra
Department of Production Engineering
Punjab Engineering College
Chandigarh, India
pkalra_chd@yahoo.com

Neelam Rup Prakash
Department of Electronics Engineering
Punjab Engineering College
Chandigarh, India
nprakash_pec@yahoo.co.in

Abstract – *The inverse kinematics solution of a robotic manipulator requires the solution of non-linear equations having transcendental functions and involving time-consuming calculations. Artificial neural networks with their massively parallel architecture are natural candidates for providing a solution to this problem. In this work, a neuro-genetic algorithm approach is used to obtain the inverse kinematics solution of a robotic manipulator. A multi-layered feed-forward neural network architecture is used. The weights of the neural network are obtained during the training phase using a real-coded genetic algorithm. This training algorithm does not suffer from the usual drawbacks of the backpropagation learning algorithm. The approach is used to obtain the inverse kinematics solution of a planar robotic manipulator.*

Keywords: Robot inverse kinematics, neuro-genetic approach, artificial neural networks, real-coded genetic algorithm.

1 Introduction

The inverse kinematics solution of a robotic manipulator determines the joint variables that would result in a desired position of the robot end-effector. This mapping from Cartesian space to joint space requires the solution of non-linear equations having transcendental functions and involving time-consuming calculations. Artificial neural networks (ANN) with their massively parallel architecture are natural candidates for providing a solution to this problem.

Neural network approaches to the inverse kinematics problem of robotic manipulators have been reported in [1], [7] and [8]. The majority of the reported works use the backpropagation algorithm as the learning algorithm for the ANN. The backpropagation algorithm has disadvantages which include getting stuck up in local optima and requirement of smoothness of the thresholding function [6]. Moreover, the neural network may not converge at all if the initial weights are not selected properly. Further, it has been shown that the normalization range of training data and input output mapping also affects the training time and accuracy [2]. The disadvantages of the backpropagation learning algorithm can be overcome by using a genetic algorithm (GA). GAs use a probabilistic approach based on Darwin's theory of evolution and survival of the fittest. They do not require any gradient information, use a population of solutions and have the potential of finding a near optimal solution in complex search spaces [3].

In this work, a neuro-GA approach is used to obtain the inverse kinematics solution of a robotic manipulator. The network used is a multilayered feedforward neural network with a GA as its learning algorithm.The remainder of this paper is organized as follows: Section 2 defines the inverse kinematics problem for a robot. Section 3 gives an overview of the neural network. Section 4 describes the mechanics of the proposed real-coded genetic training algorithm. Section 5 summarises the simulation experiments carried out on a planar robotic manipulator. The final section summarises the main contributions of the proposed strategy.

2 Inverse kinematics problem

For a robot having '*t*' joint variables, the position vector $\left\{{}^{O}P_{h}\right\}$ of the robot hand relative to the base coordinate frame can be extracted from the last column of the matrix $\left[{}^{O}T_{h}\right]$. The $\left[{}^{O}T_{h}\right]$ matrix relates the hand coordinate frame of the robot to the base coordinate frame and can be obtained through the use of DH matrices. The number of degrees of freedom can be reduced to '*t-3*' and the position vector of the robot wrist, $\left\{{}^{O}P_{w}\right\}$, can be obtained as

[*] 0-7803-7952-7/03/$17.00 © 2003 IEEE.

$$\left\{ {}^{O}P_{w} \right\} = \left\{ {}^{O}P_{h} \right\} - \left[{}^{O}R_{t-2} \right] \left\{ {}^{t-2}P_{h} \right\} \qquad (1)$$

where $\left[{}^{O}R_{t-2} \right]$ relates the axes of coordinate frame '*t-2*' to the base coordinate frame and $\left\{ {}^{O}P_{w} \right\}$ is a function of the joint variable vector $\{q\} = \{q_1 \ \ q_2 \ \ \ldots q_{(t-3)} \ \ \}^T$. $\{q\}$ represents the unknown vector of joint variables at the desired configuration of the robotic manipulator.

