Robotica (1997) volume 15, pp 617–625. Printed in the United Kingdom © 1997 Cambridge University Press

Finding the inverse kinematics of manipulator arm using artificial neural network with lookup table

A.S. Morris and A. Mansor

Department of Automatic Control & Systems Engineering, University of Sheffield, Mappin St., Sheffield SI 3JD (UK)

(Received in Final Form: January 14, 1997)

SUMMARY

Neural networks were used to find the inverse kinematics of a two-link planar and three-link manipulator arms. The neural networks utilised were multi-layered perceptions with a back-propagation training algorithm. Because of the redundancy in the manipulators studied, this work used lookup tables for the different configurations of the manipulator arm.

KEYWORDS: Manipulator arm; Inverse kinematics; Neural network; Redundancy; Lookup table.

1. INTRODUCTION

The forward and inverse kinematic solutions for manipulator arms are computer intensive and time consuming, especially for many degrees of freedom manipulator arm. Colbaugh et al.[1] proposed an adaptive algorithm based on Model Reference Adaptive Control (MRAC) to solve the inverse kinematics problem of manipulator. Arteaga-Bravo[2] used a multi-layered back propagation network to solve the equations for forward and inverse kinematics. Even though he modelled a two-link planar manipulator, the training of the artificial neural network for the inverse kinematics solution was done on a simulated single-link manipulator. Guo et al.[2] proposed an approach based on the Jacobian control technique for the solution of the inverse kinematics. They used a Hopfield neural network to find the solutions of the inverse kinematics of a planar four degree-of-freedom manipulator. The end effector traversed a straight line trajectory in the *xy*-coordinate frame.

Whatever inverse kinematics algorithm is used for redundant manipulators, there are many solutions that are available for any single point or position of the end effector. The work presented here is an extension to the solution of inverse kinematics using artificial neural network. The inverse kinematics of a manipulator arm is solved by using a feedforward multi-layered perceptron with back-propagation algorithm for the training session. The network is then trained with data for a number of end effector positions expressed in Cartesian co-ordinates and the corresponding joint angles. The data consists of the different configurations available for the arm. For any position of the end effector in Cartesian space, lookup tables of the weights for the artificial neural network are created for each of the configurations or orientations of the arm. The lookup tables[4] can then be used to plan the trajectory of the arm for whichever configuration of the arm is required.

2. THEORY

The work described is based on two manipulator arm configurations. One is a two-link planar manipulator and the other is a three-link manipulator arm in three-dimensional space. For both the two-link and three-link manipulators, there are two orientations or poses that are possible for every position of the end effector in Cartesian space. The different poses of the arm are then used to train a three-layer, fully connected back-propagation model (Figure 1). Thus, this gave two sets of weights for each manipulator arm after the training session was over.

The network is trained until it gives an error of about 2% or less for the difference between an actual and desired output, the error value of 2% having been chosen because it was sufficient for this work. The weights of the connections are saved as lookup tables. This lookup tables can then be used, depending on the orientations of the manipulator required, to find the joint angles of the arm given the position of the end effector in Cartesian space.

A block diagram of the proposed work is shown in Figure 2.

The signals, o_{ji}, are presented to a hidden layer neuron in the network via the input neurons. Each of the signals from the input neurons are multiplied by the value of the weights of the connection, w_j, between the respective input neurons and the hidden neuron. The net input to a hidden neuron is calculated as the sum of the values for all connections coming into the neuron.

$$\text{net-input}_{(\text{hidden neuron})} = i_h = \sum_{j=1}^{n} w_j \times o_{ji} \qquad (1)$$

for n inputs.

The output, o_{kj}, of a hidden neuron as a function of its net input is described in equation 2. The sigmoid function is:

$$\text{output} = o_{kj} = \frac{1}{1 + e^{-i_h}} \qquad (2)$$

Once the outputs of the hidden layer neurons have been calculated, the net input to each output layer is calculated in a similar manner as in equation 1.

During the training phase, the feedforward output

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press

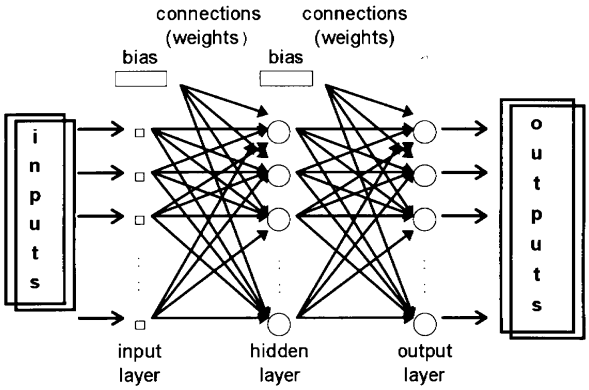


Fig. 1. A model of the 3 layer perceptron neural network.

state calculation is combined with backward error propagation and weight adjustment calculations that represent the network's learning, or training. Equation 3 presents the definition of the error signal for the ouptut layer neurons.

$$\delta = f'(i)(t - o_k) \tag{3}$$

and from equation 2:

$$\delta_l = o_k(1 - o_k)(t - o_k) \tag{4}$$

where t is the target or desired value, and o_k is the actual value from output neuron after going through the feedforward calculation. The error calculation was implemented on a neuron-by-neuron basis over the entire set (epoch) of patterns. This error value, δ, was used to perform the appropriate weight adjustments of the weight connection between the output layer and hidden layer. The error value from equation 5 was used for the weight adjustments between the hidden layer and input layer.

$$\delta_h = f'(i_h) \sum_{l=0}^{n_l} w_{lh}\delta_l = o_h(1 - o_h) \sum_{l=0}^{n_l} w_{lh}\delta_l \tag{5}$$

where δ_h is the error value of the hidden layer, δ_l is the error value of the output layer, o_h is the output of the sigmoid function and w_{lh} is the connection weights between the output and hidden layers.