In order to achieve a desired position $\left\{ {}^{O}P_{w,des} \right\}$ of the robot wrist, eqn. (1) can be used to form a vector equation of the form

$$\left\{ {}^{O}P_{w} \right\} - \left\{ {}^{O}P_{w,des} \right\} = \{0\} \qquad (2)$$

The limits on the joint variable values can be expressed as

$$q_k^L \le q_k \le q_k^U \qquad k = 1,2,\ldots,t-3 \qquad (3)$$

where q_k^L and q_k^U represent the lower and upper limit of joint variable '*k*' respectively.

The inverse kinematics problem can thus be stated as

$$\begin{aligned} & Evaluate\{q_1 \ \ q_2 \ \ \ldots q_{(t-3)} \ \ \}^T \ subject\ to \\ & \left\| \left\{ {}^{O}P_{w} \right\} - \left\{ {}^{O}P_{w,des} \right\} \right\| = 0 \\ & q_k^L \le q_k \le q_k^U \qquad k = 1,2,\ldots,t-3 \end{aligned} \qquad (4)$$

Eqn. (4) requires the solution of a non-linear equation having transcendental functions and involving time-consuming calculations. Artificial neural networks (ANN) with their massively parallel architecture are natural candidates for providing a solution to this problem.

3 Details of neural network

3.1 Network architecture

The desired position coordinates of the robot wrist would constitute the inputs to the neural network. The network output would be the joint variables at the desired configuration of the robot. A multi-layered feedforward neural network (MFNN) with no feedback connections can be used for this purpose. This network architecture is shown in fig. 1.

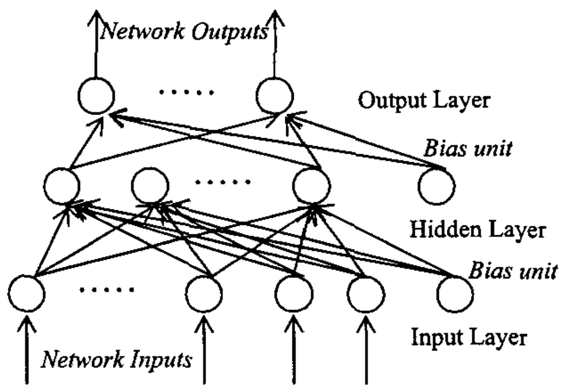

Figure 1. Neural network architecture

3.2 Network operational equations

Let the neural network input and output vector be represented as

$$\{p_i\} = \left\{ p_{i,1} \ p_{i,2} \cdots p_{i,n_1} \right\}^T \in R^{n_1} \qquad (5)$$

$$\{q_i\} = \left\{ q_{i,1} \ q_{i,2} \cdots q_{i,n_m} \right\}^T \in R^{n_m} \qquad (6)$$

Here n_k represents the number of neurons in the k^{th} layer, *i* refers to the i^{th} vector training pair and *m* refers to the output layer. The augmented output vector of the k^{th} layer is given by

$$\left\{ out_{ia}^k \right\} = \left\{ out_{i,1}^k \ out_{i,2}^k \ \ldots \ out_{i,n_k}^k \ \ 1 \right\} \qquad (7)$$

where

$$\begin{aligned} & out_{i,r}^1 = p_{i,r} && r = 1,2,\ldots,n_1 \\ & out_{i,r}^k = f\left(net_{ir}^k\right) && r = 1,2,\ldots,n_k \quad k = 2,3,\ldots,m-1 \\ & out_{i,r}^m = net_{ir}^m && r = 1,2,\ldots,n_m \end{aligned} \qquad (8)$$

and

$$net_{ir}^k = \sum_{j=1}^{n_{k-1}} w_{r,j}^{k-1} \ out_{i,j}^{k-1} + \mu_r^k \qquad (9)$$

Here $w_{r,j}^{k-1}$ is the weight from the j^{th} neuron of the *(k-1)*th layer to the r^{th} neuron of the k^{th} layer, μ_r^k is the threshold of the r^{th} neuron of the k^{th} layer and *f(.)* is the neuron's non-linear activation function. It should be noted that the values of $out_{i,r}^m$ in eqn. (8) are in fact the elements of $\{q_i\}$ evaluated by the neural network.

The non-linear activation function *f(.)* is taken as the unipolar sigmoid function

$$f(x) = \frac{1}{1+e^{-\lambda x}} \qquad (10)$$

Here λ is proportional to the neuron gain determining the steepness of the activation function near $x=0$.

3.3 Generation of training pairs

The training pairs for the network learning are generated using the inverse kinematics problem definition given in eqn. (4). Random values of $\{q\}$ satisfying eqn.(3) constitute the output training pairs of the network. These values of $\{q\}$ are substituted in eqn. (1) to evaluate $\{{}^{O}P_{w}\}$. The evaluated $\{{}^{O}P_{w}\}$ form the corresponding input training pairs. The network is trained for these training pairs using the algorithm described in the next section.

4 Real-coded genetic training algorithm

4.1 Individual representation

The success of a genetic algorithm for a specific problem depends on the basic mechanism used to represent an individual in the population. When a binary coding is used for continuous search space, a number of difficulties arise. Binary coding of the variables associated with an individual is therefore not advisable for continuous search spaces [4]. For the neural network training, the individual in a population would be represented by $\{w\}$, the vector consisting of the weights and threshold values of the network neurons. This vector can be represented as $\{w_{1,1}^{k-1}\ w_{2,1}^{k-1} \cdots w_{n_k,1}^{k-1} \cdots w_{1,n_{k-1}}^{k-1}\ w_{2,n_{k-1}}^{k-1} \cdots w_{n_k,n_{k-1}}^{k-1}\ \mu_1^k\ \mu_2^k \cdots \mu_{n_k}^k\}$ with $k = 2,3,\ldots m$.

4.2 Mechanics of genetic algorithm

The components of a GA include initialisation, iterative selection made on the basis of fitness, recombination and mutation. A flow chart showing the mechanics of a GA is shown in fig. 2.

4.2.1 Initialisation

The initial values of the weights and threshold values are chosen in the closed interval [-3, 3]. An initial population of vectors containing the weight and threshold values of the network neurons is generated by random sampling from the variable search space. The goal is to select individuals spread over the entire search space.

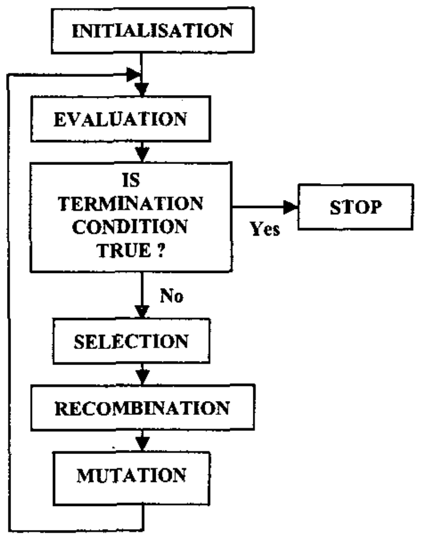

Figure 2. Mechanics of a GA

4.2.2 Evaluation and selection

The fitness of an individual is a measure of how good a solution it provides to the training pairs of the network. The aim is to generate values of network weights and threshold values such that the errors in the determination of the output training pairs are minimum. The error measure can be mathematically represented as

$$Error = \sum_{i=1}^{N} \left\| \{q_{i,neural}\} - \{q_{i,actual}\} \right\| \qquad (11)$$

where $\{q_{i,actual}\}$ are the actual output training pairs and $\{q_{i,neural}\}$ are obtained using eqns. (8) and (9) with $k = m$. Eqn. (11) itself can be taken as a measure of an individual fitness. The fitness function to be minimized is therefore defined as

$$Fitness = \sum_{i=1}^{N} \left\| \{q_{i,neural}\} - \{q_{i,actual}\} \right\| \qquad (12)$$

Fitness proportional selection methods have a scaling problem. Here, the raw fitness values have to be mapped within a range of scaled fitness values in order to have an appropriate level of selection pressure. The tournament selection operator does not have this scaling problem. In the current work, the binary tournament selection operator is used with fitness being evaluated according to its definition given in eqn. (12).