The weight changes were calculated according to equation 6:

$$w(\text{old}) = w(\text{new}) + \eta\delta o + \alpha[\Delta w(\text{old})] \tag{6}$$

The aim of the training phase is to minimise this average sum squared error over all training patterns. The speed of convergence of the network depends on the training rate, η, and the momentum factor, α.

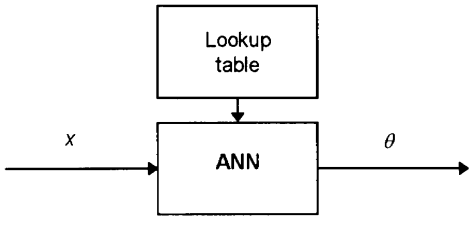


Fig. 2. A block diagram of the system.

The back-propagation algorithm provides an approximation to the trajectory in weight space computed by the method of steepest descent. The smaller the learning rate, η, the smaller will the changes to the synaptic weights in the network be from one iteration to the next, and the smoother will be the trajectory in weight space. This improvement, however, is at the expense of a slower rate of learning. However, if the learning rate parameter is made large so as to speed up the rate of learning, the resulting large changes may result in the network becoming unstable, i.e. oscillatory. A simple method of increasing the rate of learning and yet avoiding the danger of instability is to include a momentum term, α. The other problem with this kind of weight updating is that it sometimes gets caught in local energy minima. By multiplying the momentum term with the previous weight change, the network can overcome the pull of the local minima and continues its gradient descent towards the globally optimal solution.

3. TWO-LINK PLANAR MANIPULATOR ARM

Figure 3 shows the two-link planar manipulator arm simulated and Table I shows its joint parameters. Both the link lengths are taken to be 1. The equations that relate the Cartesian position of the end effector with the joint angles of the manipulator arm are given in equation 7.

$$x = \cos(\theta_1 + \theta_2) + \cos(\theta_1) \tag{7a}$$

$$y = \sin(\theta_1 + \theta_2) + \sin(\theta_1) \tag{7b}$$

giving

$$\theta_1 = \text{atan } 2(y, x) - \text{atan } 2(k_2, k_1) \tag{8a}$$

$$\theta_2 = \text{atan } 2(\sin\theta_2, \cos\theta_2) \tag{8b}$$

where $k_1 = 1 + \cos\theta_2$

$$k_2 = \sin\theta_2$$

$$\cos\theta_2 = (x^2 + y^2 - 2)/2$$

$$\sin\theta_2 = \pm\sqrt{1 - \cos^2\theta_2}$$

A single hidden layer neural network with two inputs, x and y, and two outputs, θ_1 and θ_2, was trained using the back-propagation algorithm described earlier, along a trajectory of the end effector in the xy-plane. The final weights, when the error results were less than or equal to 2%, were saved as lookup tables to be used later as a check of the system on different points along that trajectory.

Tables IIa and IIb show the training data or patterns

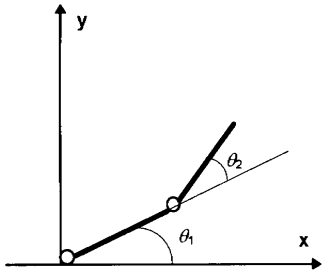


Fig. 3. A model of the two-link planar manipulator arm.

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press

Table I. Joint parameters of the two-link planar manipulator arm

Joint i	θ_i	a_i	α_i	d_i
1	θ_1	1	0	0
2	θ_2	1	0	0

used during the training stage of the two orientations of the two-link manipulator arm.

3.1 Results for the two-link manipulator arm

The trajectory of the end effector that was used to train the back-propagation neural network in two-dimensional space is shown in Figures 4a and 4b. Both the figures give the orientation of the arm while traversing the same path or trajectory. For the first orientation, the hidden layer has 5 neurons, the rate of training, η, was 2.018, and the momentum factor, α, was 0.54.

The training of the 9 patterns was done 3000 times. The average percentage error, i.e. the percentage of the difference between actual and targeted/desired output, at the final iteration of the final session was 0.02% for the first joint and 0.3% for the second joint. Figure 5 shows the graph of the training session done on the arm for the first orientation. An average sum squared error is taken as an indication of the average error to offset any negative number present in the error. The average sum squared errors obtained were 0.7% for the first joint and 1.2% for the second joint and 1.1% for the first joint and 0.7% for the second joint, for the first and second orientation respectively. Figures 6a and 6b shows the difference between the actual and desired outputs for the different joints for the first orientation.

As a check, a neural network with 6 hidden neurons

Table II. The normalised training data of the two-link manipulator arm for (a) the first orientation, (b) the second orientation

Operational space		Joint space	
x	*y*	θ_1^*	θ_2^{**}
1.0	0.55	0.6665	0.8759
0.9	0.60	0.7215	0.9085
0.8	0.65	0.7780	0.9358
0.7	0.70	0.8356	0.9573
0.6	0.75	0.8936	0.9726
0.5	0.80	0.9508	0.9814
0.5	0.85	0.9519	0.9592
0.6	0.90	0.9010	0.9085
0.7	0.95	0.8525	0.8543

Operational space		Joint space	
x	*y*	θ_1^*	θ_2^*
1.0	0.55	0.6787	0.8759
0.9	0.60	0.6414	0.9085
0.8	0.65	0.5891	0.9358
0.7	0.70	0.5173	0.9573
0.6	0.75	0.4169	0.9726
0.5	0.80	0.2596	0.9814
0.5	0.85	0.1269	0.9592
0.6	0.90	0.1287	0.9085
0.7	0.95	0.0628	0.8543