4.2.3 Recombination and mutation

A number of real-coded crossover operators have been developed that create two children solutions from two parent solutions. In most cases, a probability distribution centring the parent solutions is assumed and two children solutions are created based on that probability distribution. Creating children solutions using a fixed probability distribution, which does not depend on the location of the parent solutions, makes the search adaptive. For example, if the parent solutions are close to each other, the children solutions are expected to lie in the neighbourhood of the parent solutions. On the other hand, if the parent solutions are far away from each other, children solutions far away from the parent solutions are expected. In early generations of a GA simulation, parent solutions are expected to be away from each other and almost any solution can be created as a child solution. But when the search converges towards a solution, parent solutions become similar and children solutions also become closer to the parent solutions. This adaptiveness in the search power of the real-coded crossover operator is similar in principle to the search power of the binary-coded single-point crossover operator. However, one main difference is that for the binary-coded crossover operator, no explicit probability distribution is used to create a child solution. But there is an implicit probability distribution that depends on the string length used to code the variable. In a real-coded crossover operator, a probability distribution is explicitly used to create a child solution. Since an explicit probability distribution is used, the performance of real-coded GAs depends on that distribution. In this work, a SBX (Simulated Binary Crossover) operator suggested in [4] with a probability distribution similar to that in the single-point crossover operator used in binary-coded GAs is used. The following procedure is used to create two children solutions, $\left(w_{n,c}^{(1)}, w_{n,c}^{(2)}\right)$ from two parent solutions $\left(w_{n,p}^{(1)}, w_{n,p}^{(2)}\right)$:

1. Create a random number u between 0 and 1.
2. Calculate spread factor parameter β' from a schema processing point of view as

$$\begin{aligned} \beta' &= (2u)^{\frac{1}{\eta+1}} && , u \le 0.5 \\ &= \left(\frac{1}{2-2u}\right)^{\frac{1}{\eta+1}} && , \textit{otherwise} \end{aligned} \tag{13}$$

3. Create two children solutions as follows

$$\begin{aligned} w_{n,c}^{(1)} &= 0.5\left[\ \left(w_{n,p}^{(1)} + w_{n,p}^{(2)}\right) - \beta'\left|w_{n,p}^{(2)} - w_{n,p}^{(1)}\right|\ \right] \\ w_{n,c}^{(2)} &= 0.5\left[\ \left(w_{n,p}^{(1)} + w_{n,p}^{(2)}\right) + \beta'\left|w_{n,p}^{(2)} - w_{n,p}^{(1)}\right|\ \right] \end{aligned} \tag{14}$$

For creating children solutions in situations where lower and upper bounds of variables are specified as given in section 4.2.1 above, eqn. (14) is suitably modified as suggested in [4].

The mutated values of the i^{th} component of the weight and threshold vector are evaluated as

$$w_{n,m}^{(c+1)} = w_n^c + \delta\left[w_n^U - w_n^L\right] \tag{15}$$

The parameter δ is evaluated using the equations suggested in [5] for bounded decision variables. This ensures a perturbance of order $O(1/\eta_m)$ where the parameter η_m and the probability of mutation p_m are evaluated as

$$\eta_m = 100 + c \tag{16}$$

$$p_m = \frac{1}{s} + \left(\frac{c}{c_{\max}}\right)\left(1 - \frac{1}{s}\right) \tag{17}$$

Here s is the number of variables and c and $c_{\max}$ are the current generation number and the maximum number of generations allowed respectively. Thus, in the initial generation, one variable is mutated on an average with an expected 1% perturbance and more variables are mutated with less perturbance as generations proceed.

5 Results

Simulation experiments were performed on the planar robotic manipulator shown in fig. 3. The link lengths (a_1 and a_2) are taken as 50cm and 40cm respectively and the limits on the joint variables are taken as $-2.0 \le q_1$ (in rad) ≤ 2.0 and $-2.53 \le q_2$ (in rad) ≤ 2.53.