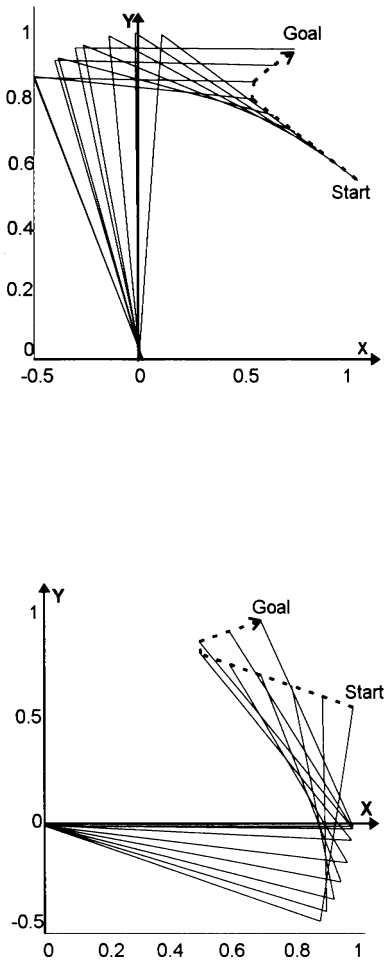


Fig. 4. The planar manipulator arm traversing a path used for training in (a) first orientation (b) second orientation.

was used for the training, with all other parameters remaining the same. The average sum squared error were found to be 0.6% and 1.3% for the first and second link respectively.

The number of hidden neurons were randomly chosen for the neural network. There are no specific method or algorithm to choosing the number of hidden neurons at the initial stage.[5,6] Then the learning parameter and the momentum factor were initialised to small random numbers. These two numbers were then adjusted (by increasing or decreasing the value) to give the smallest error value of the average sum squared error at the output of the network. Table III shows the average sum

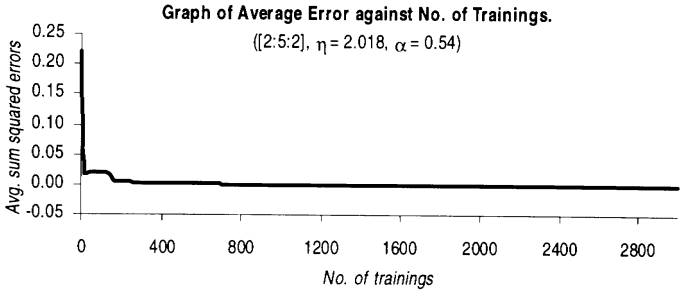
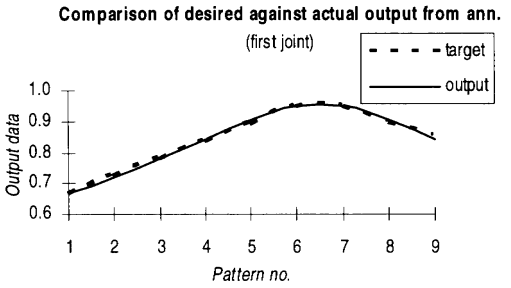


Fig. 5. A graph of the training session for first orientation of the two-link arm.

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press



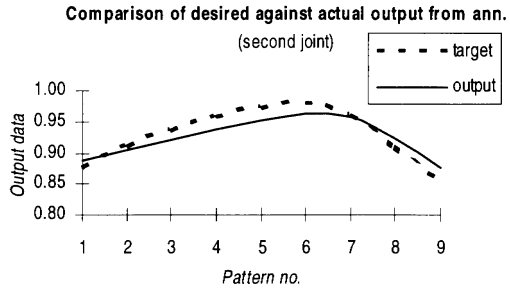


Fig. 6. Comparison of target and actual outputs of the neural network training session of the two-link arm for (a) first joint of first orientation (b) second joint of first orientation.

squared error with different values of the learning parameter and momentum factor for the two-link arm in first orientation. The number of iterations done was 3000. The testing of the arm was done on different points of the same trajectory as the training, and the average error was found to be 0.01% for the first joint and 0.9% for the second joint. Figure 7 shows the graph of both the actual and target output for the two joints for first orientation.

The same was done for the second orientation using a

Table III. A table of the different values of the learning term and momentum factor with their corresponding error for the first orientation of the two-link arm

η	α	error
1.000	0.54	0.000419
1.500	0.54	0.000173
1.900	0.54	0.000107
2.018	0.54	0.000102
2.018	0.3	0.000248
2.018	0.4	0.000158
2.018	0.5	0.00011
2.018	0.6	0.00011
2.018	0.7	0.00062
2.020	0.54	0.000102
2.010	0.54	0.000102
2.200	0.54	0.000105
2.005	0.54	0.000103
2.018	0.53	0.000104
2.018	0.55	0.000102
2.500	0.54	0.000162
3.000	0.54	0.003146

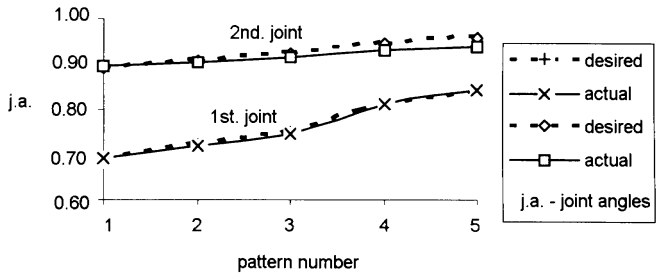


Fig. 7. Comparison between actual and desired outputs during the test session of the two joints for first orientation of the two-link arm.