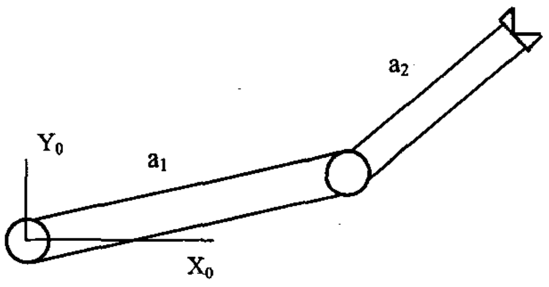

Figure 3. Planar robotic manipulator

For this robotic manipulator, the location of the robot wrist is given as

$$\left\{{}^{O}P_w\right\} = \begin{Bmatrix} a_1\cos(q_1) + a_2\cos(q_1+q_2) \\ a_1\sin(q_1) + a_2\sin(q_1+q_2) \end{Bmatrix} \tag{18}$$

A 2-3-2 MFNN was trained to provide the inverse kinematics solution of the planar manipulator. Training pairs for the neural network were generated using the procedure specified in Section 3.3. The input and output training pairs in Cartesian space and joint variable space are shown in figs. 4 and 5 respectively.

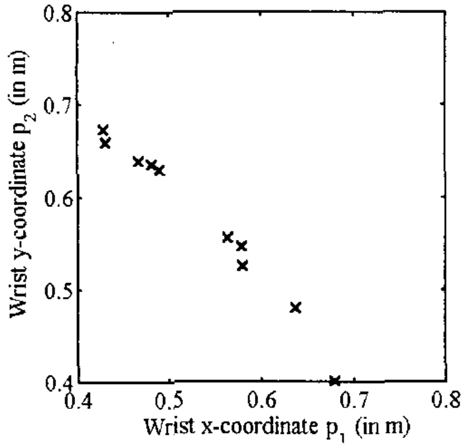

Figure 4. Input training pairs in Cartesian space

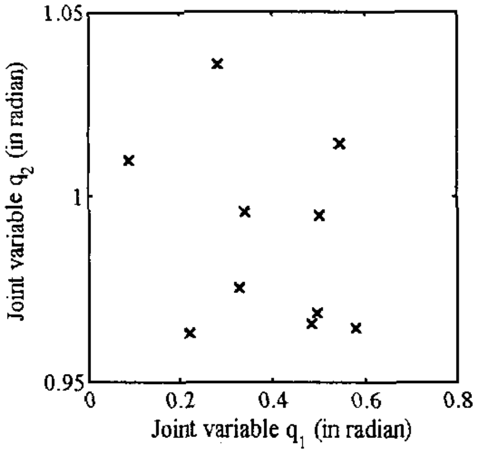

Figure 5. Output training pairs in joint space

The following control parameters are used for the GA: Population size = 170, Cross-over probability = 0.9, SBX parameter η = 1, c_{max} (for purpose of calculation of mutation probability) = 800. The termination condition was taken as an absolute relative error of less than 1% for all the joint variables of the robot in each of the training pairs. Fig. 6 shows the variation of fitness of the best individual in the population with generations. The best fitness values show a rapid decrease with the termination condition of training being achieved in 100 generations of the GA.

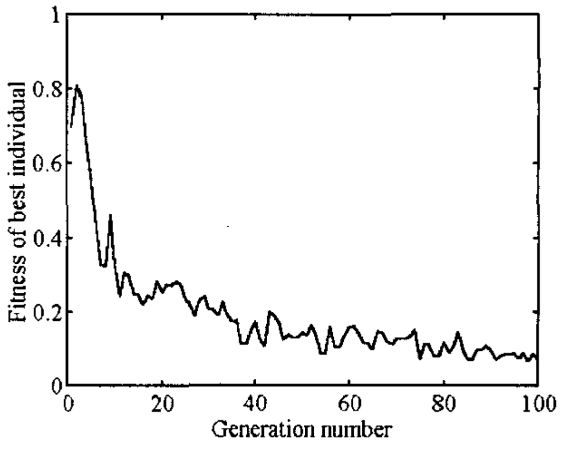

Figure 6. Variation of fitness of best individual with generations

The vector $\{w\}$ consisting of the weights and threshold values of the network neurons obtained at the end of training phase are shown in Table 1.