6 neurons hidden layer. The training rate was at 2.13 and the momentum factor was at 0.23. As before, the number of hidden neurons, the learning rate and momentum factor were initally chosen randomly. They were then adjusted accordingly to get the smallest error output from the network. Figure 8 shows the graph of the training session done on the arm for the first. Figures 9a and 9b show the difference between the actual and desired outputs for the different joints for the second orientation. Again the average error of the first joint was 0.6% and 0.1% for the second joint. The average error for the testing session was 0.7% for the first joint and 0.1% for the second joint. Figure 10 shows the graph of both the actual and desired output for the two joints for second orientation. The second orientation of the arm needs more neurons for the training stage because of the wide distribution of the output data. The network was found to need more training either by increasing the number of hidden neurons or iterations when the data or patterns are widely distributed.

4. THREE-LINK MANIPULATOR ARM IN 3-D SPACE

Figure 11 shows the three-link manipulator arm for which the link parameters are shown in Table IV. Again the link lengths are taken to be 1 as above, to simplify the calculations. The arm can move around in three-dimensional workspace. Again, an artificial neural network, with three inputs and outputs, and a single hidden layer was used for the training session. The path that was used to train the neural network is the trajectory of the end effector in three-dimensional workspace as

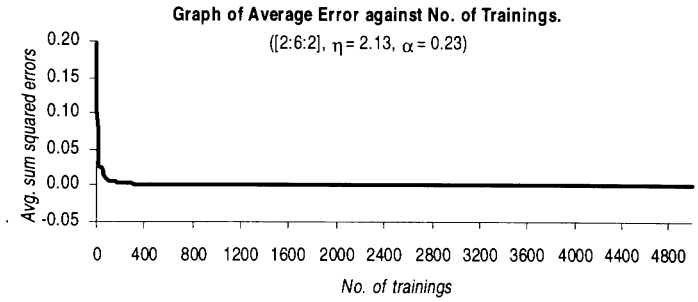
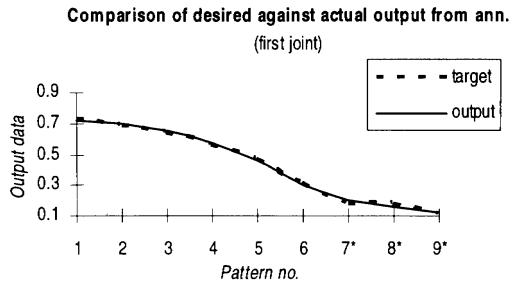


Fig. 8. A graph of the training session for second orientation of the two-link arm.

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press



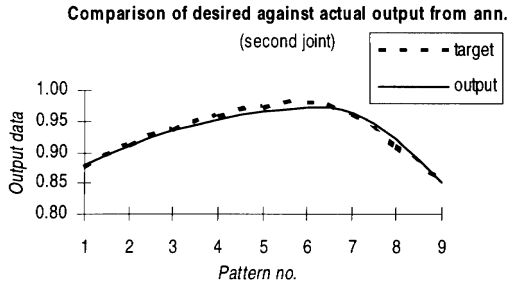


Fig. 9. Comparison of target and actual outputs of the neural network training session of the two-link arm for (a) first joint of second orientation (b) second joint of second orientation.

shown in Figures 12a and 12b for the first and second orientation of the arm respectively. Tables V(a) and V(b) shows the respective data used during the training of the neural network.

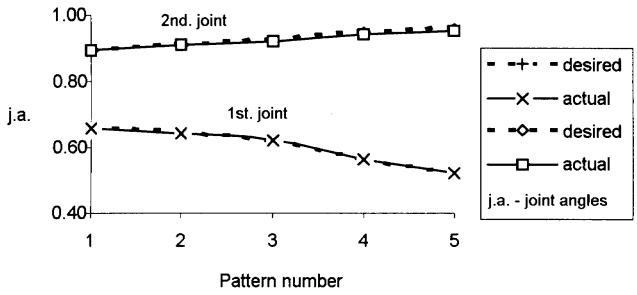


Fig. 10. Comparison between actual and desired outputs during the test session of the two joints for second orientation.

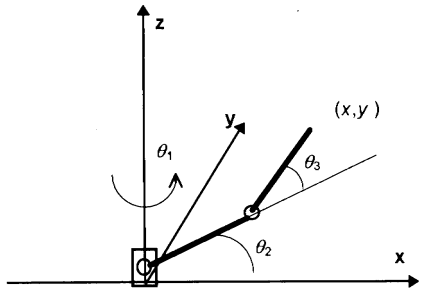


Fig. 11. A model of the three-link manipulator arm.

Table IV. Joint parameters of the three-link manipulator arm

Joint i	θ_i	a_i	α_i	d_i
1	θ_1	0	90	0
2	θ_2	1	0	0
3	θ_3	1	0	0

The equations relating the xyz-coordinates with the joint angles are given in equation 9:

$$x = \cos\theta_1 \times [\cos(\theta_2 + \theta_3) + \cos\theta_2] \quad (9a)$$

$$y = \sin\theta_1 \times [\cos(\theta_2 + \theta_3) + \cos\theta_2] \quad (9b)$$

$$z = \sin(\theta_2 + \theta_3) + \sin\theta_2 \quad (9c)$$

thus giving:

$$\theta_1 = \text{atan } 2(y, x) \quad (10a)$$

$$\theta_2 = \text{atan } 2(z, x/\cos\theta_1) - \text{atan } 2(k_2, k_1) \quad (10b)$$

$$\theta_3 = \text{atan } 2(\sin\theta_3, \cos\theta_3) \quad (10c)$$

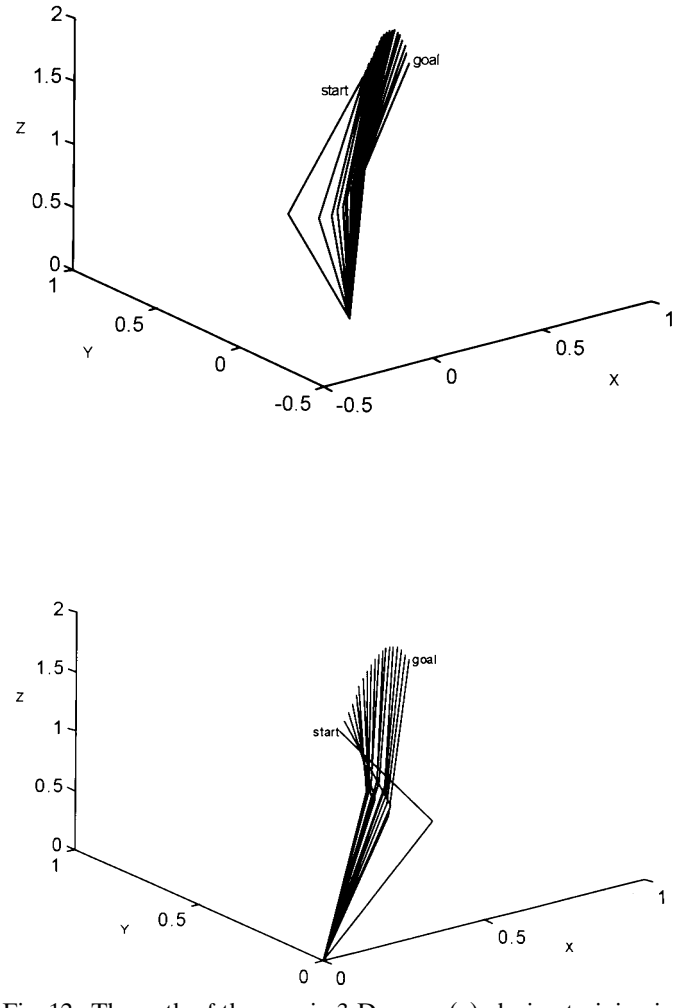


Fig. 12. The path of the arm in 3-D space (a) during training in first orientation and (b) in second orientation.

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press

Table V. The normalised training data of the three-link manipulator arm for (a) the first orientation, (b) the second orientation

x	y	z	θ_1	θ_2	θ_3
0.05	0.00	0.950	0.00000	0.93060	0.42227
0.10	0.05	0.950	0.23182	0.91293	0.41850
0.15	0.10	0.948	0.29400	0.89260	0.41223
0.20	0.15	0.946	0.32175	0.87158	0.40499
0.25	0.20	0.942	0.33737	0.85065	0.39840
0.30	0.25	0.936	0.34737	0.83040	0.39408
0.35	0.30	0.929	0.35431	0.81135	0.39361
0.40	0.35	0.918	0.35941	0.79390	0.39839
0.45	0.40	0.904	0.36332	0.77828	0.40943
0.50	0.45	0.887	0.36641	0.76450	0.42726
0.55	0.50	0.867	0.36891	0.75231	0.45184
0.60	0.55	0.842	0.37097	0.74128	0.48265
0.65	0.60	0.813	0.37271	0.73075	0.51881
0.70	0.65	0.778	0.37419	0.71994	0.55918
0.75	0.70	0.739	0.37546	0.70797	0.60249
0.80	0.75	0.694	0.37658	0.69381	0.64729
0.85	0.80	0.643	0.37755	0.67634	0.69197
0.90	0.85	0.585	0.37842	0.65429	0.73462
0.95	0.90	0.521	0.37919	0.62625	0.77301

x	y	z	θ_1	θ_2	θ_3
0.05	0.00	0.950	0.00000	0.81852	0.42227
0.10	0.05	0.950	0.46365	0.79874	0.41850
0.15	0.10	0.948	0.58800	0.77791	0.41223
0.20	0.15	0.946	0.64350	0.75712	0.40499
0.25	0.20	0.942	0.67474	0.73580	0.39840
0.30	0.25	0.936	0.69474	0.71312	0.39408
0.35	0.30	0.929	0.70863	0.68819	0.39361
0.40	0.35	0.918	0.71883	0.66014	0.39839
0.45	0.40	0.904	0.72664	0.62827	0.40943
0.50	0.45	0.887	0.73282	0.59207	0.42726
0.55	0.50	0.867	0.73782	0.55124	0.45184
0.60	0.55	0.842	0.74195	0.50572	0.48265
0.65	0.60	0.813	0.74542	0.45552	0.51881
0.70	0.65	0.778	0.74838	0.40074	0.55918
0.75	0.70	0.739	0.75093	0.34146	0.60249
0.80	0.75	0.694	0.75315	0.27779	0.64729
0.85	0.80	0.643	0.75510	0.20982	0.69197
0.90	0.85	0.585	0.75683	0.13777	0.73462
0.95	0.90	0.521	0.75838	0.06199	0.77301

where $k = 1 + \cos\theta_3$

$$k_2 = \sin\theta_3$$
$$\cos\theta_3 = (x^2 + y^2 + z^2 - 2)/2$$
$$\sin\theta_3 = \pm\sqrt{1 - \cos^2\theta_3}$$

4.1 Results for the three-link manipulator arm

The training of the first orientation was done using a three layer back-propagation neural network with one hidden layer. The hidden layer had 6 neurons. For the first orientation, η was 0.71 and α was 0.055. These two values gave the smallest output error of the neural network during the training stage. The average percentage error was 0.7%, 0.01% and 0.002% for the first, second and third joint respectively when the training was stopped at 30,000 iterations. The number of patterns used for the training was 19. Training was stopped at 30,000 iterations because the maximum size integer possible on a PC is 32,000.

Figure 13 and 14 shows the graph of the training session on the 19 patterns of the neural network for the first and second orientations respectively. Figure 15 and 16 shows the output data compared to the target data for all the joints in first and second orientation respectively.