Table 1. Weights and threshold values of network neurons

	n=1	n=2	n=3
$w^1_{n,1}$	0.0061	-1.557	0.215
$w^1_{n,2}$	2.9999	1.377	0.967
μ^2_n	-1.870	-0.420	-1.401
$w^2_{n,1}$	-0.566	-1.164	-
$w^2_{n,2}$	2.573	1.051	-
$w^2_{n,3}$	2.999	0.138	-
μ^3_n	-1.327	0.868	-

The network was subsequently evaluated and used to obtain the inverse kinematics solution of the planar manipulator. The network response is compared with the desired response and the relative errors of the joint variables are evaluated for different points in the manipulator workspace. The results of the evaluation phase are given in Table 2.

Table 2 shows that the GA achieved an absolute relative error of less than 2 % in the determination of the joint variables in all the evaluations. These errors are small and the neuro-genetic algorithm approach is, therefore, acceptable for obtaining the inverse kinematics solution of the robotic manipulator.

Table 2. Results of network evaluation

S.No.	Network input (in cm)		Network output (in rad)		Absolute relative % error in	
	Wrist x-coordinate	Wrist y-coordinate	q_1	q_2	q_1	q_2
1	0.403	0.686	0.612	0.970	0.36	0.88
2	0.541	0.582	0.382	0.966	1.58	1.98
3	0.482	0.635	0.488	0.962	1.17	1.10
4	0.645	0.454	0.173	0.997	1.99	1.37
5	0.609	0.504	0.248	0.982	1.16	1.82
6	0.511	0.609	0.436	0.963	1.24	1.79
7	0.428	0.672	0.575	0.966	0.11	0.87
8	0.412	0.678	0.595	0.971	0.86	1.92

6 Conclusions

Artificial neural networks with their massively parallel architecture are natural candidates for providing a solution to the inverse kinematics problem of robotic manipulators. The backpropagation training algorithm suffers from certain shortcomings like getting struck up in the local optima. A real-coded genetic algorithm was used to train a multi-layered feedforward neural network for providing the inverse kinematics solution of a robotic manipulator. The training algorithm used was a real-coded GA employing a simulated binary crossover and a parameter based mutation approach. The network response was then compared with the desired response and the relative errors of the joint variables were evaluated for different points in the manipulator workspace. These errors are small and the neuro-genetic algorithm approach is, therefore, acceptable for obtaining the inverse kinematics solution of a robotic manipulator.

References

[1] F.J. Arteaga-Bravo, "Multi-layer backpropagation network for learning the forward and inverse kinematic equations", *Proceedings of IEEE International Neural Network Conference*, Vol. II, pp. 319-321, 1990.

[2] D.K. Chaturvedi, P.S. Satsangi and P.K. Kalra, "Effect of different mappings and normalizations on neural network models", *9th National Power Systems Conference*, Vol. 1, pp. 377-382, 1996.

[3] K. Deb, *Optimization for engineering design*, Prentice Hall of India, New Delhi, 1995.

[4] K. Deb and R.B. Agrawal, "Simulated binary crossover for continuous search space", *Complex Systems*, Vol. 9, pp. 115-148, 1995.

[5] K. Deb and M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design", *Computer Science and Informatics*, Vol. 26, No. 4, pp. 30-45, 1996.

[6] L. Fu, *Neural networks in computer intelligence*, McGraw Hill Inc., New York, pp. 81-91, 1994.

[7] A. Guez and Z. Ahmad, "Solution to the inverse kinematics problem in robotics by neural networks", *Proceedings of the II International Conference on Neural Networks, San Diego*, Vol. 2, pp. 617-624, 1988.

[8] B. Karlik and S. Aydin, "An improved approach to the solution of inverse kinematics problems for robot manipulators", *Engineering Applications of Artificial Intelligence*, Vol. 13, pp. 159-164, 2000.