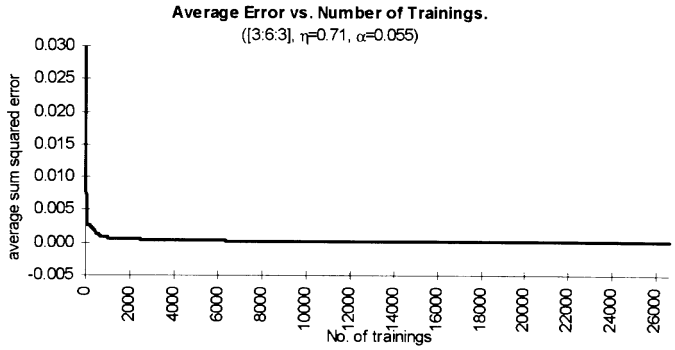


Fig. 13. A graph of the training session for first orientation of the three-link arm.

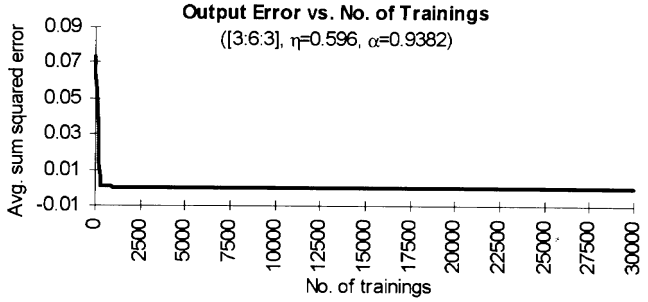


Fig. 14. A graph of the training session for second orientation of the three-link arm.

The average percentage error when using the lookup table for the testing was found to be 2.4%, 0.1% and 0.5% for the first, second and third joints, respectively. The test was done on 15 different points lying on the same trajectory as the training path. Figure 17 shows the difference between the actual and the desired outputs for all three joints in first configurations or orientations in the testing session. Table VI shows the percentage error

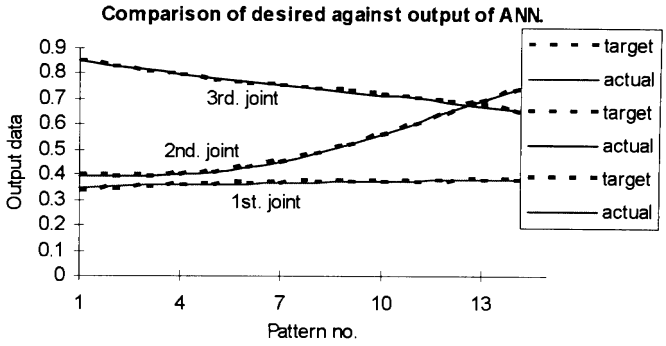


Fig. 15. Comparison between actual and desired outputs of three joints for first orientation in 3-D for the train patterns.

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press

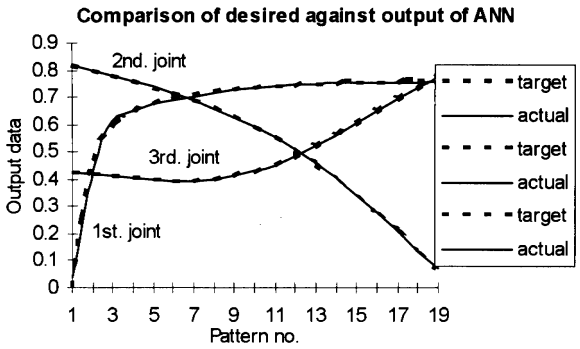


Fig. 16. Comparison between actual and desired outputs of three joints for second orientation in 3-D for the train patterns.

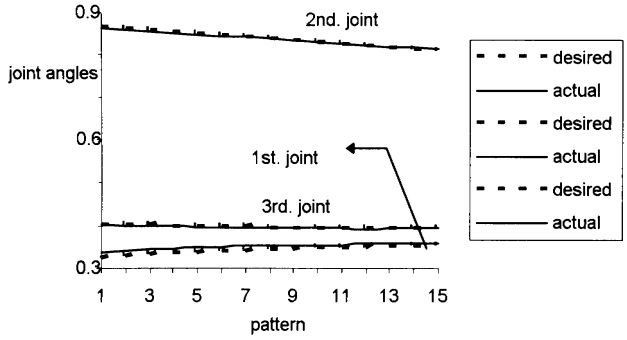


Fig. 17. Comparison between actual and desired outputs of three joints for first orientation in 3-D for the test patterns.

value for each train pattern for joint 1 of the first orientation. Even though the average percentage error was 0.7%, the percentage pattern error for the first 4 patterns was quite large. The neural network was not trained long enough to give a good generalisation of the first 4 patterns. Table VII shows the percentage error value for each test pattern. As can be seen, those patterns that fall in the training pattern range (Table VI: about 0.33 and less) gave large percentage pattern error. This was why the average percentage error for the first

Table VI. Train patterns with pattern error for the first joint of the first orientation. (* division by zero)

Desired	Actual	% error
0	0.05576	*
0.23182	0.189439	**-18.28**
0.294	0.291069	**-1.00**
0.32175	0.333577	**3.68**
0.33737	0.34859	**3.33**
0.34737	0.35404	**1.92**
0.35431	0.356595	**0.64**
0.35941	0.358418	**-0.28**
0.36332	0.360277	**-0.84**
0.36641	0.362406	**-1.09**
0.36891	0.364859	**-1.10**
0.37097	0.36757	**-0.92**
0.37271	0.37047	**-0.60**
0.37419	0.373368	**-0.22**
0.37546	0.376085	**0.17**
0.37658	0.378297	**0.46**
0.37755	0.379675	**0.56**
0.37842	0.379839	**0.37**
0.37919	0.378537	**-0.17**

Table VII. Test patterns with pattern error for the first joint of the first orientation

Desired	Actual	% error
0.32554	0.337986	**3.82**
0.32895	0.341523	**3.82**
0.33203	0.344385	**3.72**
0.33482	0.346705	**3.55**
0.33737	0.34859	**3.33**
0.33971	0.35013	**3.07**
0.34186	0.351395	**2.79**
0.34384	0.352444	**2.50**
0.34567	0.353322	**2.21**
0.34737	0.354068	**1.93**
0.34895	0.354683	**1.64**
0.35043	0.355249	**1.38**
0.35181	0.355728	**1.11**
0.3531	0.35619	**0.88**
0.35432	0.356595	**0.64**

link of the first orientation of the three-link manipulator arm was quite large, 2.4%.

A similar network was used for the second orientation. The training rate, η, was 0.596 and the momentum factor, α, was 0.932. The average percentage error for the first, second and third joints were 0.1%, 0.6% and 0.001%, respectively. After the test was done, the average percentage error was found to be 0.5%, 0.2% and 0.3% for the first, second and third joints. Figure 18 shows difference in outputs for all the joints for the second orientation in the testing session.

4.1.1 Effect of increasing the training period. These results suggested that a larger training period might produce better results. As the maximum integer size on a PC does not allow a longer training session, i.e. more than 30,000 iterations, the work was repeated on a SUN workstation for the three-link manipulator arm in the first orientation with 45,000 iterations during the training period. The average percentage error for the first link is now 1.1%. Table VIII shows the new percentage error value for each test pattern. A graph of the training session, the training results and the test results are shown in Figures 19, 20 and 21, respectively.

5. DISCUSSION AND CONCLUSION

For a multiple degree-of-freedom manipulator arm, there is redundancy and a particular end effector position can

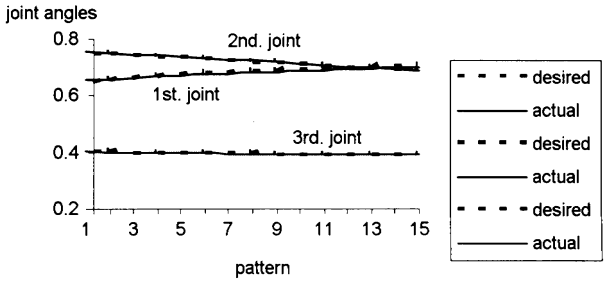


Fig. 18. Comparison between actual and desired outputs of three joints for second orientation in 3-D for the test patterns.

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press

Table VIII. Test patterns with pattern error for the first joint of the first orientation with 45,000 iterations

target	actual	% error
0.32554	0.332058	2.002
0.32895	0.334684	1.743
0.33203	0.336832	1.446
0.33482	0.338617	1.134
0.33737	0.340124	0.816
0.33971	0.34142	0.503
0.34186	0.342556	0.204
0.34384	0.343572	-0.078
0.34567	0.344497	-0.339
0.34737	0.345354	-0.580
0.34895	0.346135	-0.807
0.35043	0.346904	-1.006
0.35181	0.347621	-1.191
0.3531	0.348345	-1.347
0.35432	0.34903	-1.493

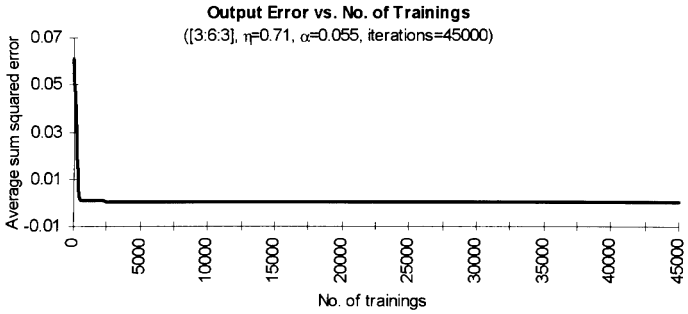


Fig. 19. A graph of the training session for first orientation for 45,000 iterations for the three-link arm.

be reached from many orientations of the arm. The aim of the work described here was to use the neural network as lookup tables for such redundant manipulators. (Tables IX and X).

The results presented show that the lookup tables worked well. The average error for the difference between the actual and the desired outputs is below 2% (Figures 9a, 9b and 10), except for the first joint of the first orientation of the three link manipulator arm due to the short training time (Figures 6 and 7). The result could have been better for all joints in all configurations if the training time had been lengthened. This was not possible as the simulation was done on an IBM-compatible personal computer, as the number of iterations is limited to a maximum of 32,000 iterations. This is probably due to the PC under MS-DOS being a 16 bit system. The largest integer size is 2^{16}, which is 65,536. With two's complement, for negative values, this drops down to about 32,000.

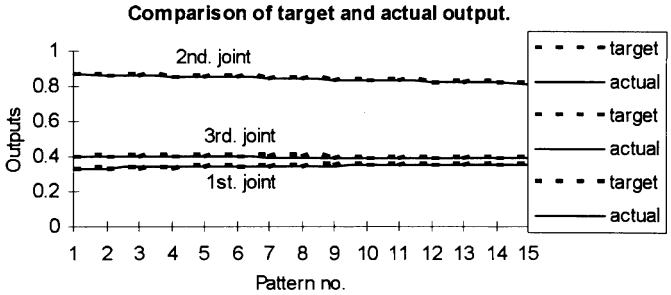


Fig. 21. Comparison between actual and desired outputs of three joints for first orientation in 3-D for the test patterns (45,000 iterations).

In a real working environment this would not matter, as the training would have been done off-line. The training of the new workspace could be done once the manipulator has been installed. All the possible orientations of the arm are used in the training of the network, and the corresponding weights then saved as lookup tables.

Future work will address the limitations identified so far, in particular shorter training time.

Table IX. The error value taken for the test pattern of the two-link manipulator arm for first and second joints for (a) the first orientation, and (b) the second orientation

θ_1^*	θ_1^+	Error %	θ_2^*	θ_2^+	Error %
0.694	0.691	-0.4	0.893	0.897	0.4
0.722	0.718	-0.5	0.909	0.905	-0.3
0.750	0.747	-0.4	0.923	0.914	-0.9
0.807	0.810	0.4	0.947	0.932	-1.7
0.836	0.842	0.8	0.957	0.940	-1.8
* Calculated value (target).			+ Output from ann (output).		

θ_1^*	θ_1^+	Error %	θ_2^*	θ_2^+	Error %
0.662	0.659	-0.5	0.893	0.894	0.2
0.641	0.643	0.2	0.909	0.909	0.1
0.617	0.623	0.9	0.923	0.923	0.0
0.556	0.565	1.6	0.947	0.945	-0.3
0.517	0.523	1.2	0.957	0.953	-0.4
* Calculated value (target).			+ Output from ann (output).		

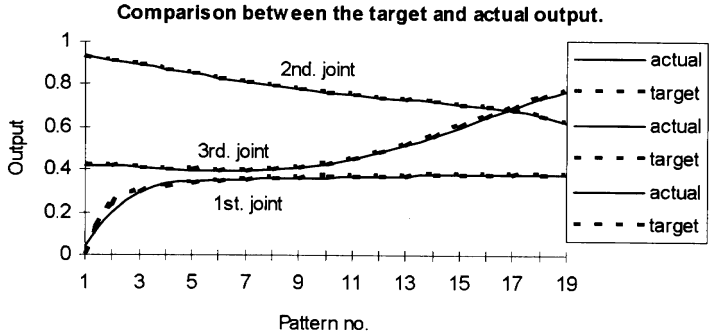


Fig. 20. Comparison between actual and desired outputs of three joints for first orientation in 3-D for the train patterns (45,000 iterations).

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press

Table X. The error value taken for the test pattern for the second orientation of the three-link manipulator arm for first, second and third joints.

desired	actual	error	desired	actual	error	desired	actual	error
Θ_1	Θ_1	%	Θ_2	Θ_2	%	Θ_3	Θ_3	%
0.651	0.656	**0.8**	0.753	0.755	**0.3**	0.404	0.402	**-0.5**
0.658	0.660	**0.4**	0.749	0.751	**0.3**	0.402	0.400	**-0.5**
0.664	0.664	**0.1**	0.744	0.747	**0.3**	0.401	0.399	**-0.5**
0.670	0.668	**-0.2**	0.740	0.742	**0.3**	0.400	0.398	**-0.5**
0.675	0.672	**-0.4**	0.736	0.738	**0.3**	0.398	0.397	**-0.5**
0.679	0.675	**-0.6**	0.731	0.733	**0.2**	0.397	0.396	**-0.4**
0.684	0.679	**-0.7**	0.727	0.728	**0.2**	0.396	0.395	**-0.4**
0.688	0.682	**-0.8**	0.722	0.724	**0.2**	0.395	0.394	**-0.3**
0.691	0.685	**-0.9**	0.718	0.719	**0.2**	0.395	0.394	**-0.3**
0.695	0.688	**-0.9**	0.713	0.714	**0.1**	0.394	0.393	**-0.2**
0.698	0.691	**-1.0**	0.708	0.709	**0.1**	0.394	0.393	**-0.1**
0.701	0.694	**-0.9**	0.703	0.704	**0.0**	0.393	0.393	**-0.1**
0.704	0.697	**-0.9**	0.698	0.698	**0.0**	0.393	0.393	**0.0**
0.706	0.700	**-0.9**	0.693	0.693	**0.0**	0.393	0.393	**0.0**
0.709	0.703	**-0.9**	0.688	0.688	**-0.1**	0.394	0.394	**0.1**

The inverse kinematics solution of a manipulator by using neural network can also be applied to flexible manipulators. The neural network can be trained by using the position of the actual end effector of the flexible manipulator in Cartesian co-ordinate frame against the joint angles. And again this can be saved as lookup tables. The inverse kinematics solution of any end effector position can then be found by giving the position in Cartesian frame into the system. As the flexibility of the manipulator was already included in the training session, the result obtained would also give the position of the joint angles with the flexibility included.

An extension to this work will be done at a later stage where the architecture of the neural network will be the same for the lookup tables, i.e. the same number of hidden neurons for all the lookup tables. Then there is no need to use a different neural network for the different lookup tables. The same neural network architecture can be used for all the different orientations of the arm by just using the corresponding lookup tables.

References

1. R. Colbaugh, K. Glass & H. Seraji, "An Adaptive Inverse Kinematics Algorithm for Robot Manipulators" *Int. J. Modelling & Simulation* **11,** No. 2, 33–38 (1991).
2. F.J. Arteaga-Bravo, "Multilayer back-propagation network for learning the forward and inverse kinematics equations" *Proc. Joint 1990 IEEE Int. Neural Network Conf.* (1990) pt. 2, pp. 319–321.
3. J. Guo & V. Cherkassky, "A solution to the inverse kinematics problem in robotics using neural network processing." *Proc. Int. Conf. On Neural Network* (1989) pp. II299–II304.
4. H. Ritter, T. Martinetz & K. Schulten, *Neural Computation and Self-organising Maps: An Introduction.* (Addison-Wesley, Reading, Mass., 1992).
5. Simon S. Haykin, *Neural Networks, a Comprehensive Foundation.* (Maxwell Macmillan Int., London, 1994).
6. Russell C. Eberhart & Roy W. Dobbin, *Neural Network PC Tools: a practical guide.* (Academic Press, New York, 1990).

https://doi.org/10.1017/S026357479700074X Published online by Cambridge University Press